1 Preface

SDF-A (System Dialog Facility-Administration) is a software product for administration and modification of the BS2000 user interface, which is implemented using SDF.
SDF (System Dialog Facility) requires, for the internal processing of inputs, syntax descriptions; these are kept in special files, called syntax files. These syntax files can be arranged in a hierarchical structure with a maximum of three levels (see section “File hierarchy” on page 33).

The specification of the user interface simultaneously defines the scope of the system. By modifying the user interface, it is therefore possible, for example, to provide specific users with additional privileges or to restrict the rights and authority granted to certain users. SDF-A can be used to process the user interface accordingly.

This manual describes how to use SDF-A to create new syntax files or to modify existing syntax files.

The results of processing a syntax file with SDF-A become effective as soon as this syntax file is activated.

1.1 Target group

This manual is intended primarily for system administration and experienced BS2000 users. It provides, on the one hand, a description of the features and applications of SDF-A. On the other hand, it acts as a reference manual for all SDF-A statements and macros and for the interface between SDF and high-level languages.

The manual also contains a description of SDF-SIM (System Dialog Facility-SIMulator). SDF-SIM is designed for testing commands and statements in a defined environment of syntax files.

We recommend the following manuals as preparatory reading matter and for more detailed information about SDF:

– "Introductory Guide to the SDF Dialog Interface" [1]
  This manual describes the dialog interface implemented with SDF and explains how to enter SDF commands and statements as well as commands and standard statements belonging directly to SDF.

– "SDF Management" [2]
  This manual describes the commands for controlling SDF, the installation and management of syntax files, and the statements for SDF-I, SDF-U and SDF-PAR.

– "Commands" [4], Volumes 1 through 6
  These manuals describe all BS2000/OSD commands in SDF format, including the commands reserved for system administration.

– "Introductory Guide to Systems Support" [6]
  This manual describes the tasks to be performed by system operators and system administrators.

There is also a "Ready Reference" [3] for SDF-A. This contains the SDF-A statements, the macros and the function calls for the interface between SDF and high-level programming languages, and the SDF-SIM statements. This Ready Reference is intended as a reference work for experienced users.

1.2 Summary of contents

Chapter 2, “Syntax files and their processing with SDF-A”, is intended as an introduction to the concept of SDF-A and to the basic principles for working with SDF-A.

Detailed examples illustrating the use and features of SDF-A can be found in chapters 3, “Modifying command and statement definitions”, and 4, “Definition and implementation of commands and statements by the user”.

The SDF-A statements, the SDF program interface, and SDF-SIM are described in separate chapters, where they are listed in alphabetical order. This part of the manual is intended for use as a reference section.

README file

Functional changes and supplements to the current product version not documented in this manual are explained in the product-specific README file. The name of this file on your BS2000 host is SYSRME.*product.version.language* and is SYSRME.SDF-A.041.E for SDF-A V4.1. The user ID under which the README file is cataloged can be obtained from your systems support group. The full path name is also output using the following command:

```
    /SHOW-INSTALLATION-PATH INSTALL-UNIT=SDF-A,LOGICAL-IDENTIFIER=SYSRME.E
```

You can view the README file with the `SHOW-FILE` command or in an editor, or you can use the following command to print the contents of the file on a standard printer:

```
/PRINT-DOCUMENT filename, LINE-SPACING=*BY-EBCDIC-CONTROL
```

1.3 Changes since the last version of the manual

The manual for SDF-A V4.1E contains the following changes since the last version of the manual (SDF-A V4.1A, published in June, 1996):

- SHOW-STMT (standard statement in SDF V4.4A and higher)
- The standard statements are not described any more in this manual. You will find detailed descriptions of the standard statements in the “Introductory Guide to the SDF Dialog Interface“ user guide [1].
- The following statements have been changed:

Statement	**Changes**
ADD-CMD, MODIFY-CMD	IMPLEMENTOR=*PROCEDURE(...) structure: – in the NAME=*BY-IMON(...) structure: new ELEMENT operand – new UNLOAD-PROGRAM operand In the *YES(...) and *NO(...) structures of the DIALOG-, DIALOG-PROC-, GUIDED-, BATCH-, BATCH-PROC- and CMD-ALLOWED operands: – PRIVILEGE=*ALL eliminated (corresponds to *SAME); still supported for compatibility reasons, though
ADD-STMT, MODIFY-STMT	New STMT-VERSION operand
DEFINE-ENVIRONMENT	PARAMETER-FILE=*NO(...) structure: – new SYSTEM=*CURRENT operand value

- The description of the interface to higher programming languages has been revised. Of special note is that the interface to higher programming languages only supports the standardized transfer area in the old format (see also the section “Changes to the SDF program interface” on page 593ff), i.e. the functionality of the “old” macro calls RDSTMT, CORSTMT and TRSTMT. The extended functionality of the corresponding “new” macro calls CMDRST, CMDCST and CMDTST (e.g. statement return code) for the standardized transfer area in the new format is only available through the Assembler interface. The example of using the C interface has been updated accordingly.

SDF-A V4.1E can be run in BS2000/OSD V1.0 and higher (with SDF V4.1B and higher). However, when the full scope of the functions are used, then the syntax files created can only be used in BS2000/OSD V3.0 and higher.

1.4 Notes on the user interface

The statements described in this manual are processed by the command processor SDF (System Dialog Facility). The product thus offers various forms of guided or unguided dialog with the facility for requesting help menus for the statements and commands. In the case of an incorrect input, a correction dialog can be executed.
Detailed information on various options provided by SDF can be found in the “Introductory Guide to the SDF Dialog Interface” [1].

Abbreviation of names

SDF permits inputs to be abbreviated in interactive and batch modes and in procedures, provided the abbreviations used are unambiguous within the related syntax environment. Note, however, that an abbreviation that is currently unambiguous could potentially become ambiguous at a later date if new functions are added to the product. For this reason, it is best to avoid abbreviations entirely, or at least to ensure that only guaranteed abbreviations are used. In the statement formats, these guaranteed abbreviations are shown in boldface.

Command and statement names, operands and keyword values may be abbreviated as follows:

– Complete name components may be omitted from right to left; the hyphen preceding the dropped name component is also omitted.

– Individual characters of a name component may be omitted from right to left.

– An asterisk (*) preceding a keyword value is not considered a valid abbreviation for that value. As of SDF V4.0A, keyword values are always represented with a leading asterisk. The asterisk may be omitted only if there is no possible alternative variable operand value with a value range that includes the name of the keyword value. This form of abbreviation may be restricted due to extensions in later versions. For reasons of compatibility, operand values that were previously represented without an asterisk are still accepted without the asterisk.

Example

Unabbreviated command format:

```
/MODIFY-SDF-OPTIONS SYNTAX-FILE=*NONE,GUIDANCE=*MINIMUM
```

Abbreviated command format:

```
/MOD-SDF-OPT SYN-F=*NO,GUID=*MIN
```

The guaranteed abbreviations are only intended as recommendations for abbreviated input; they may not always be the shortest possible input in your syntax environment. They are, however, clear and easy to understand and are designed to remain unique in the long term.

In some cases, an additional abbreviation is documented in the manual next to the command or statement name. This abbreviation is implemented as an alias for the command or statement name and is guaranteed in the long term. The alias consists of a maximum of 8 letters (A...Z) that are derived from the command or statement name. Aliases cannot be abbreviated further.

Default values

Most operands are optional, i.e. need not be explicitly specified. Such operands are preset to a specific operand value, the so-called default value. In the syntax, the default value for each operand is shown underscored. If an optional operand is not explicitly specified, its default value is automatically inserted when executing the command or statement. If some default values do not correspond to the values most suitable for your purposes, you can temporarily assign your individual default values for your task as of SDF V4.1A.

Positional operands

SDF permits operands to be specified either as keyword operands or as positional operands. However, it is quite possible that the positions of operands may change in future versions of the product. It is therefore advisable to avoid the use of positional operands, especially in procedures.

XHCS support

Every terminal works with a particular character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Extended character sets can be used when working with the software product XHCS. SDF interprets inputs in accordance with the standard code table **EBCDIC.DF.03** (e.g. when converting uppercase/ lowercase letters).
The coding of the following characters in an extended character set must be the same as the coding used in the standard code table:

$, # , @ , ! , ” , ? , ^ , = , : , / , * , - , (,) , [,] , < , > , comma, semicolon, single quote.

SDF does not interpret any additional characters from an extended character set unless they appear within the data types <c-string> and <text>. In other words, the conversion of uppercase/lowercase letters is handled using a code table supplied by XHCS for the extended character set. Any additional characters used within other data types are rejected as syntax errors.

When statements are entered, the CCS used is the one specified in the appropriate macro (RDSTMT/CMDRST, TRSTMT/CMDTST or CORSTMT/CMDCST). If no CCS was specified, the one used for entering commands is assumed.

For further details on XHCS support, see the manuals “Introductory Guide to the SDF Dialog Interface” [1] and “XHCS” [11].

1.5 Metasyntax

1.5.1 Notational conventions

The following notational conventions are used in this manual:

ℹ for notes

⚠ for warnings

Runtime examples are shown in `typewriter script.` Inputs expected from the user are highlighted in **`bold`**.

In the text, references to other publications are given as abbreviated titles with numbers enclosed in square brackets [ ]. All references that follow refer to BS2000/OSD-BC V5.0. The complete title of each publication to which reference is made is listed under “Related publications” at the back of the manual. This is followed by information on how to order manuals.

1.5.2 SDF syntax description

This syntax description is valid for SDF V4.5A.The syntax of the SDF command/statement language is explained in the following three tables.

Table 1: Notational conventions

The meanings of the special characters and the notation used to describe command and statement formats are explained in table 1.

Table 2: Data types

Variable operand values are represented in SDF by data types. Each data type represents a specific set of values. The number of data types is limited to those described in table 2.

The description of the data types is valid for the entire set of commands/statements. Therefore only deviations (if any) from the attributes described here are explained in the relevant operand descriptions.

Table 3: Suffixes for data types

Data type suffixes define additional rules for data type input. They contain a length or interval specification and can be used to limit the set of values (suffix begins with *without*), extend it (suffix begins with *with*), or declare a particular task mandatory (suffix begins with *mandatory*). The following short forms are used in this manual for data type suffixes:

cat-id	cat
completion	compl
correction-state	corr
generation	gen
lower-case	low
manual-release	man
odd-possible	odd
path-completion	path-compl
separators	sep
temporary-file	temp-file
underscore	under
user-id	user
version	vers
wildcard-constr	wild-constr
wildcards	wild

The description of the ‘integer’ data type in table 3 contains a number of items in italics; the italics are not part of the syntax and are only used to make the table easier to read.
For special data types that are checked by the implementation, table 3 contains suffixes printed in italics (see the *special* suffix) which are not part of the syntax.

The description of the data type suffixes is valid for the entire set of commands/statements. Therefore only deviations (if any) from the attributes described here are explained in the relevant operand descriptions.

Metasyntax

Representation	Meaning	Examples
UPPERCASE LETTERS	Uppercase letters denote keywords (command, statement or operand names, keyword values) and constant operand values. Keyword values begin with *.	**HELP-SDF** **SCREEN-STEPS** = <u>***NO**</u>
UPPERCASE LETTERS in boldface	Uppercase letters printed in boldface denote guaranteed or suggested abbreviations of keywords.	**GUID**ANCE-**MODE** = ***Y**ES
=	The equals sign connects an operand name with the associated operand values.	**GUID**ANCE-**MODE** = <u>***NO**</u>
< >	Angle brackets denote variables whose range of values is described by data types and suffixes (see tables 2 and 3).	**SYNTAX-F**ILE = <filename 1..54>
<u>Underscoring</u>	Underscoring denotes the default value of an operand.	**GUID**ANCE-**MODE** = <u>***NO**</u>
/	A slash serves to separate alternative operand values.	**NEXT-FIELD** = <u>***NO**</u> / ***Y**ES
(…)	Parentheses denote operand values that initiate a structure.	,**UNGUID**ED-**DIA**LOG = <u>***Y**ES</u> (...) / ***NO**
[ ]	Square brackets denote operand values which introduce a structure and are optional. The subsequent structure can be specified without the initiating operand value.	**SELECT** = [***BY-ATTR**IBUTES](...)

Indentation	Indentation indicates that the operand is dependent on a higher-ranking operand.	,**GUID**ED-**DIA**LOG = <u>***Y**ES</u> (...)
		  <u>***Y**ES</u>(...)
		    | **SCREEN-STEPS** = <u>***NO**</u> /
		    | ***Y**ES

Table 1: Metasyntax (part 1 of 2)

Representation	Meaning	Examples
|	A vertical bar identifies related operands within a structure. Its length marks the beginning and end of a structure. A structure may contain further structures. The number of vertical bars preceding an operand corresponds to the depth of the structure.	**SUP**PORT = ***TAPE**(...) ***TAPE(...)** | **VOL**UME = ***ANY**(...) | ***ANY**(...) | | ...
,	A comma precedes further operands at the same structure level.	**GUID**ANCE-**MODE** = ***NO** / ***Y**ES ,**SDF-COM**MANDS = ***NO** / ***Y**ES
list-poss(n):	The entry “list-poss” signifies that a list of operand values can be given at this point. If (n) is present, it means that the list must not have more than n elements. A list of more than one element must be enclosed in parentheses.	list-poss: ***SAM** / ***ISAM** list-poss(40): <structured-name 1..30> list-poss(256): ***OMF** / ***SYSLST**(...) / <filename 1..54>
Alias:	The name that follows represents a guaranteed alias (abbreviation) for the command or statement name.	**HELP-SDF** Alias: **HPSDF**

Table 1: Metasyntax (part 2 of 2)

Data types

Data type	**Character set**	**Special rules**
alphanum-name	A…Z 0…9 $, #, @	
cat-id	A…Z 0…9	Not more than 4 characters; must not begin with the string PUB
command-rest	freely selectable	
composed-name	A…Z 0…9 $, #, @ hyphen period catalog ID	Alphanumeric string that can be split into multiple substrings by means of a period or hyphen. If a file name can also be specified, the string may begin with a catalog ID in the form :cat: (see data type filename).
c-string	EBCDIC character	Must be enclosed within single quotes; the letter C may be prefixed; any single quotes occurring within the string must be entered twice.
date	0…9 Structure identifier: hyphen	Input format: yyyy-mm-dd yyyy: year; optionally 2 or 4 digits mm: month dd: day
device	A…Z 0…9 hyphen	Character string, max. 8 characters in length, corresponding to a device available in the system. In guided dialog, SDF displays the valid operand values. For notes on possible devices, see the relevant operand description.
fixed	+, - 0…9 period	Input format: [sign][digits].[digits] [sign]: + or - [digits]: 0...9 must contain at least one digit, but may contain up to 10 characters (0...9, period) apart from the sign.

Table 2: Data types (part 1 of 6)

Data type	Character set	Special rules
filename	A…Z 0…9 $, #, @ hyphen period	Input format: $\text{[:cat:][\$user.]}\left\{\begin{array}{l}\text{file}\\ \text{file(no)}\\ \text{group}\\ \\ \text{group}\left\{\begin{array}{l}\text{(*abs)}\\ \text{(+rel)}\\ \text{(-rel)}\end{array}\right\}\end{array}\right\}$:cat: optional entry of the catalog identifier; character set limited to A...Z and 0...9; maximum of 4 characters; must be enclosed in colons; default value is the catalog identifier assigned to the user ID, as specified in the user catalog. $user. optional entry of the user ID; character set is A…Z, 0…9, $, #, @; maximum of 8 characters; first character cannot be a digit; $ and period are mandatory; default value is the user's own ID. $. (special case) system default ID file file or job variable name; may be split into a number of partial names using a period as a delimiter: $\text{name}_1\text{[.name}_2\text{[...]]}$ name_i does not contain a period and must not begin or end with a hyphen; file can have a maximum length of 41 characters; it must not begin with a $ and must include at least one character from the range A...Z.

Table 2: Data types (part 2 of 6)

Data type	**Character set**	**Special rules**
filename (contd.)		#file (special case) @file (special case) # or @ used as the first character indicates temporary files or job variables, depending on system generation. file(no) tape file name no: version number; character set is A...Z, 0...9, $, #, @. Parentheses must be specified. group name of a file generation group (character set: as for "file") $\text{group}\left\{\begin{matrix}\text{(*abs)}\\\text{(+rel)}\\\text{(-rel)}\end{matrix}\right\}$ (*abs) absolute generation number (1-9999); * and parentheses must be specified. (+rel) (-rel) relative generation number (0-99); sign and parentheses must be specified.
integer	0…9, +, -	+ or -, if specified, must be the first character.
name	A…Z 0…9 $, #, @	Must not begin with 0...9.

Table 2: Data types (part 3 of 6)

Data type	**Character set**	**Special rules**
partial-filename	A…Z 0…9 $, #, @ hyphen period	Input format: [:cat:][$user.][partname.] :cat: see filename $user. see filename partname optional entry of the initial part of a name common to a number of files or file generation groups in the form: $name_1$.[$name_2$.[...]] $name_i$ (see filename). The final character of “partname” must be a period. At least one of the parts :cat:, $user. or partname must be specified.
posix-filename	A...Z 0...9 special characters	String with a length of up to 255 characters; consists of either one or two periods or of alphanumeric characters and special characters. The special characters must be escaped with a preceding \ (backslash); the / is not allowed. Must be enclosed within single quotes if alternative data types are permitted, separators are used, or the first character is a ?, ! or ^. A distinction is made between uppercase and lowercase.

posix-pathname	A...Z
	0...9
	special characters
	structure identifier: slash
		Input format: [/]$part_1$/.../$part_n$
		where $part_i$ is a posix-filename;
		max. 1023 characters;
		must be enclosed within single quotes if alternative data types are permitted, separators are used, or the first character is a ?, ! or ^.

Table 2: Data types (part 4 of 6)

Data type	**Character set**	**Special rules**
product-version	A…Z 0…9 period single quote	Input format: [[C]’][V][m]m.naso[’] (so: correction status; a: release status) where m, n, s and o are all digits and a is a letter. Whether the release and/or correction status may/must be specified depends on the suffixes to the data type (see suffixes without-corr, without-man, mandatory-man and mandatory-corr in table 3). product-version may be enclosed within single quotes (possibly with a preceding C). The specification of the version may begin with the letter V.
structured-name	A…Z 0…9 $, #, @ hyphen	Alphanumeric string which may comprise a number of substrings separated by a hyphen. First character: A...Z or $, #, @
text	freely selectable	For the input format, see the relevant operand descriptions.
time	0…9 structure identifier: colon	Time-of-day entry: Input format: { hh:mm:ss | hh:mm | hh } hh: hours mm: minutes ss: seconds } Leading zeros may be omitted
vsn	a) A…Z 0…9	a) Input format: pvsid.sequence-no max. 6 characters pvsid: 2-4 characters; PUB must not be entered sequence-no: 1-3 characters
	b) A…Z 0…9 $, #, @	b) Max. 6 characters; PUB may be prefixed, but must not be followed by $, #, @.

Table 2: Data types (part 5 of 6)

Data type	**Character set**	**Special rules**
x-string	Hexadecimal: 00…FF	Must be enclosed in single quotes; must be prefixed by the letter X. There may be an odd number of characters.
x-text	Hexadecimal: 00…FF	Must not be enclosed in single quotes; the letter X must not be prefixed. There may be an odd number of characters.

Table 2: Data types (part 6 of 6)

Suffixes for data types

Suffix	Meaning
x..y *unit*	With data type “integer”: interval specification x minimum value permitted for “integer”. x is an (optionally signed) integer. y maximum value permitted for “integer”. y is an (optionally signed) integer. *unit* with “integer” only: additional units. The following units may be specified: *days* *byte* *hours* *2Kbyte* *minutes* *4Kbyte* *seconds* *Mbyte* *milliseconds*
x..y *special*	With the other data types: length specification For data types catid, date, device, product-version, time and vsn the length specification is not displayed. x minimum length for the operand value; x is an integer. y maximum length for the operand value; y is an integer. x=y the length of the operand value must be precisely x. *special* Specification of a suffix for describing a special data type that is checked by the implementation. “special” can be preceded by other suffixes. The following specifications are used: *arithm-expr* arithmetic expression (SDF-P) *bool-expr* logical expression (SDF-P) *string-expr* string expression (SDF-P) *expr* freely selectable expression (SDF-P) *cond-expr* conditional expression (JV) *symbol* CSECT or entry name (BLS)
with	Extends the specification options for a data type.
-compl	When specifying the data type “date”, SDF expands two-digit year specifications in the form yy-mm-dd to: 20yy-mm-dd if yy < 60 19yy-mm-dd if yy ≥ 60
-low	Uppercase and lowercase letters are differentiated.
-path-compl	For specifications for the data type “filename”, SDF adds the catalog and/or user ID if these have not been specified.

Table 3: Data type suffixes (part 1 of 7)

Suffix	**Meaning**	
with (contd.)		
-under	Permits underscores (_) for the data type “name”.	
-wild(n)	Parts of names may be replaced by the following wildcards. n denotes the maximum input length when using wildcards. Due to the introduction of the data types posix-filename and posix-pathname, SDF now accepts wildcards from the UNIX world (referred to below as POSIX wildcards) in addition to the usual BS2000 wildcards. However, as not all commands support POSIX wildcards, their use for data types other than posix-filename and posix-pathname can lead to semantic errors. Only POSIX wildcards or only BS2000 wildcards should be used within a search pattern. Only POSIX wildcards are allowed for the data types posix-filename and posix-pathname. If a pattern can be matched more than once in a string, the first match is used.	
	BS2000 wildcards	Meaning
	*	Replaces an arbitrary (even empty) character string. If the string concerned starts with *, then the * must be entered twice in succession if it is followed by other characters and if the character string entered does not contain at least one other wildcard.
	Terminating period	Partially-qualified entry of a name. Corresponds implicitly to the string “./*”, i.e. at least one other character follows the period.
	/	Replaces any single character.
	<s_x:s_y>	Replaces a string that meets the following conditions: – It is at least as long as the shortest string (s_x or s_y) – It is not longer than the longest string (s_x or s_y) – It lies between s_x and s_y in the alphabetic collating sequence; numbers are sorted after letters (A...Z, 0...9) – s_x can also be an empty string (which is in the first position in the alphabetic collating sequence) – s_y can also be an empty string, which in this position stands for the string with the highest possible code (contains only the characters X’FF’)
	<s_1,…>	Replaces all strings that match any of the character combinations specified by s. s may also be an empty string. Any such string may also be a range specification “s_x:s_y” (see above).

Table 3: Data type suffixes (part 2 of 7)

Suffix	**Meaning**	
with-wild(n) (contd.)	-s	Replaces all strings that do not match the specified string s. The minus sign may only appear at the beginning of string s. Within the data types filename or partial-filename the negated string -s can be used exactly once, i.e. -s can replace one of the three name components: cat, user or file.
	Wildcards are not permitted in generation and version specifications for file names. Only system administration may use wildcards in user IDs. Wildcards cannot be used to replace the delimiters in name components cat (colon) and user ($ and period).	
	POSIX wildcards	Meaning
	*	Replaces any single string (including an empty string). An * appearing at the first position must be duplicated if it is followed by other characters and if the entered string does not include at least one further wildcard.
	?	Replaces any single character; not permitted as the first character outside single quotes.
	[c_x-c_y]	Replaces any single character from the range defined by c_x and c_y, including the limits of the range. c_x and c_y must be normal characters.
	[s]	Replaces exactly one character from string s. The expressions [c_x-c_y] and [s] can be combined into [s_1c_x-c_ys_2].
	[!c_x-c_y]	Replaces exactly one character not in the range defined by c_x and c_y, including the limits of the range. c_x and c_y must be normal characters. The expressions [!c_x-c_y] and [!s] can be combined into [!s_1c_x-c_ys_2].
	[!s]	Replaces exactly one character not contained in string s. The expressions [!s] and [!c_x-c_y] can be combined into [!s_1c_x-c_ys_2].

Table 3: Data type suffixes (part 3 of 7)

Suffix	**Meaning**
with (contd.)	
wild-constr(n)	Specification of a constructor (string) that defines how new names are to be constructed from a previously specified selector (i.e. a selection string with wildcards). See also with-wild. n denotes the maximum input length when using wildcards. The constructor may consist of constant strings and patterns. A pattern (character) is replaced by the string that was selected by the corresponding pattern in the selector. The following wildcards may be used in constructors:

Wildcard	Meaning
*	Corresponds to the string selected by the wildcard * in the selector.
Termina-ting period	Corresponds to the partially-qualified specification of a name in the selector; corresponds to the string selected by the terminating period in the selector.
/ or ?	Corresponds to the character selected by the / or ? wildcard in the selector.
<n>	Corresponds to the string selected by the n-th wildcard in the selector, where n is an integer.

Allocation of wildcards to corresponding wildcards in the selector:
All wildcards in the selector are numbered from left to right in ascending order (global index).
Identical wildcards in the selector are additionally numbered from left to right in ascending order (wildcard-specific index).
Wildcards can be specified in the constructor by one of two mutually exclusive methods:

1. Wildcards can be specified via the global index: <n>
2. The same wildcard may be specified as in the selector; substitution occurs on the basis of the wildcard-specific index. For example: the second “/” corresponds to the string selected by the second “/” in the selector

Table 3: Data type suffixes (part 4 of 7)

Suffix	**Meaning**
with-wild-constr(n) (contd.)	The following rules must be observed when specifying a constructor: – The constructor can only contain wildcards of the selector. – If the string selected by the wildcard <...> or [...] is to be used in the constructor, the index notation must be selected. – The index notation must be selected if the string identified by a wildcard in the selector is to be used more than once in the constructor. For example: if the selector “A/” is specified, the constructor “A<n><n>” must be specified instead of “A//”. – The wildcard * can also be an empty string. Note that if multiple asterisks appear in sequence (even with further wildcards), only the last asterisk can be a non-empty string, e.g. for “****” or “*//*”. – Valid names must be produced by the constructor. This must be taken into account when specifying both the constructor and the selector. – Depending on the constructor, identical names may be constructed from different names selected by the selector. For example: “A/*” selects the names “A1” and “A2”; the constructor “B*” generates the same new name “B” in both cases. To prevent this from occurring, all wildcards of the selector should be used at least once in the constructor. – If the constructor ends with a period, the selector must also end with a period. The string selected by the period at the end of the selector cannot be specified by the global index in the constructor specification.

Table 3: Data type suffixes (part 5 of 7)

Suffix	Meaning
with-wild-constr(n) (contd.)	Examples:

Selector	Selection	Constructor	New name
A//*	AB1 AB2 A.B.C	D<3><2>	D1 D2 D.CB
C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<3>.XY<2>	G.A.D.XYA G.A.D.XYB G.B.F.XYA G.B.F.XYB
C.<A:C>/<D,F>	C.AAD C.ABD C.BAF C.BBF	G.<1>.<2>.XY<2>	G.A.A.XYA G.A.B.XYB G.B.A.XYA G.B.B.XYB
A//B	ACDB ACEB AC.B A.CB	G/XY/	GCXYD GCXYE GCXY. [1] G.XYC

[1] The period at the end of the name may violate naming conventions (e.g. for fully-qualified file names).

Suffix	Meaning
without	Restricts the specification options for a data type.
-cat	Specification of a catalog ID is not permitted.
-corr	Input format: [[C]'][V][m]m.na[''] Specifications for the data type product-version must not include the correction status.
-gen	Specification of a file generation or file generation group is not permitted.
-man	Input format: [[C]'][V][m]m.n[''] Specifications for the data type product-version must not include either release or correction status.
-odd	The data type x-text permits only an even number of characters.
-sep	With the data type “text”, specification of the following separators is not permitted: ; = ( ) < > ␣ (i.e. semicolon, equals sign, left and right parentheses, greater than, less than, and blank).
-temp-file	Specification of a temporary file is not permitted (see #file or @file under filename).

Table 3: Data type suffixes (part 6 of 7)

Suffix	**Meaning**
without (contd.)	
-user	Specification of a user ID is not permitted.
-vers	Specification of the version (see “file(no)”) is not permitted for tape files.
-wild	The file types posix-filename and posix-pathname must not contain a pattern (character).
mandatory	Certain specifications are necessary for a data type.
-corr	Input format: [[C]’][V][m]m.naso[’] Specifications for the data type product-version must include the correction status and therefore also the release status.
-man	Input format: [[C]’][V][m]m.na[so][’] Specifications for the data type product-version must include the release status. Specification of the correction status is optional if this is not prohibited by the use of the suffix without-corr.
-quotes	Specifications for the data types posix-filename and posix-pathname must be enclosed in single quotes.

Table 3: Data type suffixes (part 7 of 7)

2 Syntax files and their processing with SDF-A

Syntax files are supplied with the basic configuration of BS2000 and with the various software products. These contain:

- information on the syntax of all and of the program statements
- information as to how these commands are implemented in BS2000, e.g.
  - the names of the system entries via which command execution is effected
  - the specifications for passing parameters to the executing system modules
- general and command-specific specifications for guided dialogs
- explanatory texts for the commands or statements and their operands (“help texts”)
- privileges required to access domains, commands, statements, operands and operand values

The syntax files are merged, as required, to form group or system syntax files and are assigned accordingly in the system (see the “SDF Management” manual [2]). When SDF processes a command that has been entered, it retrieves the information needed to do so from the activated syntax files (see section “File hierarchy” on page 33).

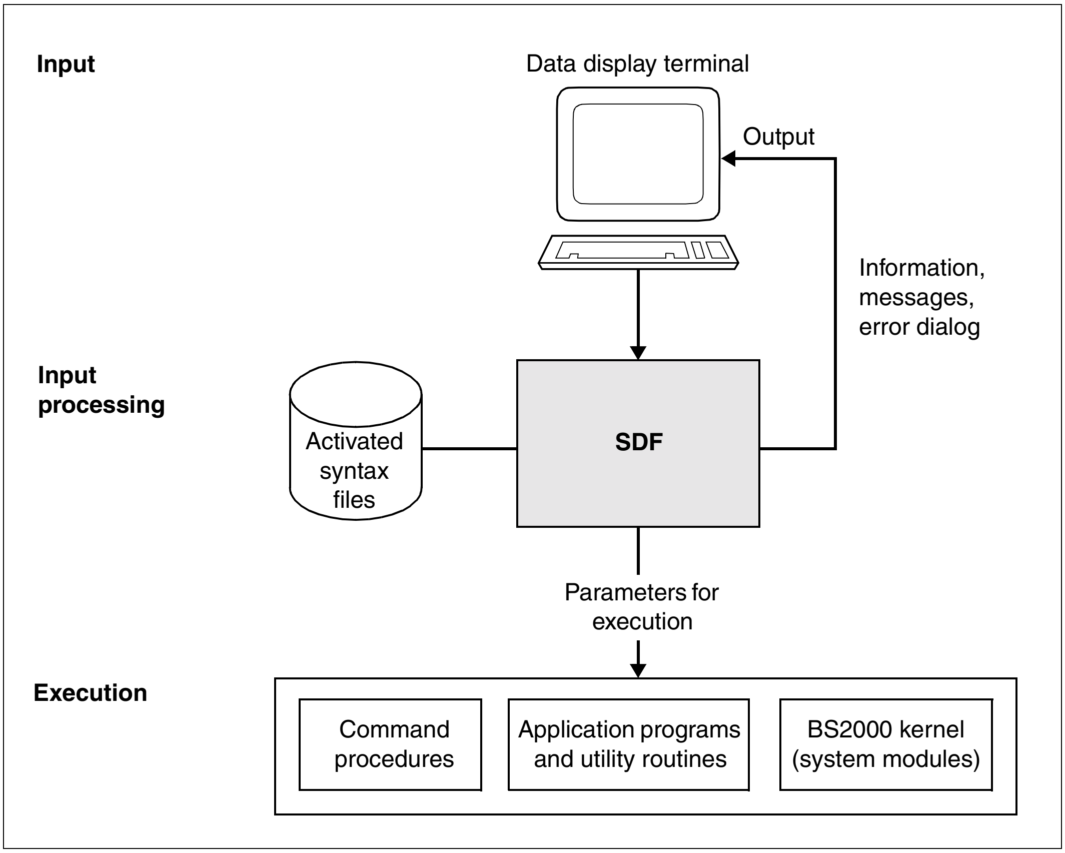


Figure 1: Principle of operation of SDF

2.1 File types

There are three types of syntax file:

– system syntax files

– group syntax files

– user syntax files

In addition to the system syntax files, one group and one or more user syntax files may be activated for a task. In this context, there is a fixed file hierarchy that determines how SDF is to work with more than one syntax file (see section “File hierarchy” on page 33ff).

SDF-A supports syntax files with and without a PAM key.

2.1.1 System syntax files

Syntax files of the type SYSTEM are supplied with the basic configuration of BS2000 and with each software product. These contain the definitions of commands and programs which are to be available throughout the system. They are supplied by Fujitsu Siemens Computers and can be modified with SDF-A. As of SDF V3.0 multiple system syntax files can be active simultaneously. The basic system syntax file must be one of those active, and may be accompanied by a number of subsystem syntax files.

Basic system syntax file

You require at least the basic system syntax file in order to work with SDF. It contains the definitions of the domains, the standard statements and the SDF-specific commands (from the SDF domain) as well as the system-wide global information settings. The syntax file SYSSDF.SDF.045 is activated for this purpose by default. Systems support can make the modifications (e.g. restrict functions) with the SDF-A (or SDF-U) utility routine. Modifications apply to all users of the system.

When SDF starts, the name of the basic system syntax file is read from the SDF parameter file and the file is automatically activated.

During a session, system administration can replace the activated basic system syntax file with another by means of the MODIFY-SDF-PARAMETERS command. A change to the basic system syntax file during the session immediately affects all existing and future tasks.

Subsystem syntax files

There can be also several subsystem syntax files activated in addition to the basic syntax file. A subsystem syntax file contains the definitions of commands and programs that belong to a subsystem managed by DSSM or even to any installation unit (including BS2CP). Subsystem syntax files can be modified using SDF-A. Any such modifications, e.g. limitations on functional scope, apply for all users of the system.

The activation of a subsystem syntax file during the session takes immediate effect for all existing and future tasks.

There are two ways to activate subsystem syntax files:

- The name of the subsystem syntax file was defined in the subsystem declaration. In this case the subsystem syntax file is automatically activated when the subsystem is loaded and deactivated when the subsystem is unloaded.
- The name of the subsystem syntax file is entered in the SDF parameter file. In this case the subsystem syntax file is automatically activated at system initialization. If the pubset on which this file is stored is not available when the system is initialized, then it can only be activated after importing the pubset.
  It can be modified (deactivated or exchanged) via the SDF parameter file only. In this case, the subsystem syntax file is available independently of the availability of the corresponding subsystem.

If a command or statement definition is defined in more than one active subsystem syntax files, then the following cases arise:

- Only one version of the subsystem can be active at a time. In this case the command or statement definition is used that is found in the syntax file of the version of the subsystem last activated.
- Several versions of the subsystem can be active at a time (coexistence). The following cases arise:
  - The command or statement definition from the syntax file of the corresponding subsystem version is used for a task that is assigned to a specific subsystem version.
    The syntax analysis of a START-<utility> command is always performed before the program is loaded, i.e. before connecting the task to a subsystem version of the program called. This is why the syntax file of the first version of the subsystem activated is evaluated.
  - The command or statement definition contained in the first version of the subsystem activated is used for tasks that are not assigned to any subsystem version.

2.1.2 Group syntax file

Syntax files of type GROUP may also be supplied with software products. These contain the definitions of commands and programs which are to be available only to certain user IDs. They are supplied by Fujitsu Siemens Computers and can be modified with SDF-A. System administration can also create group syntax files with SDF-A. With SDF-I [2] they are merged into the appropriate group syntax file. The modifications, such as functional restrictions, apply to all user IDs to which the group syntax file is assigned.

A group syntax file may contain extensions, restrictions and other modifications with respect to the system syntax files. By assigning a group syntax file, system administration can match the available functional scope precisely to the needs of certain users.

If, for example, system administration has restricted the functional scope supplied by Fujitsu Siemens Computers in the basic system syntax file, a group syntax file can be used to cancel these restrictions for certain user IDs. Conversely, a group syntax file can be used to prevent certain user IDs from using functions available on a global basis.

A group syntax file may be activated, but need not be. System administration assigns it to a profile ID via the file name. A profile ID may be assigned to one or more user IDs. User IDs with the same profile ID all use the same group syntax file (see section “Naming conventions” on page 36).

With the MODIFY-SDF-PARAMETERS command, system administration can

– change an existing assignment,

– cancel an existing assignment, or

– assign a group syntax file to a profile ID for the first time.

The SCOPE operand defines this assignment as permanent, permanent as of the next session, or valid for the current session only. A specification made with MODIFY-SDF-PARAMETERS applies only to tasks generated *after* this specification. The period of validity for this specification can be defined more precisely with the SCOPE operand. Existing tasks remain unaffected. When a group syntax file is assigned to a profile ID, it is automatically activated following processing of the SET-LOGON-PARAMETERS command. It remains activated until the end of the task.

Only one group syntax file can ever be activated per task. At startup, SDF generates a class 4 list containing the names of the group syntax files and the related profile IDs as laid down in the SDF parameter file.

If the group syntax file to be activated is not available when the LOGON command is processed, e.g. because it is being processed itself, a message is output and the task is created without the group syntax file.

A warning is output if the SDF parameter file does not exist or is invalid. If the TSOS group syntax file to be activated is invalid, SDF issues a console prompt requesting a correct TSOS group syntax file. Thereafter, this group syntax file is the one activated at the next LOGON under TSOS.

The basic system syntax file containing the command and program definitions for the user IDs TSOS, SYSPRIV and SYSAUDIT is the only one required as of BS2000/OSD-BC V1.0. Since the TSOS, SECURITY-ADMINISTRATION, SAT-FILE-MANAGEMENT and SAT-FILE-EVALUATION privileges are assigned to these definitions, this special functionality is available only to those users who possess the corresponding privileges.

Group syntax files are supplied in conjunction with certain products or created by the users themselves with the aid of SDF-A.

2.1.3 User syntax files

A user syntax file may contain extensions, restrictions and other modifications with respect to the basic system syntax file or the subsystem syntax files and, where applicable, with respect to the group syntax file. Possible extensions and other functional modifications are limited to statements to user programs and to commands implemented via procedures. Functional restrictions for BS2000 commands (implemented via system modules), specified in the system syntax files or the group syntax file cannot be canceled in a user syntax file.

A user syntax file may be activated, but need not be. As of SDF V4.1 several user names can be active simultaneously for each task. If a command or statement is defined in several simultaneously activated user syntax files, the definition from the last activated user syntax file is used.

If a user syntax file has the name $<userid>.SDF.USER.SYNTAX and a task under the user ID <userid> is started, this user syntax file is automatically activated following processing of the SET-LOGON-PARAMETERS command. If the user syntax file to be activated is not available at this point, e.g. because it is being processed itself, a message is output and the task is created without the user syntax file.

During the task, the user can activate the user syntax file(s) by means of the MODIFY-SDF-OPTIONS command/statement. MODIFY-SDF-OPTIONS can also be used to deactivate the user syntax file.

User syntax files must be generated by the user with the aid of SDF-A.

2.2 Concept of privileges

As of BS2000 V10.0, the privileges formerly granted to the user ID TSOS can be distributed to other user IDs if the software product SECOS [10] is loaded in the system. The privileges that can be assigned to the user IDs are known as global privileges and authorize the user to perform certain tasks such as global user administration (USER-ADMINISTRATION) or security administration (SECURITY-ADMINISTRATION). If a user ID has a privilege of this nature, the user can perform privileged operations, using certain privileged commands for the purpose

2.2.1 Privileges in syntax files

Until now, privileged objects (domains, programs, commands, statements, operands and operand values) were defined in group syntax files assigned to the user IDs with the appropriate privileges. This mechanism ensured that no other user ID had access to privileged objects.

As of Version 2.0 of SDF-A, it is possible to define the individual privileges (e.g. TSOS, USER-ADMINISTRATION or SECURITY-ADMINISTRATION) that a user ID requires in order to work with specific objects (domains, programs, commands, statements, operands and operand values). Consequently, all objects can be defined in a single syntax file to which all users have access. A user request cannot access an object unless it possesses at least one of the privileges allocated to the object. A user request can have more than one privilege.

Privileges are keywords. Possible privileges are discussed in the “SECOS” User Guide [10]; note that privileges also depend on the software installed (for example the HSMS-ADMINISTRATION and VM2000-ADMINISTRATION privileges).

Privileges are allocated to an object by SDF-A by means of a suitable ADD statement and modified by means of a suitable MODIFY statement.

Only those objects for which the user has the requisite privileges and can therefore access are displayed in a guided dialog.

The specification and evaluation of privileges for objects in the syntax file are supported as follows:
- by SDF-A as of Version 2.0
- by SDF as of Version 2.0
- in BS2000 as of BS2000/OSD-BC V1.0 in conjunction with SRPM (see “SECOS” User Guide [10]).

2.2.2 Assigning privileges; notes and conventions

- If a privileges list is modified, those privileges that remain unchanged must be repeated in the MODIFY statement.
- If a new privilege is assigned to a structure, only the dependent structures defined with PRIVILEGE=*SAME receive the new privilege. This means that after every modify operation, dependent structures having privileges of their own must be altered separately (with EDIT and MODIFY-...).
- The same test is run in all MODIFY and ADD statements, in order to prevent a dependent structure having more privileges than the higher-order structures.
- If a privilege is canceled in a structure, it is also canceled in those dependent structures that have privileges of their own. This prevents inconsistencies in the syntax tree.
  The privilege in question is canceled in those dependent structures defined with PRIVILEGE=*SAME.
  The last privilege in a dependent structure cannot be canceled. An attempt to cancel the last privilege is rejected with an error message. If this happens, use the REMOVE statement to explicitly remove the structure.
- If an operand has a default value, the default value must have the same privileges as the operand.
- If an operand does not have a default value, it must have the same privileges as its higher-order structure (command or value). For each of the operand's privileges, there must be at least a value having the same privilege. These two rules ensure that, irrespective of its privileges, each user request can assign a value to the operand.

2.3 File hierarchy

The existence of a basic system syntax file is a prerequisite for the use of SDF. In addition, multiple subsystem syntax files and, for each task, one group syntax file and one or more user syntax files can also be active. If a command or a statement is defined in more than one active system syntax file, the definition from the system syntax file last activated is used.

Certain rules govern how SDF selects a syntax file in order to fetch the information it requires in instances when syntax files of more than one type are simultaneously active for a user ID. The rules produce a hierarchy of syntax files that is applicable for a task under a certain user ID.
The following constellations are possible:

- only the system syntax files are active
- the system syntax files and a group syntax file are active
- the system syntax files and the user syntax files are active
- the system syntax files, a group syntax file and the user syntax files are active
- only a group syntax file is active
- a group syntax file and the user syntax files are active

Note that for all these constellations, as of BS2000/OSD-BC V1.0, the set of possible commands and statements also depends on the privileges assigned to the objects in question.

Only the system syntax files are active

In the simplest case, only the system syntax files are activated. Which commands and statements may be entered then depends solely upon the contents of the system syntax files.

System syntax files and group syntax file are active

If a group syntax file has been activated in addition to the system syntax files for the user ID, the definitions contained in the group syntax file (including global information) take precedence over those in the system syntax files. The disabling of a command or a statement in the group syntax file also counts as a definition. Only when a command or a statement is defined exclusively in the system syntax files does SDF take the information from there. Which commands and statements may be entered depends on the contents of both types of file.

System syntax files and user syntax files are active

If user syntax files have been activated in addition to the system syntax files, the definitions of BS2000 commands (implemented via system modules) contained in the former must be fully contained in the system syntax files. Otherwise, the definitions in the user syntax files (including global information and disabled commands) take precedence over those in the system syntax files.

System, group and user syntax files are active

If both a group syntax file and user syntax files have been activated in addition to the system syntax files, the following applies:

The command set permitted in a user syntax file is determined by the definitions contained in the group syntax file and the system syntax files. This means that a user syntax file must not contain system commands which

- are disabled in the group syntax file or the system syntax files,
- or that do not exist in these files,
- or for which the user does not have at least one of the requisite privileges.

However, disabled commands, modifications and global information in the user syntax files take precedence over the definitions in the group and system syntax files.

If a system command (implemented via system modules) is defined in all three types of syntax file, the definition contained in the user syntax file must be fully contained in the group syntax file.

The following possible cases exist.

- A user syntax file contains user-defined commands (implemented in the form of procedures):
  all system commands (implemented via system modules) contained in the procedure must be covered by definitions in the system syntax files or the group syntax file.
- A command is disabled in a user syntax file:
  the command cannot be executed, even if its definition is covered by the group syntax file or the system syntax files.
- A user syntax file contains a system command for which an operand value has been changed:
  the command is executed as defined in the user syntax file.
- The group syntax file or the system syntax files contain a command which is not defined in any of the user syntax files:
  the user may use the command as long as it is not explicitly disabled in the user syntax file.

- A command is disabled in the group syntax file.
  The command cannot be executed, even if it is defined in the user syntax file and covered by a definition in the system syntax files.

Only a group syntax file is active

The MODIFY-SDF-PARAMETERS command (see “SDF Management” [2]) can be used to assign a group syntax file to a profile ID. If the HIERARCHY operand is set to *NO, *only* the specified group syntax file (no system syntax files) is activated (at LOGON) for all tasks of user IDs with this profile ID. In this case, the possible command and statement inputs are determined solely by the contents of this group syntax file.

One group syntax file and user syntax files are active

If no system syntax files are active for a task (see above) and one or more user syntax files are active in addition to the group syntax file, the definitions of the BS2000 commands (implemented via system modules) contained in the user syntax file(s) must be fully covered by the group syntax file. Apart from this, the definitions in the user syntax file(s) (including global information and disabled commands) take precedence over those in the group syntax file.

2.4 Naming conventions

In addition to the system syntax files, there may be one group syntax file and one or more user syntax files active for each user task. In this case, a file hierarchy determines how SDF uses the various syntax files. System syntax files and group syntax files can be activated by means of the MODIFY-SDF-PARAMETERS command, user syntax files with the MODIFY-SDF-OPTIONS command. The names of the system syntax files and the group syntax files to be activated are entered in an SDF parameter file.

The basic system syntax file is automatically activated when SDF is loaded. The same applies to the subsystem syntax files if they are named in the SDF parameter file, i.e. when their names are entered in the SDF parameter file.
If no parameter file is specified in the DSSM declaration for SDF, then $TSOS.SYSPAR.SDF is used as the parameter file. If the parameter file does not have any valid contents, then a new name is requested via console message CMD0691. When "*STD“ is the reply, $TSOS.SYSSDF.SDF.045 is activated as the basic system syntax file (and $TSOS.SYSSDF.BS2CP.<bs2vers> is activated as the subsystem syntax file).
The activated basic system syntax file can be switched during the session by system administration (with the MODIFY-SDF-PARAMETERS command). The basic system syntax file activated during a session with MODIFY-SDF-PARAMETERS may have any name.

System administration assigns a group syntax file to a profile ID and can change this assignment during the session with the aid of the MODIFY-SDF-PARAMETERS command. The SCOPE operand defines this assignment as permanent, permanent as of the next session, or valid for the current session only.
The group syntax file assigned in this way is activated automatically after LOGON processing for each task under a user ID with the corresponding profile ID.

A warning is output if the SDF parameter file does not exist or is invalid.

User syntax files are linked to a user ID. If a user syntax file is to be activated automatically after LOGON processing, it must be cataloged under the name $<userid>.SDF.USER.SYNTAX. The user may activate or deactivate user syntax files during execution of a user task (with MODIFY-SDF-OPTIONS). If a user syntax file is activated with MODIFY-SDF-OPTIONS, it may have any desired name.

2.5 Processing syntax files

SDF-A can be used to create new syntax files or modify existing files. SDF-A works with the syntax files exclusively on the logical level, i.e. the physical structure of the syntax files remains hidden.

As of SDF-A V3.0, every user has access to the full range of SDF-A functions. While enabling the user to process group and system syntax files (taking due account of file attributes and privileges), this does not allow him/her to activate group and system syntax files (see page 36). Only system administration can control which system syntax files are to be activated for the system and which group syntax file is to be activated for a user ID.

2.5.1 Functional scope of SDF-A

SDF-A offers the following facilities:

– definition of user commands, implemented via command procedures, and modification of such definitions

– definition of user programs and the statements to them, and modification of such definitions

– disabling of commands, programs, statements, operands and operand values

– modification of default values

– renaming of domains, of command, statement and operand names, and of operand values that are keywords

– modification or replacement of help texts for guided dialog, e.g. replacement of German help texts by those in the appropriate local language

– modification of the assignment of commands to domains, and definition of new domains

– modification of the default values of SDF options, e.g. for guided dialog

– deletion and reconstruction of command and statement definitions

2.5.2 Opening a syntax file

The SDF-A statement OPEN-SYNTAX-FILE is used to open a syntax file for subsequent processing.

System syntax file

When opening a system syntax file for processing, it is possible to assign another system syntax file (with any name) as a reference file by means of the SYSTEM-CONTROL operand. This reference file should contain the definitions of all commands available throughout the system, in the version supplied by the implementation. In particular, this file should contain the original version of the definitions furnished by Siemens for the user commands. In the simplest case, this file is merely a copy of the basic system syntax file received from Fujitsu Siemens Computers.

With the aid of the SYSTEM-CONTROL file, SDF-A checks whether the changes to be made are compatible with the implementation. If no SYSTEM-CONTROL file is assigned, it is possible for SDF-A to modify the syntax file in a way that is not actually permitted. This will cause difficulties later on, when the modified system syntax file is activated.

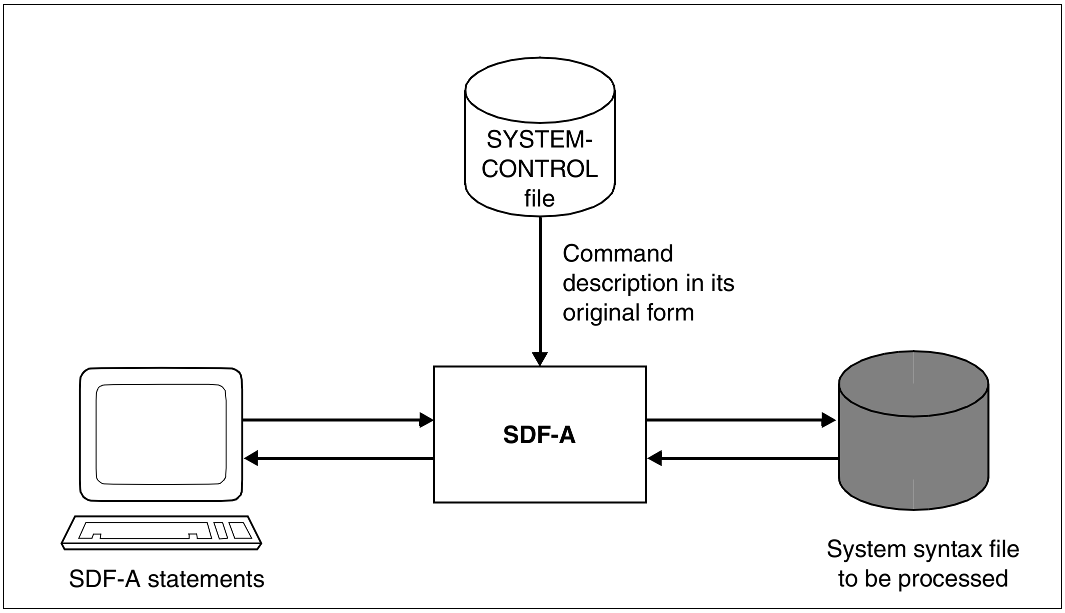

SYSTEM-CONTROL file
Command description in its original form
SDF-A
SDF-A statements
System syntax file to be processed

Figure 2: Processing a system syntax file with SDF-A

Group syntax file

When opening a group syntax file for processing, it is possible to assign a system syntax file (with any name) as a reference file by means of the SYSTEM-CONTROL operand. This file is subject to the same conditions as a SYSTEM-CONTROL file assigned when opening a system syntax file.

A special SYSTEM-CONTROL file is needed when BS2000 V10 is used and the group syntax file is to be prepared for system administration. Such a file is obtained by copying the definitions of the commands and programs that are partly or entirely reserved for system administration into a copy of the normal SYSTEM-CONTROL file, using COPY...,OVERWRITE-POSSIBLE=*YES (see the description of the COPY statement).

In addition, another system syntax file (with any name) may be assigned by means of the SYSTEM-DESCRIPTIONS operand. This should be the file that will later be activated together with the processed group syntax file.

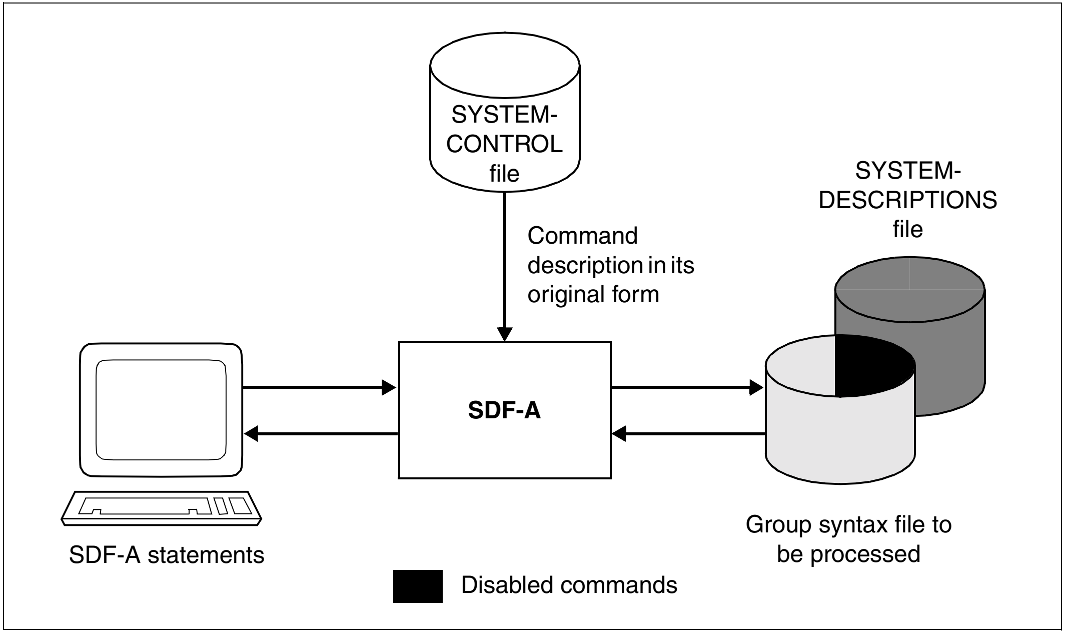


Figure 3: Processing a group syntax file with SDF-A

If the group syntax file is opened with specification of a SYSTEM-DESCRIPTIONS file, the commands and statements that are defined exclusively within the SYSTEM-DESCRIPTIONS file can also be processed. If a definition is modified, SDF-A writes the modified version into the group syntax file. The definition in the SYSTEM-DESCRIPTIONS file remains unchanged.

If a command that is defined in the SYSTEM-DESCRIPTIONS file is to be disabled, it is absolutely essential to specify this file when the group syntax file is opened. Otherwise, SDF-A merely deletes the definition in the group syntax file (if there is one), without placing a corresponding lock entry there.

If a command modification made in the group syntax file is to be retracted, you must not specify the SYSTEM-DESCRIPTIONS file. The modified definition in the group syntax file can then be deleted. SDF will then use the (unmodified) definition from the system syntax file.

User syntax file

When opening a user syntax file for processing, a user syntax file (with any name) may be assigned as a reference file by means of the USER-CONTROL operand. This file should contain the original version of definitions of user commands. Otherwise, the same conditions apply to this file as apply to a SYSTEM-CONTROL file assigned when a group or system syntax file is opened.

In addition, a group or system syntax file may be assigned with the aid of the operands GROUP-DESCRIPTIONS and SYSTEM-DESCRIPTIONS, respectively. These files should be the files that will later be activated together with the processed user syntax file.

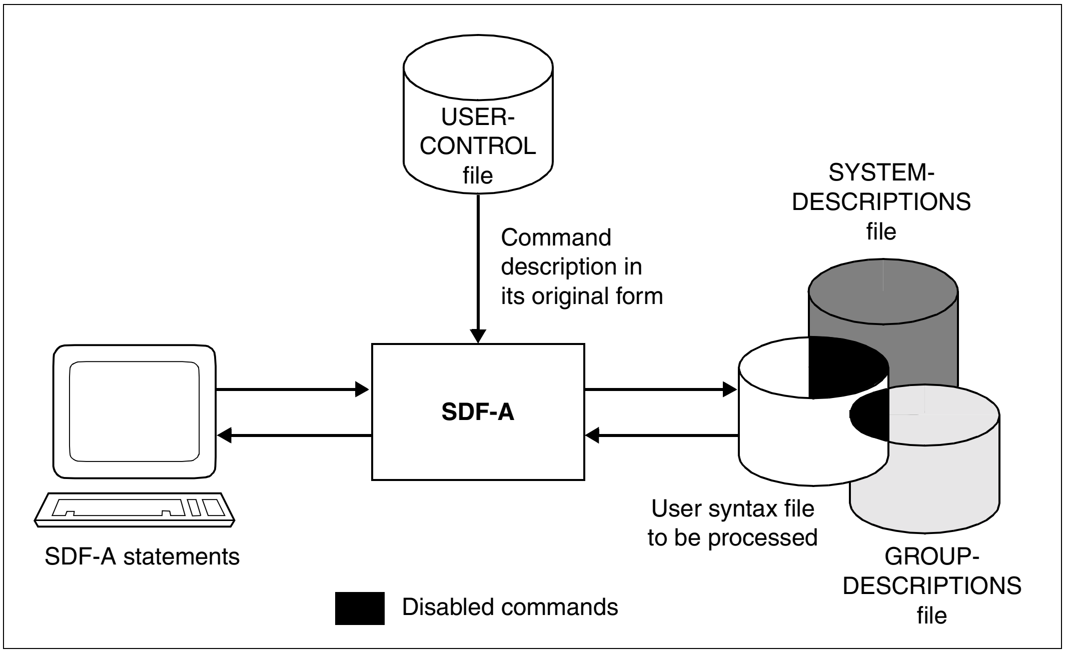


Figure 4: Processing a user syntax file with SDF-A

In principle, the same conditions apply to the GROUP-DESCRIPTIONS and SYSTEM-DESCRIPTIONS files specified when opening a user syntax file as apply to the SYSTEM-DESCRIPTIONS file specified when opening a group syntax file. However, here the functional scope of the BS2000 commands (implemented via system modules) defined in the user syntax file must be completely contained in the GROUP-DESCRIPTIONS and SYSTEM-DESCRIPTIONS files. Here, these files also have the function of the SYSTEM-CONTROL file. In the user syntax file, definitions of commands that are implemented via system modules may be modified but not redefined.

2.5.3 Processing command and statement definitions

Command and statement definitions consist of

- a global definition of the command or statement (header),
- a definition of each operand, and
- a definition of each operand value (input alternatives)

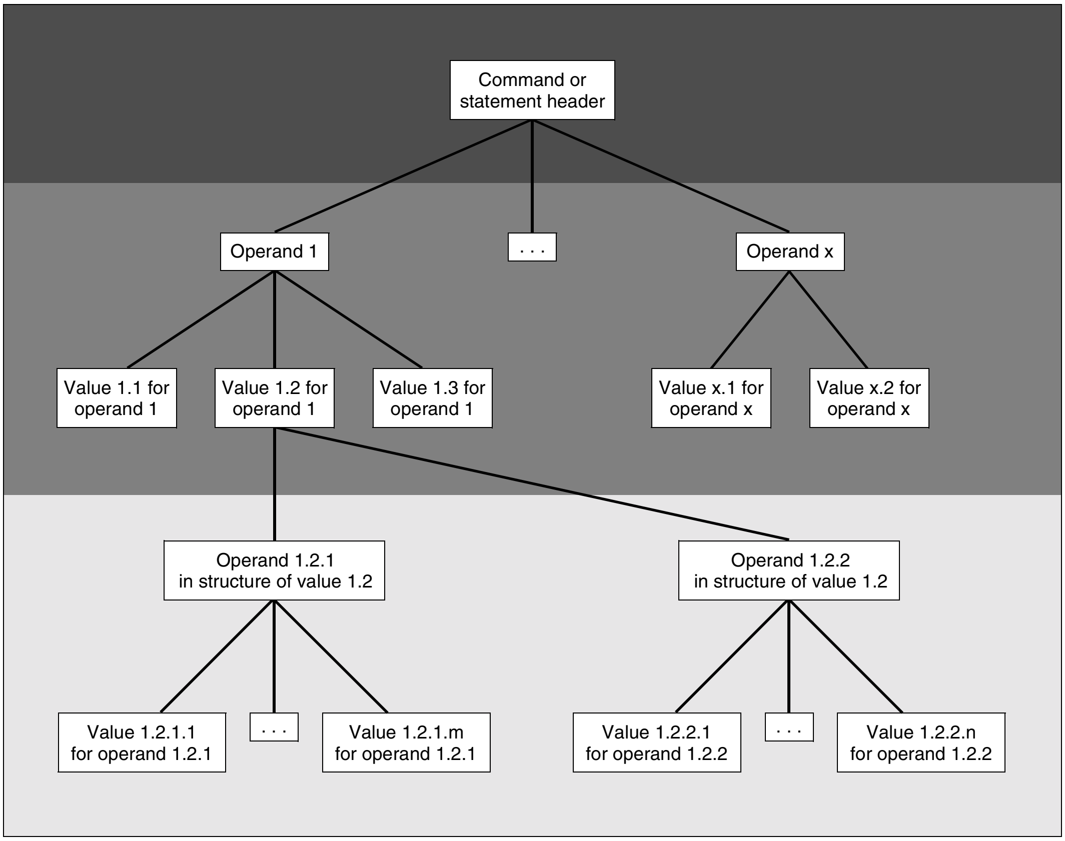

Command or statement header
Operand 1
. . .
Operand x
Value 1.1 for operand 1
Value 1.2 for operand 1
Value 1.3 for operand 1
Value x.1 for operand x
Value x.2 for operand x
Operand 1.2.1 in structure of value 1.2
Operand 1.2.2 in structure of value 1.2
Value 1.2.1.1 for operand 1.2.1
. . .
Value 1.2.1.m for operand 1.2.1
Value 1.2.2.1 for operand 1.2.2
. . .
Value 1.2.2.n for operand 1.2.2

Figure 5: Representation of a command or statement definition as a tree

At least one operand value must be defined for each operand. This also applies to the case where an operand with a default value is masked out at the user interface for the command or statement. If there is a structure (see the glossary at the back of the manual) linked to an operand value, the structure is defined globally in the definition of the operand value. In addition, there is

– a definition of each individual operand of the structure, and

– a definition of each value belonging to one of these operands.

The structure is ended by the CLOSE-STRUCTURE statement.

The command or statement definition is ended by the CLOSE-CMD-OR-STMT statement.

Statements to SDF-A can include comments, which must be enclosed in quotation marks (").

```
//ADD-CMD ... or //ADD-STMT
//ADD-OPERAND <operand 1>,...
//ADD-VALUE <value 1.1>,...
//ADD-VALUE <value 1.2>,...,STRUCTURE=*YES(...),..
//ADD-OPERAND <operand 1.2.1>,...
//ADD-VALUE <value 1.2.1.1>,...
 . . .
//ADD-VALUE <value 1.2.1.m>,...
//ADD-OPERAND <operand 1.2.2>,...
//ADD-VALUE <value 1.2.2.1>,...
 . . .
//ADD-VALUE <value 1.2.2.n>,...
//CLOSE-STRUCTURE
//ADD-VALUE <value 1.3>,...
 . . .
//ADD-OPERAND <operand x>,...
//ADD-VALUE <value x.1>,...
//ADD-VALUE <value x.2>,...
//CLOSE-CMD-OR-STMT
```

Figure 6: Sequence of SDF-A statements used to set up the definitions shown in Figure 5

A command or statement definition can be

- created
- modified
- deleted, or
- output on SYSOUT or SYSLST.

The same applies to the definition of an operand or an operand value.

When a command or statement definition is newly created, this does not necessarily mean that all of the associated operand and operand value definitions also have to be newly created. It is often possible to draw on existing operand or operand value definitions, which are then copied into the new command or statement definition (see COPY). The existing definition need not necessarily be exactly the definition required. If an existing definition deviates only in minor details from the one required, it is more practical to make a copy and to change the copy accordingly (see below).

A practical procedure for creating a command or statement definition would be as follows:

Step 1

Establish the framework of the command or statement (its name, operands, operand values, structures) using very simple ADD statements containing only a few specifications (default values).

Step 2

Display the provisional command or statement using the SHOW statement and check it.

Step 3

Complete the provisional command or statement definition using MODIFY statements in guided dialog (insertion of help texts and implementation specifications). Here, SDF-A proceeds through the command or statement tree.

A command or statement definition is modified either by changing the global definition (see MODIFY-CMD and MODIFY-STMT) or

- by adding operand or operand value definitions by creating them (see ADD-OPERAND and ADD-VALUE) or copying them (see COPY),
- by modifying operand or operand value definitions (see MODIFY-OPERAND and MODIFY-VALUE), or
- by deleting such definitions (see REMOVE).

Adding or modifying operand or operand value definitions requires prior positioning to the point to be edited in the command or statement definition, either explicitly by means of the EDIT statement or implicitly via the preceding processing operation.

A command or statement definition is deleted by means of the REMOVE statement.

A command, statement, program, operand or operand value may be disabled by means of the REMOVE statement. A system-global lock must be defined in the basic system syntax file. Commands implemented via system modules, as well as operands and operand values belonging to such commands, may also be disabled for a specific user ID in a group syntax file. When defining a lock in a group syntax file, the basic system syntax file must be assigned as a reference file. Technically, a lock can also be defined in a user syntax file, but since the user activates and deactivates a user syntax file, a lock of this nature applies only to the user.

The user can clear a lock on a command, program or statement by means of the RESTORE statement. To do so, their definitions must still exist in a command file which occupies a higher position in the file hierarchy than the syntax file being processed.

When displaying definitions (see SHOW), it is possible to specify

– how much detail should be shown, and

– whether the definitions should be shown in a form similar to a manual or, instead, as a series of SDF-A statements with which they could be newly defined.

2.6 Integrity of syntax files

If an interrupt is generated when a syntax file is being processed with SDF-A, typically as a result of pressing the K2 key or due to the HOLD-PROGRAM or EXECUTE-SYSTEM-CMD statement, it could potentially endanger the integrity of the file. In specially critical situations, the K2 key is ignored.

A syntax file that is only opened for reading (i.e. with OPEN-SYNTAX-FILE ....,MODE= *READ) is not endangered by such interrupts.

Any syntax file that needs to be created or modified must, however, be opened with MODE=*CREATE / *UPDATE. So long as an object is being processed, the occurrence of an interrupt could have an adverse effect on the integrity of the object or even the syntax file. SDF-A issues warnings if there is a risk of losing data when interrupts occur.

SDF-A initiates a write operation to disk only when an object is explicitly closed. This occurs in the case of the following statements:

– when CLOSE-CMD-OR-STMT closes the command or statement definition

– when OPEN opens a new syntax file, but first saves the syntax file that was open at the time (if present)

– when END saves the current syntax file and terminates SDF-A.

Caution!
If an interrupt occurs when the file is being saved, the syntax file may be corrupted or destroyed.

The creation, deletion or modification of a command usually involves multiple write operations, since the command or statement tables also need to be updated. Interrupts should be avoided especially in such cases, since they could otherwise produce inconsistencies between the command/statement tables and the actually present objects. If SDF detects such inconsistencies when activating a syntax file, the syntax file is rejected.
A syntax file that was rejected by SDF can be opened again with SDF-A (with MODE=*UPDATE(...). You will receive message SDA0446 as a warning. You can then check the syntax file, correct it if required, and save it again so that it is no longer rejected by SDF.

The following steps are recommended to prevent potential problems:

– As far as possible, you should execute a CLOSE-CMD-OR-STMT before viewing objects with SHOW. This will ensure that no objects are lost if the SHOW output is interrupted with and you do not return immediately to SDF-A with RESUME-PROGRAM.

– Do not open syntax files with MODE=*UPDATE(...) unless you really wish to make changes.

2.7 Syntax files with the old format

The objects in the syntax files are not structured the same way in all the SDF-A versions. It is therefore important to read the following notes in order to avoid compatibility problems:

- SDF V2.0 and higher and SDF-A V2.0 and higher can handle both the old and new formats for the syntax files. SDF-A converts the objects it accesses to the new format. SDF, in contrast, converts the objects only in virtual memory and makes no changes to the syntax files.
- Syntax files which are to run under an SDF version < 2.0 must not be created with SDF-A versions ≥ 2.0.
- The following function is provided for correcting syntax files with the old format (required under SDF V1.4 and earlier):
  If task switch 15 is set, SDF-A (V2.0 or higher) internally loads the program SDF-A-V1, which has the same functionality as SDF-A V1.0D. The syntax of the SDF-A statements then corresponds to that of SDF-A V1.0D, with three exceptions:
  - The default for the SYSTEM-DESCRIPTIONS and GROUP-DESCRIPTIONS operands in the OPEN-CMD-SET statement is *NO instead of *CURRENT, with the result that no reference syntax files are allocated by default. If the reference syntax files are specified explicitly they, too, must have the old format.
  - System syntax files cannot be created.
  - The SDF-A statement SHOW is not supported.
- If syntax files with the old format generated by SDF-A V1.0D are processed with SDF-A as of V2.0A, the processed objects and the global information are given the new format.
  These syntax files can then *no longer* be used under an SDF version < 2.0.
- As of SDF-A V4.0 you can select the syntax file format via the DEFINE-ENVIRONMENT statement. Syntax files created and processed with SDF-A V3.0 retain their original format if SYNTAX-FILE-FORMAT=*V3 is specified. Note that a syntax file can never be modified using an earlier SDF-A version than the one in which it was created and stored.

3 Modifying command and statement definitions

System administration may modify the command and statement definitions provided by Fujitsu Siemens Computers, either system-globally (system syntax files) or specifically for individual users (group or user syntax file). Any system-global modification that has been made can be retracted specifically for particular users (by means of entries in the group syntax files).

As of SDF-A V3.0, each user has access to the full range of SDF-A functions. While enabling the user to process group and system syntax files (taking due account of file attributes and privileges), this does not allow the user to activate group and system syntax files. System syntax files can be activated for systems and group syntax files for user IDs only at the discretion of system administration. For example, the user can generate a group syntax file with modified command definitions. System administration can then take this group syntax file generated by the user and assign it to a user ID such that it will be activated when the SET-LOGON-PARAMETERS command is processed.

The definitions provided by Fujitsu Siemens Computers for commands implemented via system modules may be modified only so long as the modification does not affect the functional scope of the command. For example, a user may change the default value of an operand but not the data type of a value defined for that operand.
A modification often involves a number of individual but coordinated steps. If a certain modification is used repeatedly, it is advisable to write a modification procedure containing all the SDF-A statements needed to modify a command or a statement. These procedures can then be used to update the group and user syntax files every time a new version of a product is supplied.

The bulk of this chapter consists of examples illustrating the mode of operation and facilities of SDF-A. Grouped together at the end of the chapter are some general remarks on defining and removing functional limitations.

If new command definitions are supplied by Fujitsu Siemens Computers as the result of a change of a product version, all group and user syntax files in which these command definitions are modified must be reconstructed.

3.1 Examples

The following examples illustrate some typical cases in which SDF-A is used.

These runtime examples were created with SDF-A V4.1E under BS2000/OSD-BC V5.0. Note that the chargeable product SDF-P [12] was also loaded in the test environment. Typical output messages are given here; the actual output messages will depend on the current system configuration.
User inputs in the runtime examples and outputs to be specially noted are printed in a `typewriter` font in **`bold`** and *`italics`*, respectively.

3.1.1 Example 1: Disabling commands

Users of user ID EXAMPLE are to be prevented from loading and starting programs. This is achieved by creating a group syntax file for the user ID EXAMPLE in which the LOAD-PROGRAM and START-PROGRAM commands, as well as the old LOAD and EXECUTE commands, are disabled. The special START commands for certain programs (e.g. START-SDF-A, START-EDT, START-LMS, ...) are then also disabled as these START commands require access to START-PROGRAM or LOAD-PROGRAM.

In BLSSERV V2.3 and higher the functionality of START-PROGRAM and LOAD-PROGRAM are also offered with improved syntax via the new commands START- and LOAD-EXECUTABLE-PROGRAM. In this case the two new commands must also be locked.

```
/set-logon-parameters tsos,... ——————————————————————————————————————————  (1)
 ...
/start-sdf-a ————————————————————————————————————————————————————————————  (2)
% %  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*create ————————  (3)
//remove *command((load,exec,load-prog,start-prog,load-exec,start-exec))  (4)
//end
```

1. A task is initiated under the privileged user ID TSOS.
2. SDF-A is loaded and started.
3. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is opened as a new file to be created. By default, the activated system syntax file is assigned as a reference file. The command definitions appearing in the reference file may be modified in the open group syntax file.
4. The commands LOAD, EXECUTE, LOAD-PROGRAM, START-PROGRAM, LOAD- and START-EXECUTABLE-PROGRAM are disabled.

```
/mod-file-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all    (5)
/mod-user example,profile-id=user1 ────────────────────────────────────── (6)
/mod-sdf-parameters scope=*permanent,
         syntax-file-type=*group(sys.sdf.group.syntax.example,user1) ──  (7)
%  CMD0681 SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
 PARAMETER FILE '$.SYSPAR.SDF'
%  CMD0718 GROUP SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
 ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/exit-job
```

5. File SYS.SDF.GROUP.SYNTAX.EXAMPLE is declared as shareable. Only read access is allowed.
6. The profile ID USER1 is assigned to user ID EXAMPLE.
7. Group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is assigned to profile ID USER1. This assignment is permanently stored in the SDF parameter file.

```
/set-logon-parameters example ─────────────────────────────────────────── (8)
/show-sdf-options ─────────────────────────────────────────────────────── (9)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%               VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/start-prog $edt ———————————————————————————————————————————————————— (10)
%  CMD0086 OPERATION NAME 'START-PROG' REMOVED BY USER
/start-edt —————————————————————————————————————————————————————————— (11)
%  CMD0086 OPERATION NAME 'START-PROGRAM' REMOVED BY USER
/load-prog $edt
%  CMD0086 OPERATION NAME 'LOAD-PROG' REMOVED BY USER
/exec $edt —————————————————————————————————————————————————————————— (12)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
/load $edt
%  CMD0187 ABBREVIATION OF OPERATION NAME 'LOAD' AMBIGUOUS WITH REGARD TO
'LOAD-ALIAS-CATALOG,LOAD-LOCAL-SUBSYSTEM-CATALOG'
/exit-job
  .
  .
```

8. A task is initiated under the user ID EXAMPLE.
9. The activated syntax files are listed. SYS.SDF.GROUP.SYNTAX.EXAMPLE, the group syntax file prepared beforehand by the privileged user ID TSOS, is activated.
10. SDF does not accept the START-PROG command.
11. SDF likewise does not accept the START-EDT command because START-EDT calls a procedure which in turn calls the START-PROGRAM command.
12. Since the EXEC command was removed, SDF interprets the user input as the SDF-P command EXEC-CMD and rejects it due to the invalid syntax.

3.1.2 Example 2: Changing a default value

When a new file is created with the CREATE-FILE command, access to the file under other user IDs is prohibited by default (USER-ACCESS=*OWNER-ONLY*). In this case, however, for files created under the user ID EXAMPLE, the default value is to be changed so that access from other user IDs is possible.

This can be achieved by creating a user syntax file for the user ID EXAMPLE, in which the CREATE-FILE command is modified accordingly.

Note

Default values can be very flexibly (i.e. without modifying syntax files) defined as task-specific default values. Task-specific default values are only evaluated when entered in the interactive dialog, however. In the following example, the default value could also have been defined as a task-specific default value:

```
/!create-file user-access=*all-users
```

See the "Introductory Guide to the SDF Dialog Interface’ manual [1] for more detailed information on task-specific default values.

```
/set-logon-parameters example,... ———————————————————————————————— (1)
 .
 .
/create-file demo.1 —————————————————————————————————————————————— (2)
/show-f-attr demo.1,inf=*par(security=*yes) ———————————————————————— (3)
%0000000003 :2OSG:$EXAMPLE.DEMO.1
%  ------------------------------- SECURITY     ------------------------------
%  READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = NONE
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=        3 FRE=        3 REL=        3 PAGES
 .
 .
/start-sdf-a ————————————————————————————————————————————————————— (4)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sdf.user.syntax,,*crea ——————————————————————————— (5)
```

(1) A task is initiated under the user ID EXAMPLE.

(2) Using the CREATE-FILE command, a catalog entry is created for the file DEMO.1.

(3) The protection attributes of the file DEMO.1 are displayed. It is not shareable (USER-ACC = *OWNER-ONLY*).

(4) SDF-A is loaded and started.

(5) The user syntax file SDF.USER.SYNTAX is opened as a new file to be created. Any user syntax file with this name cataloged under the user ID EXAMPLE is automatically activated for tasks of the user ID EXAMPLE when the LOGON command is processed. By default, the activated system syntax file and the activated group syntax file are assigned as reference files. The command definitions appearing in the reference files may be modified in the open user syntax file.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ——————————————————— (6)
PROTECTION = *STD
    *STD or *PARAMETERS()
    Specifies the protection attributes of the file
    STRUCTURE: *PARAMETERS
        PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
          .
          .
          .
        USER-ACCESS = *BY-PROTECTION-ATTR
            *BY-PROTECTION-ATTR or *OWNER-ONLY or *ALL-USERS or
             *SPECIAL
            Specifies whether external user IDs may access the file
        BASIC-ACL = *BY-PROTECTION-ATTR
          .
          .
          .
//edit *oper(prot,orig=*com(create-file)) ——————————————————————————— (7)
//mod-oper def='PARAMETERS' ————————————————————————————————————————— (8)
//edit *oper(user-acc) —————————————————————————————————————————————— (9)
//mod-oper def='ALL-USERS' ——————————————————————————————————————————(10)
```

(6) The PROTECTION operand of the CREATE-FILE command is displayed in its most detailed form.

(7) The file is positioned to the PROTECTION operand of the CREATE-FILE command, i.e. this operand becomes the current object in the user syntax file being processed (SDF.USER.SYNTAX).

(8) The operand that is the current object (PROTECTION) is to have the default value PARAMETERS. This change is necessary so that the subsequent modification (see steps 9 and 10) will work even when PARAMETERS, the value introducing the structure, is not explicitly specified. The value STD implicitly includes the specification USER-ACCESS=*BY-PROTECTION-ATTR (corresponding to USER-ACCESS= *OWNER-ONLY when no other default protection is defined with SECOS).

(9) The file is positioned to the USER-ACCESS operand of the command currently being processed, CREATE-FILE, i.e. this operand becomes the current object in the open user syntax file (SDF.USER.SYNTAX).

(10) The operand that is the current object (USER-ACCESS) is to have ALL-USERS as its default value.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ——————————————————— (11)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          USER-ACCESS = *ALL-USERS
              *BY-PROTECTION-ATTR or *OWNER-ONLY or *ALL-USERS or
              *SPECIAL
              Specifies whether external user IDs may access the file
          BASIC-ACL = *BY-PROTECTION-ATTR
             .
             .
             .
//end
/mod-sdf-opt synt-file=*add(*std) ——————————————————————————————————— (12)
/create-file demo.2 ————————————————————————————————————————————————— (13)
/show-f-attr demo.2,inf=*par(security=*yes) ————————————————————————— (14)
%0000000003 :2OSG:$EXAMPLE.DEMO.2
%  ------------------------------ SECURITY     -----------------------------
%  READ-PASS  = NONE        WRITE-PASS = NONE        EXEC-PASS  = NONE
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE       ACL        = NO
%  AUDIT      = NONE        FREE-DEL-D = *NONE       EXPIR-DATE = NONE
%  DESTROY    = NO          FREE-DEL-T = *NONE       EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=        3 FRE=        3 REL=        3 PAGES
 .
 .
/exit-job
```

(11) The PROTECTION operand of the command currently being processed, CREATE-FILE, is displayed in its most detailed form.

(12) The user syntax file $EXAMPLE.SDF.USER.SYNTAX is activated.

(13) A user catalog entry for the file DEMO.2 is created with the CREATE-FILE command.

(14) The protection attributes of the file DEMO.2 are displayed. It is shareable (USER-ACC = *ALL-USERS*).

3.1.3 Example 3: Enforcing file protection using four-character passwords

The users of the user IDs EXAMPLE and EXAMP1 are to be forced, for reasons of data security, to protect their files using four-byte passwords. This can be achieved by defining suitable restrictions in a group syntax file which will be assigned to these user IDs.

The definitions of the commands CREATE-FILE, CREATE-FILE-GROUP, MODIFY-FILE-ATTRIBUTES and MODIFY-FILE-GROUP-ATTRIBUTES are to be modified accordingly in the group syntax file. The definitions of the old commands CATALOG and FILE cannot be modified in this way. These commands must therefore be disabled. Since they can be called with the CMD macro, a general disabling would have unforeseeable consequences. In addition, they should continue to be available for batch mode. For this reason, the commands are merely disabled for interactive mode.

```
/set-logon-parameters sdfusr,... ————————————————————————————————————————— (1)
 .
 .
/start-sdf-a —————————————————————————————————————————————————————————————— (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,group,*crea ——————————————— (3)
//edit *command(catalog) ————————————————————————————————————————————————— (4)
//modify-cmd dial-allow=*n,dial-proc-allow=*n ———————————————————————————— (5)
//edit *command(file) ———————————————————————————————————————————————————— (6)
//modify-cmd dial-allow=*n,dial-proc-allow=*n
```

1. A task is initiated under the user ID SDFUSR.
2. SDF-A is loaded and started.
3. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is opened as a new file to be created. By default, the activated system syntax file is assigned as a reference file for the processing that follows. If no reference file were assigned, the processing that follows would have to be performed somewhat differently.
4. The file is positioned to the CATALOG command, i.e. this command becomes the current object.
5. The command that is the current object (CATALOG) is disabled for interactive mode. It is also made invalid within procedures executing in interactive mode.
6. The FILE command becomes the current object and is then disabled for interactive operation.

```
//show *oper(prot,orig=*com(create-file)),siz=*max ————————————————————— (7)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          WRITE-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                write access
          READ-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                read access
          EXEC-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                execution
          DESTROY-BY-DELETE = *BY-PROTECTION-ATTR
               *BY-PROTECTION-ATTR or *NO or *YES
             .
             .
             .
//edit *oper(prot,orig=*com(create-file)) ——————————————————————————————— (8)
//modify-oper default='PARAMETERS' —————————————————————————————————————— (9)
```

7. The PROTECTION operand of the CREATE-FILE command is displayed in its most detailed form.
8. The file is positioned to the PROTECTION operand of the CREATE-FILE command, i.e. this operand becomes the current object in the open group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE.
9. The operand that is the current object is to have PARAMETERS as its default value. STD is no longer the default value for the PROTECTION operand. Then the first operand value defined for PROTECTION becomes the current object. This is the input alternative STD.

```
//remove *value ———————————————————————————————————————————————————— (10)
//edit *oper(write-pass) ——————————————————————————————————————————— (11)
//modify-oper default=*n,struct-impl=*y ———————————————————————————— (12)
//remove *value ———————————————————————————————————————————————————— (13)
//edit *value(write-pass,*c-string) ———————————————————————————————— (14)
//modify-value *c-string(short-l=4,long-l=4) ——————————————————————— (15)
```

10. The definition of the operand value that is the current object (*STD) is deleted.
11. The file is positioned to the WRITE-PASSWORD operand of the CREATE-FILE command, i.e. this operand becomes the current object in the open group syntax file (SYS.SDF.GROUP.SYNTAX.EXAMPLE). Of course, the WRITE-PASSWORD operand is situated within a structure. Since the operand's name is unique within the entire command, neither the name of its parent operand (PROTECTION) nor PARAMETERS (which introduces the structure) need be given. The explicit specification of CREATE-FILE is not necessary, since SDF-A assumes the CREATE-FILE command by default on the basis of the preceding statements.
12. The operand that is the current object is to have no default value. The previous default value *BY-PROT-ATTR-OR-NONE (corresponding to *NONE when no other default protection is defined with SECOS) is now no longer the default value of the WRITE-PASSWORD operand. If the WRITE-PASSWORD operand is now specified on input, the structure PARAMETERS will be implicitly selected. Next, the first operand value defined for WRITE-PASSWORD becomes the current object. This is the input alternative of the type NONE.
13. The definition of the operand value that is the current object (*BY-PROT-ATTR-OR-NONE) is deleted.
14. The file is positioned to the input alternative of the type C-STRING for the WRITE-PASSWORD operand in the CREATE-FILE command, i.e. this operand value becomes the current object in the group syntax file being processed, i.e. SYS.SDF.GROUP.SYNTAX.EXAMPLE. Of course, the WRITE-PASSWORD operand is situated within a structure. Since the operand's name is unique within the entire command, neither the name of its parent operand (PROTECTION) nor PARAMETERS (which introduces the structure) need be given. The explicit specification of CREATE-FILE is not necessary, since SDF-A assumes the CREATE-FILE command by default on the basis of the preceding statements.
15. The operand value that is the current object is defined to be of the type C-STRING and to have both a minimum and a maximum length of four bytes. Following this, the next operand value defined for WRITE-PASSWORD becomes the current object. This is the input alternative of the type X-STRING.

```
//modify-value *x-string(short-l=4,long-l=4) ——————————————————————————— (16)
//remove *value ————————————————————————————————————————————————————————— (17)
//edit *oper(read-pass) ————————————————————————————————————————————————— (18)
//modify-oper default=*n,struct-impl=*y
//remove *value
//edit *value(read-pass,*c-string)
//modify-value *c-string(4,4)
//modify-value *x-string(4,4)
//remove *value
//edit *oper(exec-pass) ————————————————————————————————————————————————— (19)
//modify-oper default=*n,struct-impl=*y
//remove *value
//edit *value(exec-pass,*c-string)
//modify-value *c-string(4,4)
//modify-value *x-string(4,4)
//remove *value
```

16. The operand value that is the current object is defined to be of the type X-STRING and to have both a minimum and a maximum length of four bytes. Following this, the next operand value defined for WRITE-PASSWORD becomes the current object. This is the input alternative of the type INTEGER.
17. The definition of the operand value that is the current object (INTEGER) is deleted.
18. The READ-PASSWORD operand and its values are modified in exactly the same manner as was the WRITE-PASSWORD operand earlier on (see steps 12-17).
19. The EXEC-PASSWORD operand and its values are modified in exactly the same manner as was the WRITE-PASSWORD operand earlier on (see steps 12-17).

```
//show *oper(prot,orig=*com(create-file)),siz=*max ——————————————————— (20)
PROTECTION = *STD
     *STD or *PARAMETERS()
     Specifies the protection attributes of the file
     STRUCTURE: *PARAMETERS
          PROTECTION-ATTR = *BY-DEF-PROT-OR-STD
             .
             .
             .
          WRITE-PASSWORD =
               *NONE or c-string_4..4 or x-string_7..8 or *SECRET
               Specifies the password for protection against unauthorized
                write access
          READ-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                read access
          EXEC-PASSWORD =
               *BY-PROT-ATTR-OR-NONE or *NONE or c-string_1..4 or
               x-string_1..8 or integer_-2147483648..2147483647 or
               *SECRET -default-: *BY-PROT-ATTR-OR-NONE
               Specifies the password for protection against unauthorized
                execution
          DESTROY-BY-DELETE = *BY-PROTECTION-ATTR
               *BY-PROTECTION-ATTR or *NO or *YES
             .
             .
             .
 .
 .  ————————————————————————————————————————————————————————————————————— (21)
 .
//end
/mod-file-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all   (22)
/exit-job
 .
 .
```

20. The PROTECTION operand of the CREATE-FILE command is displayed in its most detailed form.
21. The definitions of the commands CREATE-FILE-GROUP, MODIFY-FILE-ATTRIBUTES and MODIFY-FILE-GROUP-ATTRIBUTES are modified in exactly the same manner as for the definition of the CREATE-FILE command earlier on.
22. The file SYS.SDF.GROUP.SYNTAX.EXAMPLE is declared as shareable. It may only be accessed for reading.

```
/set-logon-parameters tsos,...  ──────────────────────────────────── (23)
/copy-file from-file=$sdfusr.sys.sdf.group.syntax.example,to-file=-
/sys.sdf.group.syntax.example,prot=*same  ─────────────────────────── (24)
/modify-user example,profile-id=user1  ─────────────────────────────── (25)
/modify-sdf-param scope=*temporary,syntax-file=*group-
/(sys.sdf.group.syntax.example,user1)  ─────────────────────────────── (26)
/modify-user examp1,profile-id=user1  ──────────────────────────────── (27)
/exit-job
 .
 .
```

23. A task is initiated under the privileged user ID TSOS.
24. The group syntax file $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE generated under the user ID SDFUSR is copied. The name of the copy is $TSOS.SYS.SDF.GROUP.SYNTAX.EXAMPLE. It has the same protection attributes as the original file.
25. Profile ID USER1 is assigned to user ID EXAMPLE.
26. Group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is assigned to profile ID USER1.
27. Profile ID USER1 is assigned to user ID EXAMP1.

```
/set-logon-parameters example ——————————————————————————————————————————— (28)
/show-sdf-options ——————————————————————————————————————————————————————— (29)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE          : *EXPERT
%  LOGGING           : *INPUT-FORM
%  CONTINUATION      : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING      : *NO
%  MODE              : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS     : *STYLE-GUIDE-MODE
%  INPUT-HISTORY     : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/catalog demo ——————————————————————————————————————————————————————————— (30)
%  CMD0087 OPERATION NAME 'CATALOG' IS NOT PERMITTED AT THE MOMENT
/file demo —————————————————————————————————————————————————————————————— (31)
%  CMD0087 OPERATION NAME 'FILE' IS NOT PERMITTED AT THE MOMENT
```

28. A task is initiated under the user ID EXAMPLE.
29. The activated syntax files are listed. The group syntax file $.SYS.SDF.GROUP.SYNTAX.EXAMPLE has been activated.
30. SDF does not accept the CATALOG command. Since it has been disabled for interactive mode, it is treated as unknown.
31. SDF does not accept the FILE command. Since it has been disabled for interactive mode, it is treated as unknown.

```
/create-file demo,wr-pass=2,ex-pass='3' ———————————————————————————————— (32)
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:WRITE-PASSWORD'
%  CMD0064 OPERAND VALUE 'P' DOES NOT MATCH DATA TYPE 'C-STRING_4..4 OR
 X-STRING_7..8 OR SECRET'
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:READ-PASSWORD'
%  CMD0099 MANDATORY OPERAND MISSING OR INVALID
%  CMD0051 INVALID OPERAND 'PROTECTION=PARAMETERS:EXEC-PASSWORD'
%  CMD0062 LENGTH OF VALUE ''P'' NOT IN PERMISSIBLE RANGE FOR DATA TYPE
 'C-STRING_4..4'
/create-file demo,read-pass='1234',wr-pass='2345',ex-pass='3456' —————— (33)
/show-file-attr demo,inf=*par(security=*yes) ——————————————————————————— (34)
%0000000003 :2OSG:$EXAMPLE.DEMO
%  ------------------------------- SECURITY     ------------------------------
%  READ-PASS  = YES          WRITE-PASS = YES          EXEC-PASS  = YES
%  USER-ACC   = OWNER-ONLY  ACCESS     = WRITE        ACL        = NO
%  AUDIT      = NONE         FREE-DEL-D = *NONE        EXPIR-DATE = NONE
%  DESTROY    = NO           FREE-DEL-T = *NONE        EXPIR-TIME = NONE
%  SP-REL-LOCK= NO
%:2OSG: PUBLIC:      1 FILE  RES=         3 FRE=         3 REL=         3 PAGES
 .
 .
/exit-job
```

32. SDF does not accept the CREATE-FILE command, because
    – the specified write password is neither equal to the keyword SECRET nor of the type C-STRING or X-STRING, and
    – because no read password has been specified, and
    – the specified execute password of type C-STRING is only 1 byte long.
33. SDF accepts the CREATE-FILE command, because passwords of the type C-STRING that are four bytes long have been specified.
34. A catalog entry has been created for the file DEMO. It is protected by passwords.

3.1.4 Example 4: Permitting only one program (EDT)

Users with the user ID EXAMPLE are to be permitted to load and start only one program, in this case, the file editor EDT, which is cataloged under the privileged user ID TSOS.

The desired restriction may be put into effect via the group syntax file $TSOS.SYS.SDF.GROUP.SYNTAX.EXAMPLE. In this file, the definitions of the commands START-PROGRAM and LOAD-PROGRAM must be modified accordingly.

- The START-PROGRAM command is to be renamed to START-EDITOR and have no visible operands. The START-EDITOR command must not be already defined in the syntax file hierarchy.
- The LOAD-PROGRAM command is to retain its name. Its operands, with the exception of FROM-FILE, are to be invisible.
- In BLSSERV V2.3 and higher the functionality of START-PROGRAM and LOAD-PROGRAM are also offered with improved syntax via the new commands START- and LOAD-EXECUTABLE-PROGRAM. In this case the two new commands must also be locked or their syntax definitions are to be modified in the same manner as for START-PROGRAM and LOAD-PROGRAM.
- The old commands EXECUTE and LOAD and the START commands of other programs are disabled so that the defined limitation cannot be bypassed.

```
/set-logon-parameters tsos,...  ---------------------------------------- (1)
 .
 .
/start-sdf-a  ---------------------------------------------------------- (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*crea  ---------- (3)
//remove *command((load,exec))  ---------------------------------------- (4)
//show *command(start-prog)  ------------------------------------------- (5)
```

1. A task is initiated under the privileged user ID TSOS.
2. SDF-A is loaded and started.
3. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is opened as a new file to be created. By default, the activated system syntax file is assigned as a reference file. The command definitions contained in the reference file can be modified in the open group syntax file.
4. The commands LOAD and EXECUTE are disabled.
5. The command START-PROGRAM is displayed. By default, the display is at the MINIMUM level of detail (see page 65).

```
START-PROGRAM(SRPG,SR,START-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     FROM-FILE =
          STRUCTURE: *MODULE
               LIBRARY = *DBL-DEFAULT
                    STRUCTURE: *LINK
                         LINK =
               ELEMENT-OR-SYMBOL(ELEMENT,ELEM) = *ALL
                    STRUCTURE: composed-name
                         VERSION = *STD
                    STRUCTURE: c-string
                         VERSION = *STD
               PROGRAM-MODE = *DBL-DEFAULT
               RUN-MODE = *DBL-DEFAULT
                    STRUCTURE: *ADVANCED
                         ALTERNATE-LIBRARIES = *DBL-DEFAULT
                         NAME-COLLISION = *DBL-DEFAULT
                         UNRESOLVED-EXTRNS = *DBL-DEFAULT
                         ERROR-EXIT = *DBL-DEFAULT
                         MESSAGE-CONTROL = *DBL-DEFAULT
                         LOAD-INFORMATION = *DBL-DEFAULT
                         PROGRAM-MAP = *DBL-DEFAULT
                              STRUCTURE: *SYSLST
                                   SYSLST-NUMBER = *STD
                              STRUCTURE: *BOTH
                                   SYSLST-NUMBER = *STD
                         SHARE-SCOPE = *DBL-DEFAULT
                              STRUCTURE: *MEMORY-POOL
                                   SCOPE = *ALL
                         IGNORE-ATTRIBUTES = *DBL-DEFAULT
                         REP-FILE = *DBL-DEFAULT
                         AUTOLINK = *DBL-DEFAULT
                         PROGRAM-VERSION = *DBL-DEFAULT
          STRUCTURE: *PHASE
               LIBRARY =
               ELEMENT =
               VERSION = *STD
     CPU-LIMIT = *JOB-REST
     TEST-OPTIONS = *DBL-DEFAULT
     MONJV = *NONE
     RESIDENT-PAGES = *PARAMETERS
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
               MAXIMUM = *STD
     VIRTUAL-PAGES = *STD
```

```
//show *operand(from-f,orig=*com(start-prog)),siz=*med,att-inf=*n  ———  (6)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//edit *oper(from-f,orig=*com(start-prog))  ——————————————————————————  (7)
//mod-oper def='$edt',pres=*intern  ——————————————————————————————————  (8)
//edit *oper(cpu-lim)  ———————————————————————————————————————————————  (9)
//mod-oper pres=*intern
//edit *oper(test-opt)
//mod-oper pres=*intern
//edit *oper(monjv)
//mod-oper pres=*intern
//edit *oper(resid-p)
//mod-oper pres=*intern
//edit *oper(resid-p,par,min)  ——————————————————————————————————————  (10)
//mod-oper pres=*intern
//edit *oper(resid-p,par,max)
//mod-oper pres=*intern
//edit *oper(virt-p)
//mod-oper pres=*intern
```

6. The FROM-FILE operand of the START-PROGRAM command is displayed at the medium level of detail. Output of the structures attached to the values MODULE and PHASE is suppressed.
7. The file is positioned to the FROM-FILE operand of the START-PROGRAM command. This operand becomes the current object in the open syntax file.
8. The operand that is the current object (FROM-FILE) is assigned the default value '$EDT'. At the user interface of the command the operand is made invisible, provided it has a default value.
9. The other operands of the START-PROGRAM command are also made invisible. The command name need not be specified for positioning as the specification made in step 7 is still effective.
10. The RESIDENT-PAGES operand, which is made invisible, has a default value to which a structure is attached. The operands included in this structure must also be made invisible.

```
//show *com(start-prog),att-inf=*n,size=*max ——————————————————————————— (11)
START-PROGRAM(SRPG,SR,START-PROG)
     Loads a program (load or object module) to the memory and starts it
//edit *com(start-prog) ————————————————————————————————————————————————— (12)
//mod-cmd start-editor,help=(e('Loads the EDT to the memory and starts -
//it.'),d('Laedt den EDT in den Speicher und startet ihn.')) ——————————— (13)
//show *com(start-editor) ——————————————————————————————————————————————— (14)
```

11. The header of the START-PROGRAM command is displayed with the corresponding help text.
12. The file is positioned to the START-PROGRAM command.
13. The name of the command which is the current object (START-PROGRAM) is modified to START-EDITOR. The help texts are modified in accordance with the restricted functional scope. If a START-EDITOR command has already been defined in the system, the statement does not appear.
14. The START-EDITOR command is displayed (see next page). The display has the MINIMUM level of detail. By default it is assigned its previous name START-PROGRAM and several abbreviations. The operands of the structure FROM-FILE=*MODULE(...) and FROM-FILE=*PHASE(...) are still displayed in SDF-A because they are defined using PRESENCE=*NORMAL. However, these operands are no longer visible on the user interface and can no longer be entered.

```
START-EDITOR(SRPG,SR,START-PROG,START-PROGRAM)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
               LIBRARY = *STD
                    STRUCTURE: *LINK
                         LINK =
               ELEMENT = *ALL
                    STRUCTURE: filename
                         VERSION = *STD
                    STRUCTURE: c-string
                         VERSION = *STD
               PROGRAM-MODE = *24
               RUN-MODE = *STD
                    STRUCTURE: *ADVANCED
                         ALTERNATE-LIBRARIES = *NO
                         NAME-COLLISION = *STD
                         UNRESOLVED-EXTRNS = *STD
                         ERROR-EXIT = X'FFFFFFFF'
                         MESSAGE-CONTROL = *INFORMATION
                         LOAD-INFORMATION = *DEFINITIONS
                         PROGRAM-MAP = *NO
                              STRUCTURE: *SYSLST
                                   SYSLST-NUMBER = *STD
```

```
                                          STRUCTURE: *BOTH
                                               SYSLST-NUMBER = STD
                                     SHARE-SCOPE = *SYSTEM-MEMORY
                                          STRUCTURE: *MEMORY-POOL
                                               SCOPE = *ALL
                                     IGNORE-ATTRIBUTES = *NONE
                                     REP-FILE = *NONE
                                     AUTOLINK = *YES
                           LIBRARY =
                           ELEMENT =
                           VERSION = *STD
//remove *com((start-archive,start-binder,start-hsms,...)) ———————————— (15)
//show *com(load-prog) ———————————————————————————————————————————————— (16)
```

15. All the other (still functioning) START commands must be disabled so that only the program EDT can be started. The START commands of many programs are eliminated without being disabled using the command
    `%  CMD0185 OPERAND NAME 'FROM-FILE' COULD NOT BE IDENTIFIED.`
16. The LOAD-PROGRAM command is displayed. The display has the MINIMUM level of detail (see page 69).

```
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     FROM-FILE =
          STRUCTURE: *MODULE
               LIBRARY = *DBL-DEFAULT
                    STRUCTURE: *LINK
                         LINK =
               ELEMENT-OR-SYMBOL(ELEMENT,ELEM) = *ALL
                    STRUCTURE: composed-name
                         VERSION = *STD
                    STRUCTURE: c-string
                         VERSION = *STD
               PROGRAM-MODE = *DBL-DEFAULT
               RUN-MODE = *DBL-DEFAULT
                    STRUCTURE: *ADVANCED
                         ALTERNATE-LIBRARIES = *DBL-DEFAULT
                         NAME-COLLISION = *DBL-DEFAULT
                         UNRESOLVED-EXTRNS = *DBL-DEFAULT
                         ERROR-EXIT = *DBL-DEFAULT
                         MESSAGE-CONTROL = *DBL-DEFAULT
                         LOAD-INFORMATION = *DBL-DEFAULT
                         PROGRAM-MAP = *DBL-DEFAULT
                              STRUCTURE: *SYSLST
                                   SYSLST-NUMBER = *STD
                              STRUCTURE: *BOTH
                                   SYSLST-NUMBER = *STD
                         SHARE-SCOPE = *DBL-DEFAULT
                              STRUCTURE: *MEMORY-POOL
                                   SCOPE = *ALL
                         IGNORE-ATTRIBUTES = *DBL-DEFAULT
                         REP-FILE = *DBL-DEFAULT
                         AUTOLINK = *DBL-DEFAULT
                         PROGRAM-VERSION = *DBL-DEFAULT
          STRUCTURE: *PHASE
               LIBRARY =
               ELEMENT =
               VERSION = *STD
     CPU-LIMIT = *JOB-REST
     TEST-OPTIONS = *DBL-DEFAULT
     MONJV = *NONE
     RESIDENT-PAGES = *PARAMETERS
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
               MAXIMUM = *STD
     VIRTUAL-PAGES = *STD
```

```
//show *oper(from-f,orig=*com(load-prog)),siz=*med,att-inf=*n ———————— (17)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//edit *oper(from-f,orig=*com(load-prog)) ————————————————————————————— (18)
//mod-oper def='$edt' —————————————————————————————————————————————————— (19)
//mod-value value=('$edt','edt'(outp='$edt')) —————————————————————————— (20)
//remove *value ———————————————————————————————————————————————————————— (21)
//remove *value(from-f,phase) —————————————————————————————————————————— (22)
//edit *oper(cpu-lim) ————————————————————————————————————————————————— (23)
//mod-oper pres=*intern
//edit *oper(test-opt)
//mod-oper pres=*intern
//edit *oper(monjv)
//mod-oper pres=*intern
//edit *oper(resid-p)
//mod-oper pres=*intern
//edit *oper(resid-p,par,min)
//mod-oper pres=*intern
//edit *oper(resid-p,par,max)
//mod-oper pres=*intern
```

17. The FROM-FILE operand of the LOAD-PROGRAM command is displayed at the medium level of detail. Output of the structures attached to the values MODULE and PHASE is suppressed.
18. The file is positioned to the FROM-FILE operand of the LOAD-PROGRAM command. This operand becomes the current object in the open syntax file.
19. The operand that is the current object (FROM-FILE) is assigned the default value '$EDT'. Subsequently, the definition of the first operand value attached to FROM-FILE (FILENAME) becomes the current object.
20. $EDT and EDT are defined as the only permissible values for the operand value definition that is the current object (FILENAME). For the input value EDT it is defined that instead of this value SDF passes value $EDT to the implementation. Subsequently, the definition of the next operand value attached to FROM-FILE (MODULE) becomes the current object.
21. The definition of the operand value that is the current object (MODULE) is deleted. **Note:** after this, the definition of the operand value FILENAME becomes the current object.
22. The definition of the operand value PHASE of the FROM-FILE operand of the LOAD-PROGRAM command is deleted. To do this, it is not necessary to specify the command name as the specification made in step 18 is still effective.
23. The other operands of the LOAD-PROGRAM command are made invisible.

```
//edit *oper(virt-p)
//mod-oper pres=*intern
//show *com(load-prog),att-info=*y,siz=*max ———————————————————————————— (24)
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
     loads a program (load or  object module) to the memory
     FROM-FILE = $EDT
          $EDT or EDT
          name of the file containing the load  module or details of the
          object module/load module library
//end
/mod-f-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all ——— (25)
/mod-user example,profile-id=user1 ————————————————————————————————————— (26)
/mod-sdf-parameters scope=*permanent,syntax-file=*group-
/(sys.sdf.group.syntax.example,user1) —————————————————————————————————— (27)
%  CMD0681 SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
PARAMETER FILE '$.SYSPAR.SDF'
%  CMD0718 GROUP SYNTAX FILE '$.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/start-prog $example.edt ——————————————————————————————————————————————— (28)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//end
/exit-job
```

24. The LOAD-PROGRAM command is displayed. The display has the maximum level of detail. The only operand visible at the user interface is FROM-FILE, which has the default value $EDT. The values permitted are $EDT and EDT.
25. The file SYS.SDF.GROUP.SYNTAX.EXAMPLE is declared as shareable. Access to it is permitted only for reading.
26. Profile ID USER1 is assigned to user ID EXAMPLE.
27. Group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is assigned to profile ID USER1. This assignment is permanently stored in the SDF parameter file.
28. The file $EXAMPLE.EDT contains the program SDF-A. The privileged user ID TSOS may access this file because the START-PROGRAM command is not restricted for TSOS. The following shows that a user under the user ID EXAMPLE cannot access a program with the name EDT, even if this program exists under his/her own user ID. Due to the restrictions implemented above, only the program $EDT can be loaded.

```
/set-logon-parameters example,... ———————————————————————————————————— (29)
/show-sdf-options ——————————————————————————————————————————————————— (30)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : UNDEFINED
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE          : *EXPERT
%  LOGGING           : *INPUT-FORM
%  CONTINUATION      : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING      : *NO
%  MODE              : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS     : *STYLE-GUIDE-MODE
%  INPUT-HISTORY     : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
/exec $sdf-a ————————————————————————————————————————————————————————— (31)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
```

29. A task is initiated under the user ID EXAMPLE.
30. The activated syntax files are displayed. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE previously processed by the privileged user ID TSOS is activated.
31. Since the EXEC command was removed, SDF interprets the user input as the SDF-P command EXEC-CMD and rejects it due to the invalid syntax.

```
/load-program $sdf-a ──────────────────────────────────────────────── (32)
%  CMD0051 INVALID OPERAND 'FROM-FILE'
%  CMD0063 OPERAND VALUE '$SDF-A' NOT A MEMBER OF THE SINGLE VALUE LIST OF
 SCOPE '$EDT OR EDT'
/load-program edt ─────────────────────────────────────────────────── (33)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1996. ALL RIGHTS
RESERVED
/load-program ─────────────────────────────────────────────────────── (34)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1996. ALL RIGHTS
RESERVED
/start-program $sdf-a ─────────────────────────────────────────────── (35)
%  CMD0376 SPECIFICATION OF POSITIONAL OPERANDS UP FROM POSITION '1' NOT
 PERMITTED
/start-editor ─────────────────────────────────────────────────────── (36)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
.
 .
halt
%  EDT8000 EDT TERMINATED
/start-program ────────────────────────────────────────────────────── (37)
%  BLS0500 PROGRAM 'EDT', VERSION '16.6A' OF '1996-06-04' LOADED
.
 .
halt
%  EDT8000 EDT TERMINATED
/exit-job
```

32. SDF does not accept the operand value $SDF-A in the LOAD-PROGRAM command.
33. SDF does not pass the value EDT to the implementation, but instead the value $EDT. Instead of the program SDF-A contained in the file $EXAMPLE.EDT (see step 28) the program contained in the file $TSOS.EDT is loaded.
34. SDF transfers the default value $EDT of the FROM-FILE operand to the implementation. EDT is loaded.
35. START-PROGRAM is the default name of the renamed command START-EDITOR. SDF recognizes the START-EDT command and rejects the operand value $SDF-A as not permissible.
36. Entering START-EDITOR causes EDT to be loaded and started.
37. SDF recognizes the START-EDITOR command by its default name START-PROGRAM. EDT is loaded and started.

3.1.5 Example 5: Restricting the set of usable programs

Users with the user ID EXAMPLE are to be permitted to load and start only those programs that are cataloged under the user ID SDFUSR. All of these programs, without exception, reside in files and not in libraries.

The desired restriction may be implemented via the group syntax file $TSOS.SYS.SDF.GROUP.SYNTAX.EXAMPLE. It is to receive the version number EXAMPLE#5. In this file, the definitions of the commands START-PROGRAM and LOAD-PROGRAM are to be suitably modified. Since users with the user ID EXAMPLE do not test programs, the operand TEST.OPTIONS is to be disabled when the two command definitions are modified.

Note

In BLSSERV V2.3 and higher the functionality of START-PROGRAM and LOAD-PROGRAM are also offered with improved syntax via the new commands START- and LOAD-EXECUTABLE-PROGRAM. In this case the two new commands must also be locked or their syntax definitions are to be modified in the same manner as for START-PROGRAM and LOAD-PROGRAM.

The definitions of the old commands EXECUTE and LOAD cannot be modified as desired. If EXECUTE and LOAD were to be disabled only for interactive and batch mode but not for calls using the CMD macro, such disabling could easily be bypassed. Consequently, the commands are to be generally disabled.

User guidance is preset throughout the system to GUIDANCE=*EXPERT. For the user ID EXAMPLE, user guidance is to be preset to GUIDANCE=*NO.

The program EDT is available under the user ID SDFUSR.

```
/set-logon-parameters sdfusr,... ———————————————————————————————————— (1)
 .
 .
/start-sdf-a ———————————————————————————————————————————————————————— (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sys.sdf.group.syntax.example,*group,*crea ———————— (3)
```

1. A task is initiated under the user ID SDFUSR.
2. SDF-A is loaded and started.
3. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE is opened as a new file to be created. By default, the activated system syntax file is assigned as a reference file.

```
//set-glob vers=example#5,guid=*n ─────────────────────────────────────── (4)
//remove *com((load,exec)) ─────────────────────────────────────────────── (5)
//show *com(start-prog),att-inf=*n,size=*max ───────────────────────────── (6)
START-PROGRAM(SRPG,SR,START-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
     Loads a program (load or object module) to the memory and starts it
//edit *com(start-prog) ────────────────────────────────────────────────── (7)
//mod-cmd help=(e('Loads a $SDFUSR-program to the memory and starts it.'),-
//d('Laedt ein $SDFUSR-Programm in den Speicher und startet es.')) ──── (8)
//show *oper(from-f),siz=*med,att-inf=*n ───────────────────────────────── (9)
FROM-FILE =
     filename or *MODULE() or *PHASE()
//show *oper(from-f),impl=*y,att-inf=*n ───────────────────────────────── (10)
ADD-OPERAND NAME=FROM-FILE,INTERNAL-NAME=FROMFI,STANDARD-NAME=             -
            FROM-FILE,HELP=(D(TEXT='Name der Datei, die das Lademodul-
 enthaelt oder Angaben zur Bindemodul- bzw. Lademodulbibliothek'),E(    -
            TEXT='name of the file containing the load module or-
 specification of the object module/load module library')),             -
            RESULT-OPERAND-NAME=*POSITION(POSITION=1),                  -
            CONCATENATION-POS=1
//add-oper prefix,def='$sdfusr.',res-oper-n=*pos(1),conc-pos=1,pres=*int (11)
```

4. The group syntax file SYS.SDF.GROUP.SYNTAX.EXAMPLE receives the version number EXAMPLE#5. User guidance is preset to GUIDANCE=*NO.
5. The commands LOAD and EXECUTE are disabled for all users.
6. The SDF-A statement used to define the START-PROGRAM command is displayed.
7. The file is positioned to the START-PROGRAM command, i.e. the command becomes the current object in the open group syntax file.
8. The help text for the command that is the current object is changed. Thereafter, the first operand of this command (FROM-FILE) becomes the current object.
9. The FROM-FILE operand of the command currently being edited (START-PROGRAM) is displayed at the medium level of detail.
10. The SDF-A statement used to define the FROM-FILE operand of the START-PROGRAM command is displayed.
11. The PREFIX operand is defined. SDF-A inserts its definition into the definition of the START-PROGRAM command after the current object (FROM-FILE). At the user interface of the START-PROGRAM command, the operand is invisible. Its default value is “$SDFUSR”. When the command is passed to the implementation the operand has the same position as the FROM-FILE operand (see step 10) and, when concatenated with the latter, comes first.

```
//add-value *part-filename ——————————————————————————————————————————— (12)
//edit *oper(from-f) ————————————————————————————————————————————————— (13)
//mod-oper conc-pos=2,help=(e('specifies the name of the file holding the -
//load module.'),d('Name der Datei, die das Lademodul enthaelt.')) ——— (14)
//mod-value *filename(user-id=*n) ———————————————————————————————————— (15)
//remove *value —————————————————————————————————————————————————————— (16)
//remove *value(from-f,phase) ———————————————————————————————————————— (17)
//show *oper(test-opt) ——————————————————————————————————————————————— (18)
TEST-OPTIONS = *NONE
//edit *oper(test-opt) ——————————————————————————————————————————————— (19)
//mod-oper pres=*intern —————————————————————————————————————————————— (20)
```

12. For the PREFIX operand an operand value of type PARTIAL-FILENAME is defined.
13. The file is positioned to the FROM-FILE operand of the command currently being edited (START-PROGRAM), i.e. this operand becomes the current object.
14. The operand that is the current object (FROM-FILE) is to come second when concatenated with the PREFIX operand. The help texts for FROM-FILE are modified. Subsequently, the first operand value (FILENAME) of FROM-FILE becomes the current object.
15. The definition of the operand value that is the current object (FILENAME) is modified. Specification of the user ID as part of the file name is no longer permitted. After this, the operand value MODULE becomes the current object.
16. The definition of the operand value that is the current object (MODULE) is deleted. The structure attached to it is likewise deleted. Note: After this, the definition of the operand value FILENAME becomes the current object.
17. The definition of operand value PHASE belonging to the operand FROM-FILE is deleted.
18. The TEST-OPTIONS operand is displayed.
19. The file is positioned to the TEST-OPTIONS operand of the command currently being edited (START-PROGRAM), i.e. TEST-OPTIONS becomes the current object.
20. The operand that is the current object (TEST-OPTIONS) is made invisible at the user interface of the START-PROGRAM command.

```
//show *com(load-prog),att-inf=*n,size=*max ────────────────────────────── (21)
LOAD-PROGRAM(LDPG,LOAD-PROG)
   FROM :2OSH:$TSOS.SYSSDF.BLSSERV.023 (SYSTEM)
    Loads a program (load or object module) to the memory
//edit *com(load-prog)
//mod-cmd help=(e('Loads a $SDFUSR-program to the memory.'),-
//d('Laedt ein $SDFUSR-Programm in den Speicher.'))
//show *oper(from-f),siz=*med,att-inf=*n
FROM-FILE =
     filename or *MODULE() or *PHASE()
//show *oper(from-f),impl=*y,att-inf=*n
ADD-OPERAND NAME=FROM-FILE,INTERNAL-NAME=FROMFI,STANDARD-NAME=         -
            FROM-FILE,HELP=(D(TEXT='Name der Datei, die das Lademodul-
 enthaelt oder Angaben zur Bindemodul- bzw. Lademodulbibliothek'),E(   -
            TEXT='name of the file containing the load  module or-
 specification of the object module/load module library')),            -
            RESULT-OPERAND-NAME=*POSITION(POSITION=1),                 -
            CONCATENATION-POS=1
//copy *oper(prefix,orig=*com(start-prog)) ────────────────────────────── (22)
//edit *oper(from-f)
//mod-oper conc-pos=2,help=(e('specifies the name of the file holding -
//the load module.'),d('Name der Datei, die in das Lademodul enthaelt.'))
//mod-value *filename(user-id=*n)
//remove *value
//remove *value(from-f,phase)
//show *oper(test-opt)
TEST-OPTIONS = *NONE
//edit *oper(test-opt)
//mod-oper pres=*intern
//end
```

21. The definition of the LOAD-PROGRAM command is displayed, and modified in a way analogous to the way START-PROGRAM was modified previously.

22. The PREFIX operand and its operand value are defined for the LOAD-PROGRAM command using the COPY statement instead of the ADD statement. The definitions established in steps 11 and 12 for START-PROGRAM are copied.

```
/mod-f-attr sys.sdf.group.syntax.example,access=*read,user-acc=*all ─── (23)
/start-prog demo ──────────────────────────────────────────────────────── (24)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file ....
 .
 .
//end
/exit-job
 .
 .
/set-logon-parameters tsos,... ──────────────────────────────────────── (25)
/mod-user example,profile-id=user1 ──────────────────────────────────── (26)
/mod-sdf-param scope=*permanent,syntax-file=*group($sdfusr.sys.sdf.group.-
/syntax.example,user1) ──────────────────────────────────────────────── (27)
% CMD0681 SYNTAX FILE '$SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE' INSERTED IN
PARAMETER FILE '$.SYSPAR.SDF'
% CMD0718 GROUP SYNTAX FILE '$SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE' HAS BEEN
ASSOCIATED WITH 'PROFILE-ID USER1' IN MEMORY TABLES
/exit-job
 .
 .
```

23. The file SYS.SDF.GROUP.SYNTAX.EXAMPLE is declared as shareable. Access is read-only.
24. The program DEMO cataloged under the user ID SDFUSR is loaded and started. It is the program SDF-A.
25. A task is initiated under the system administrator ID TSOS.
26. Profile ID USER1 is assigned to user ID EXAMPLE.
27. Group syntax file $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE is assigned to profile ID USER1. The assignment is permanently stored in the SDF parameter file.

```
/set-logon-parameters example,... ———————————————————————————————————— (28)
 .
 .
%CMD:show-sdf-options ———————————————————————————————————————————————— (29)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.EXAMPLE
%              VERSION : EXAMPLE#5
%  USER      : *NONE
%CURRENT SDF OPTIONS :
%  GUIDANCE          : *NO
%  LOGGING           : *INPUT-FORM
%  CONTINUATION      : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING      : *NO
%  MODE              : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS     : *STYLE-GUIDE-MODE
%  INPUT-HISTORY     : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
%CMD:exec sdf-a ——————————————————————————————————————————————————————— (30)
%  SDP0222 OPERAND 'CMD' INVALID IN /EXEC-CMD, ERROR 'SDP0116'. IN SYSTEM
MODE: /HELP-MSG SDP0116
```

28. A task is initiated under the user ID EXAMPLE.
29. The activated syntax files are listed. The group syntax file $SDFUSR.SYS.SDF.GROUP.SYNTAX.EXAMPLE, with the version number EXAMPLE#5, is activated. User guidance is set to GUIDANCE=*NO. Consequently, SDF requests input of commands and statements by issuing “%CMD:” or “%STMT:”.
30. Since the EXEC command was removed, SDF interprets the user input as the SDF-P command EXEC-CMD and rejects it due to invalid syntax.

```
%CMD:start-prog sdf-a ————————————————————————————————————————————————— (31)
%  BLS0514 ERROR WHEN OPENING FILE $SDFUSR.SDF-A . DMS ERROR '0D33'. IN
SYSTEM MODE /HELP-MSG DMS0D33
%  NRTT101 ABNORMAL JOBSTEP TERMINATION BLS0514
%CMD:help-msg 0d33
%  DMS0D33 PROGRAM ERROR: REQUESTED FILE NOT CATALOGED
%  ? The requested file has not been cataloged in the system.
%    For the file or job variable (JV) no catalog entry could be found.
%  ! Correct the error and try again.
%CMD:start-prog $sdfusr.demo —————————————————————————————————————————— (32)
%  CMD0051 INVALID OPERAND 'FROM-FILE'
%  CMD0072 ATTRIBUTE SPECIFIED IN FILE NAME '$SDFUSR.DEMO' NOT PERMITTED
%ENTER OPERANDS:
%$sdfusr.demo
demo ————————————————————————————————————————————————————————————————— (33)
%  BLS0517 MODULE 'SDAMAIN' LOADED ————————————————————————————————————— (34)
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%STMT:open-syntax-file user,,*crea ————————————————————————————————————— (35)
```

31. SDF accepts the command name START-PROGRAM. However, it passes to the implementation the file name $SDFUSR.SDF-A, which it has formed through concatenation. This file does not exist.
32. SDF does not accept the file name $SDFUSR.DEMO, because it contains a user ID, which is not permitted.
33. SDF accepts the file name DEMO. However, SDF passes to the implementation the file name $SDFUSR.DEMO, which it has formed through concatenation.
34. The program SDF-A, found under the user ID SDFUSR in the file DEMO, is loaded and started (see step 24).
35. The user syntax file USER is created and opened. By default, the activated system syntax file and the activated group syntax file are assigned as reference files.

```
%STMT:show *com(start-prog),siz=*max ———————————————————————————————————— (36)
START-PROGRAM(SRPG,SR,START-PROG)
   FROM SYS.SDF.GROUP.EXAMPLE (GROUP)
     Loads a $SDFUSR-program to the memory and starts it.
     FROM-FILE =
          filename_1..54_without-user-id-generation
          Specifies the name of the file holding the load module.
     CPU-LIMIT = JOB-REST
          JOB-REST or integer_1..32767
          specifies the maximum CPU time in seconds the program may use for
          execution
     MONJV = *NONE
          *NONE or filename_1..54_without-generation
          specifies the name of the job variable which is to monitor the
          program.
     RESIDENT-PAGES = *PARAMETERS
          *PARAMETERS()
          specifies the number of resident memory pages required for program
          execution
          STRUCTURE: *PARAMETERS
               MINIMUM = *STD
                    *STD or integer_0..32767
                    specifies the minimum number of resident memory pages
                    required
               MAXIMUM = *STD
                    *STD or integer_0..32767
                    specifies the maximum number of resident memory pages
                    required
     VIRTUAL-PAGES = *STD
          *STD or integer_0..32767
          specifies the total number of memory pages (both resident and
          pageable) required for program execution
%STMT:end
%CMD:exit-job
```

36. The definition of the START-PROGRAM command is displayed in its most detailed form. The changes made can be seen. The TEST-OPTIONS operand has been made invisible.

3.1.6 Example 6: Clearing a lock on a command

In a user syntax file the START-SDF-A command is disabled. This lock is to be cleared.

```
/set-logon-parameters sdfusr,... ——————————————————————————————————————— (1)
 .
/show-sdf-options ————————————————————————————————————————————————————— (2)
%SYNTAX FILES CURRENTLY ACTIVATED :
%  SYSTEM    : :2OSH:$TSOS.SYSSDF.SDF.045
%              VERSION : SESD04.5A300
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACO.022
%              VERSION : SESD02.2A00
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.ACS.140
%              VERSION : SESD14.0B100
 .
 .
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.SDF-A.041
%              VERSION : SESD04.1E10
%  SUBSYSTEM : :2OSH:$TSOS.SYSSDF.TASKDATE.140
%              VERSION : SESD14.0A100
%  GROUP     : 2OSH:$.SYS.SDF.GROUP.SYNTAX.SDFUSR
%               VERSION : UNDEFINED
%  USER      : 2OSH:$SDFUSR.SDF.USER.SYNTAX
%               VERSION : USER001
%CURRENT SDF OPTIONS :
%  GUIDANCE           : *EXPERT
%  LOGGING            : *INPUT-FORM
%  CONTINUATION       : *NEW-MODE
%  UTILITY-INTERFACE  : *NEW-MODE
%  PROCEDURE-DIALOGUE : *NO
%  MENU-LOGGING       : *NO
%  MODE               : *EXECUTION
%     CHECK-PRIVILEGES   : *YES
%  DEFAULT-PROGRAM-NAME : *NONE
%  FUNCTION-KEYS      : *STYLE-GUIDE-MODE
%  INPUT-HISTORY      : *ON
%     NUMBER-OF-INPUTS   : 20
%     PASSWORD-PROTECTION: *YES
```

1. A task is initiated under the user ID SDFUSR. The user syntax file SDF.USER.SYNTAX cataloged under this user ID is automatically activated when the LOGON command is processed. In this syntax file the START-SDF-A command is disabled.
2. The activated syntax files are listed. The user syntax file SDF.USER. SYNTAX is activated.

```
/start-sdf-a ─────────────────────────────────────────────────────────── (3)
%  CMD0086 OPERATION NAME 'START-SDF-A' REMOVED BY USER
/mod-sdf-opt synt-file=*rem(*std) ────────────────────────────────────── (4)
/start-sdf-a ─────────────────────────────────────────────────────────── (5)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file sdf.user.syntax ───────────────────────────────────── (6)
//show *com(start-sdf-a) ──────────────────────────────────────────────── (7)
%  CMD0051 INVALID OPERAND 'OBJECT=*COM:NAME'
%  SDA0083 NAME 'START-SDF-A' UNKNOWN
%  SDA0407 CORRECTION REJECTED OR NOT POSSIBLE. STATEMENT IGNORED
//restore *com(start-sdf-a) ───────────────────────────────────────────── (8)
//end
/mod-sdf-opt synt-file=*add(*std) ────────────────────────────────────── (9)
/start-sdf-a ─────────────────────────────────────────────────────────── (10)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//end
/exit-job
  .
  .
```

3. SDF does not accept the START-SDF-A command.
4. The user syntax file SDF.USER.SYNTAX is deactivated.
5. SDF accepts the START-SDF-A command, since the user syntax file SDF.USER.SYNTAX is now deactivated. SDF-A is loaded and started.
6. The user syntax file SDF.USER.SYNTAX is opened. By default, the activated system syntax file and the activated group syntax file are assigned as reference files.
7. Since the START-SDF-A command is disabled, SDF-A rejects the attempt to issue it.
8. The lock on the START-SDF-S command is lifted, provided that the assigned reference syntax files contain the definition.
9. The previously edited user syntax file SDF.USER.SYNTAX is activated.
10. The START-SDF-A command is no longer disabled. The lock has been successfully lifted by step 8.

3.2 Limiting the range of functions

SDF-A can be used to limit the range of functions either throughout the BS2000 system or for a specific user ID. A system-global limitation must be defined in a system syntax file, a limitation for a specific user ID in a group syntax file.

When defining a limitation in a group syntax file, the relevant system syntax file must be assigned as a reference file (OPEN-SYNTAX-FILE ... SYSTEM-DESCRIPTIONS=) if the definition of the command or statement is not already present in the group syntax file.

Commands, statements, operands and operand values can be disabled either generally (using the REMOVE statement) or for specific operating modes (using the statement MODIFY-xxx ...,DIALOG-ALLOWED=...,DIALOG-PROC-ALLOWED=..,BATCH-ALLOWED=...,BATCH-PROC-ALLOWED=...). In addition, commands may be disabled as a function of the way they are called (CMD-ALLOWED=).

Limiting the functional scope of a command

If the functional scope of a command is to be effectively limited, care must be taken to ensure that the limitation defined cannot be circumvented with the help of other commands. In most cases a limitation can be circumvented with the help of the old ISP commands. These are defined such that their functional scope cannot be modified using SDF-A. They can only be disabled. If they can be called via the MCLP interface, a general lock can have unforeseeable consequences. A sensible precautionary measure would then be to disable these commands for interactive and batch mode only, but not for calls via MCLP (Macro Command Language Processor). However, such a partial disabling can be circumvented relatively easily, e.g. with the help of EDT. The appropriate action is determined by the particular case (see examples 4 and 6). Commands sent with the SDF-P command EXECUTE-CMD (e.g. to divert the command output to a variable) are not subject to the requirements for DIALOG-ALLOWED=..., DIALOG-PROC-ALLOWED=..., BATCH-ALLOWED=... and BATCH-PROC-ALLOWED=..., because they are called via the MCLP interface.

Disabling operands

If an operand is to be disabled, it is usually impossible to predict the consequences on the implementation if the definition of the operand is deleted. To be on the safe side it is advisable merely to mask out the operand at the user interface by means of the statement MODIFY-OPERAND..., PRESENCE=*INTERNAL-ONLY. A default value must then be defined, covered by an operand value defined for the operand.

It is also possible to define a limitation in the functional scope by defining a secret operand and subsequently concatenating it with a visible operand (see example 5, page 74).

Limiting operand values

Limitations as to the permissible length or the permissible digit value for operand values can be defined by means of the MODIFY-VALUE statement. Furthermore, in MODIFY-VALUE..., VALUE=(<c-string>(...),...),... you can specify a list of valid input values. All values not listed are then rejected as invalid. The statement MODIFY-VALUE... VALUE= <c-string>(...,OUTPUT=...,) can be used to direct SDF to convert a defined single value to another value before passing it to the implementation or to suppress transfer of the value. In some cases, the limitation can also be imposed by modifying the data type. In the case of operand values defined as fully or partially qualified file names, the specification of the catalog identification, the user ID, a generation number or the designation of a version number as well as the input of wildcards may be declared illegal. Wildcards can also be declared legal or illegal for the data types <alphanum-name>, <composed-name> and <name>.

Limiting the functional scope of a user program

A user program limitation defined in a system or group syntax file can be rendered ineffectual by means of a user syntax file. If the functional scope of a user program is to be limited *effectively*, care must be taken to ensure that the users concerned cannot define the full functional scope in a user syntax file.

Restricted loading and execution of programs

System administration can restrict the loading and execution of programs to certain user applications (such as EDT), by defining a START-EDITOR command in the UTILITIES or PROGRAMMING-SUPPORT domain of SDF (see example on page 64).

To prevent the commands START-PROGRAM, EXEC, LOAD-PROGRAM and LOAD from being executed directly, system or group syntax files are changed by assigning the attributes DIALOG-ALLOWED=*NO and BATCH-ALLOWED=*NO to these commands. The same attributes must be specified for the commands CALL-PROCEDURE, CALL and DO. This method prevents the direct execution of any program or procedure, but does permit procedure commands to be executed, in which case the procedures called may contain the illegal commands (START-PROGRAM,...).

3.3 Removing limitations

The actions involved in removing a limitation depend on:

– the type of syntax file in which the limitation is defined

– whether the limitation is to be removed on a global basis or only for particular users

– whether the limitation involves the functional scope of a command implemented via a system module, a command implemented via a procedure, or a user program

– whether the limitation is to be removed completely or only partially

– whether the limitation is implemented via disabling, deletion or modification of a command or statement.

The procedure often corresponds to that used to define the limitation.

– If the limitation was implemented by deleting the definition of a command, statement, operand or operand value, it may be removed by reinstating the deleted definition using COPY.

– If the limitation was implemented by modifying the definition of a command, statement, operand or operand value, the first step involves deleting the modified definition by means of REMOVE. Then, in a second step, the original definition can be reinstated using COPY.

– If a command or statement modification defined in a group or user syntax file is to be completely removed, the syntax file should be opened without assignment of a reference file and the modified definition removed by means of REMOVE. This approach presupposes that the original version of the definition is still present in the relevant system or group syntax file.

– If a command or statement lock defined in a group or user syntax file is to be lifted, this can be done with the aid of the RESTORE statement. When unlocking the definition of a command implemented via system modules for a user syntax file, one must bear in mind that the command definition must be contained in the reference syntax files assigned when the user syntax file is opened.

4 Definition and implementation of commands and statements by the user

4.1 Syntax rules and recommendations

This section lists rules and recommendations governing syntax. It does not provide information regarding specific details, e.g. as to the maximum length of an operand name. Such detailed information can be found in the descriptions of the ADD statements, which are used to enter the syntax of commands and statements in a syntax file.

In the following, the words “should” and “may” are used for recommendations; the word “must” denotes rules which must be complied with. The recommendations for commands also apply to statements.

– The name of a command, statement or operand, as well as an operand value of the type KEYWORD, may be made up of several subnames, connected together by hyphens. All subnames should be based on natural language. When two things are the same, the same subname should be used for them.

  During input, the subnames may be abbreviated as desired and/or omitted entirely from right to left, provided SDF can unambiguously identify the full name on the basis of the abbreviation.

– The name of a command should begin with a verb. The subname following the verb should indicate the object that is processed by the command. The ALIAS-NAME must be different from NAME or STANDARD-NAME. If the possibility of a command name colliding with the name of a new command supplied by Fujitsu Siemens Computers as part of a future version of the program is to be absolutely ruled out, the subname “X” can be used as the beginning of the command name.

– Each function should be dealt with by a separate command, taking care that the function is not too complex. There should be complementary commands for complementary functions.

- Each operand must have a name.
  During input, an operand may be specified either as a positional operand or as a keyword operand.
- Optional operands must have a default value.
- Operands that are relevant only when another operand has a certain specific value should be attached to that value in a structure. A structure consists of one or more operands, enclosed in parentheses. It is attached to the value of a higher-ranking operand.
- Operands logically belonging together should be placed together in a structure. It may make sense to construct a higher-ranking operand expressly for the purpose of introducing the structure.
- Structures should not be nested to more than five levels.
  =>>No more than five levels of nesting are allowed for structures.
- The name of an operand within a structure need be unique only within that structure. Nevertheless, it is expedient during input if the name is unambiguous for all commands and statements. The structure can then be implicitly selected by specifying the operand.
- An operand value must be defined as one of the following data types:

  <alphanumeric-name>
  <cat-id>
  <composed-name>
  <c-string>
  <date>
  <device>
  <fixed>
  <filename>
  <integer>
  KEYWORD
  <name>
  <partial-filename>
  <posix-pathname>
  <posix-filename>
  <product-version>
  <structured-name>
  <time>
  <vsn>
  <x-string>
  <x-text>

For some of these data types the possible operand value can be defined more precisely by means of additional specifications, e.g. by specifying a minimum and a maximum length (see ADD-VALUE). The data types <command-rest>, KEYWORD-NUMBER, <label> and <text> are reserved for Fujitsu Siemens Computers Software Development.

– With the exception of keywords, the input alternatives defined for an operand must be of different data types. These data types must be syntactically disjunct, unless overlapping of data types has been permitted by means of the statement ADD-OPERAND...VALUE-OVERLAPPING. Otherwise, for example, simultaneous definition of a value of the type NAME and of an alternative value of the type STRUCTURED-NAME for an operand is not possible (see section “Mutually exclusive data types” on page 623 in the appendix).

  i VALUE-OVERLAPPING=*YES should only be used by SDF-A experts, and even then only if the problem cannot be solved in any other way.

– A keyword must be prefixed by an asterisk when necessary to make it distinguishable from other input alternatives
  (see ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY)).

– It is possible to specify that a list of values can be entered for one operand. During input these values must be separated from one another by commas and enclosed in parentheses.

– The coexistence of operand values of data type KEYWORD and operand values of a data type with wildcards leads to special situations. A list of 6 possible cases and the results of the SDF syntax analysis for different inputs are given below:

  1. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
     ADD-VALUE TYPE=*KEYWORD(STAR=*OPTIONAL),VALUE='ALL'
     ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

  2. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
     ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
     ADD-VALUE TYPE=*KEYWORD(STAR=*OPTIONAL),VALUE='ALL'

  3. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
     ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='ALL'
     ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

  4. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
     ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
     ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='ALL'

  5. ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*YES
     ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)

6.
   ```
   ADD-OPERAND NAME=OP1,VALUE-OVERLAPPING=*NO
   ADD-VALUE TYPE=*KEYWORD(STAR=*MANDATORY),VALUE='V4.1'
   ADD-VALUE TYPE=*FILENAME(WILDCARD=*YES)
   ```

During syntax analysis, SDF V4.1 treats the inputs as the following data types:

Input of:	**Case number**					
	1	**2**	**3**	**4**	**5**	**6**
A	keyword	filename	filename	filename	filename	filename
*A	keyword	keyword	keyword	keyword	error	error
**A	filename	filename	filename	filename	filename	filename
*A/	filename	filename	filename	filename	filename	filename
*	filename	filename	filename	filename	filename	filename
*B	error	error	error	error	error	error
*V4.1	error	error	error	error	error	keyword
*V4.	filename	filename	filename	filename	filename	filename
**V4.1	filename	filename	filename	filename	filename	filename

4.2 Examples for defining and implementing commands

4.2.1 Example 1: Assembly command

A command to assemble an assembly language source program and store the generated object module in an EAM object module library is to be defined and then be implemented via a procedure. The necessary macros may be located in a user macro library. The command is to have the following format:

ASSEMBLE-SOURCE
SOURCE = <filename 1..54>
,**MACRO-LIBRARY** = ***NONE** / <filename 1..54>
,**TEST-SUPPORT** = ***NO** / ***YES**

The command is implemented via a procedure. The symbolic operands appear as positional operands in the operand list of the BEGIN-PROCEDURE command. The following procedure appears as the element ASSEMB in the program library $SDFUSR.PROC.LIB.

```
/BEGIN-PROCEDURE PARAMETERS=*YES(PROC-PARAM=(&SOURCE,&MACROLIB,&TEST))
/DELETE-SYSTEM-FILE FILE-NAME=*OMF
/ASSIGN-SYSDTA TO=*SYSCMD
/START-ASSEMBH
//     COMPILE SOURCE=&SOURCE,-
//             MACRO-LIB=&MACROLIB,-
//             TEST=&TEST
//     END
/SET-JOB-STEP
/ASSIGN-SYSDTA TO-FILE=*PRIMARY
/END-PROCEDURE
```

The command ASSEMBLE-SOURCE is defined in the user syntax file SDF.USER. SYNTAX. It is then tested.

```
/set-logon-parameters  sdfusr,... ───────────────────────────────────  (1)
/mod-sdf-options syntax-file=*remove(*std) ──────────────────────────  (2)
/start-sdf-a ────────────────────────────────────────────────────────  (3)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.user.syntax ─────────────────────────────────  (4)
%//set-glob cont=*new ───────────────────────────────────────────────  (5)
%//add-cmd assemble-source,help=e('Assembles a program'), -
%//domain=programming-support, -
%//implementor=*proc('*lib-elem(lib=$sdfusr.proc.lib,elem=assemb)') ──  (6)
%//add-oper source,res-oper-name=*pos(1) ────────────────────────────  (7)
%//add-value *filename ───────────────────────────────────────────────  (8)
%//add-oper macro-library,def='*none',res-oper-name=*pos(2) ─────────  (9)
%//add-value *keyw(*mand),value='*none' ────────────────────────────── (10)
```

1. A task is initiated under the user ID SDF.USER.
2. The user syntax file SDF.USER.SYNTAX automatically activated during LOGON processing is deactivated.
3. SDF-A is loaded and started.
4. The existing user syntax file SDF.USER.SYNTAX is opened.
5. The global information defines that the continuation character “-” for continuing lines may stand in any column between 2 and 72 when SYSCMD or SYSSTMT is input.
6. The header of the ASSEMBLE-SOURCE command is defined. The command contains an English help text and is assigned to the domain USER. It is implemented by means of the procedure located as the element ASSEMB in the program library $SDFUSR.PROC.LIB.
7. The first operand of the ASSEMBLE-SOURCE command is defined. Its name is SOURCE. It stands at the beginning of the string to be passed to the procedure.
8. It is defined that the value of the SOURCE operand must be of the type FILENAME.
9. The second operand of the ASSEMBLE-SOURCE command is defined. Its name is MACRO-LIBRARY. Its default value is *NONE. It comes second in the string to be passed to the procedure.
10. The keyword NONE is defined as a permissible value of the MACRO-LIBRARY operand. When entered, it must be prefixed by an asterisk.

```
%//add-value *filename ———————————————————————————————————————————————— (11)
%//add-oper test-support,def='no',res-oper-name=*pos(3) ———————————————— (12)
%//add-value *keyw,value='no' ————————————————————————————————————————— (13)
%//add-value *keyw,value='yes' ———————————————————————————————————————— (14)
%//close-cmd ——————————————————————————————————————————————————————————— (15)
%//show *com(assemb-source),siz=*max ——————————————————————————————————— (16)
ASSEMBLE-SOURCE
     Assembles a program
     SOURCE =
          filename_1..54
     MACRO-LIBRARY = *NONE
          *NONE or filename_1..54
     TEST-SUPPORT = *NO
          *NO or *YES
%//end ————————————————————————————————————————————————————————————————— (17)
```

11. It is defined that the value of the MACRO-LIBRARY operand may be of the data type FILENAME.
12. The third global operand of the ASSEMBLE-SOURCE command is defined. Its name is TEST-SUPPORT and it has the default value NO. It comes third in the string to be passed to the procedure.
13. It is defined that the keyword NO is a permissible value of the TEST-SUPPORT operand.
14. It is defined that the keyword YES is a permissible value of the TEST-SUPPORT operand.
15. The definition of the ASSEMBLE-SOURCE command is terminated.
16. The definition of the ASSEMBLE-SOURCE command created in the user syntax file SDF.USER.SYNTAX is output in its most detailed form. SDF-A has defined a minimum length of 1 and a maximum length of 54 for all file names.
17. SDF-A is terminated. The user syntax file SDF.USER.SYNTAX is stored implicitly.

```
/mod-sdf-opt synt-file=*add(*std),guid=*n ————————————————————————————— (18)
/assemb-source demo.prog1,macro-lib=demo.maclib.test-support=*yes ————— (19)
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2B00' OF '1998-04-24' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1990. ALL RIGHTS
RESERVED
%  ASS6010 V01.2B02 OF BS2000 ASSTRAN  READY
%  ASS6011 ASSEMBLY TIME: 302 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NOTE
%  ASS6006 LISTING GENERATOR TIME: 130 MSEC
%  ASS6012 END OF ASSTRAN
.
 .
```

18. The user syntax file SDF.USER.SYNTAX, in which the ASSEMBLE-SOURCE command is defined, is activated.
19. The ASSEMBLE-SOURCE command is entered. The user-specific macro library DEMO.MACLIB is specified. Once the command has been executed, the resultant object module has been stored in the EAM object module file.

4.2.2 Example 2: Command to output the contents of a file

A command to output the contents of a SAM or ISAM file on SYSOUT is to be defined and is to be implemented by means of a procedure. By default, the records in the file to be output are of variable length, but files with fixed-length records may also be output. When ISAM files are output, the ISAM key has a default length of eight bytes, but it may also be shorter. The command is to have the following format:

TYPE-FILE
NAME = <filename 1..54>
,ACCESS-METHOD = ***ISAM**(...) / ***SAM**
***ISAM**(...)
| **KEY-LENGTH** = **8** / <integer 1..8>
,RECORD-SIZE = ***VARIABLE** / <integer 1..256>

The command is implemented by means of a procedure. The symbolic operands appear as positional operands in the operand list of the BEGIN-PROCEDURE command. The utility routine EDT is used within the procedure. The following procedure appears in the file $SDFUSR.TYPE.FILE.

```
/      BEGIN-PROCEDURE N,PARAM=YES(PROC-PARAM=(&FILE,&ACCESS,&KEY,&RECORD),-
/      ESCAPE-CHARACTER='&')
/      MODIFY-JOB-SWITCHES ON=(4,5)
/      ASSIGN-SYSDTA TO-FILE=*SYSCMD
/      CREATE-JV JV=RECORD-SIZE
/      MODIFY-JV JV=RECORD-SIZE,VALUE='&RECORD'
/      LOAD-PROGRAM FROM-FILE=$EDT
/      SKIP-COMMAND TO-LABEL=&ACCESS   "ACCESS IST ISAM ODER SAM"
/.SAM SKIP-COMMANDS TO-LABEL=SAM1,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/      SET-FILE-LINK LINK-NAME=EDTSAM,FILE-NAME=&FILE,ACCES-METHOD=SAM,-
/      RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/.SAM1 RESUME-PROGRAM
@      READ '&FILE'
/      HOLD-PROGRAM
/      SKIP-COMMANDS TO-LABEL=COMMON,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/      REMOVE-FILE-LINK LINK-NAME=EDTSAM
/      SKIP-COMMANDS TO-LABEL=COMMON
/.ISAM CREATE-JV JV=ISAM-KEY
/      MODIFY-JV JV=ISAM-KEY,VALUE='&KEY'
/      SKIP-COMMANDS TO-LABEL=ISAM3,IF=JV(COND=(ISAM-KEY='8' AND -
/      RECORD-SIZE='VARIABLE'))
/      SKIP-COMMANDS TO-LABEL=ISAM1,IF=JV(COND=(ISAM-KEY='8'))
/      SKIP-COMMANDS TO-LABEL=ISAM2,IF=JV(COND=(RECORD-SIZE='VARIABLE'))
/      SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCESS-METHOD=-
/      ISAM(KEY-LEN=&KEY,KEY-POS=1),RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/      SKIP-COMMANDS TO-LABEL=ISAM3
/.ISAM1 SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCES-METHOD=-
/      ISAM(KEY-LEN=8,KEY-POS=1),RECORD-FORMAT=FIXED(REC-SIZE=&RECORD)
/      SKIP-COMMANDS TO-LABEL=ISAM3
/.ISAM2 SET-FILE-LINK LINK-NAME=EDTISAM,FILE-NAME=&FILE,ACCES-METHOD=-
/      ISAM(KEY-LEN=&KEY)
/.ISAM3 RESUME-PROGRAM
@      GET '&FILE'
/      HOLD-PROGRAM
/      SKIP-COMMANDS TO-LABEL=ISAM4,IF=JV(COND=(ISAM-KEY='8' AND -
/      RECORD-SIZE='VARIABLE'))
/      REMOVE-FILE-LINK LINK-NAME=EDTISAM
/.ISAM4 DELETE-JV JV=ISAM-KEY
/.COMMON RESUME-PROGRAM
@      PRINT %.-$N
@      HALT
/      SET-JOB-STEP
/      DELETE-JV JV=RECORD-SIZE
/      ASSIGN-SYSDTA TO-FILE=*PRIMARY
/      MODIFY-JOB-SWITCHES OFF=(4,5)
/      END-PROCEDURE
```

The TYPE-FILE command is defined in the user syntax file SDF.USER.SYNTAX. It is subsequently tested.

```
/set-logon-parameters sdfusr,... ──────────────────────────────────────── (1)
 .
 .
/mod-sdf-opt synt-file=*remove(*std) ──────────────────────────────────── (2)
/start-sdf-a ──────────────────────────────────────────────────────────── (3)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.user.syntax ───────────────────────────────────── (4)
%//set-globals cont=*new ──────────────────────────────────────────────── (5)
%//add-cmd type-file,help=e('Outputs the contents of an ISAM or SAM -
%//file to SYSOUT.'),domain=user,impl=*proc('$sdfusr.type.file') ────── (6)
%//add-oper name,help=e('Name of file to be output'),res-oper-nam=*pos(1) (7)
%//add-value *filename ────────────────────────────────────────────────── (8)
%//add-oper access-method,def='isam',res-oper-name=*pos(2) ───────────── (9)
```

1. A task is initiated under the user ID SDFUSR.
2. The user syntax file SDF.USER.SYNTAX automatically activated during LOGON processing is deactivated.
3. SDF-A is loaded and started.
4. The existing user syntax file SDF.USER.SYNTAX is opened.
5. The global information defines that the continuation character “-” for follow-on lines may be situated in any column between 2 and 72 when SYSCMD or SYSSTMT is input.
6. The header of the TYPE-FILE command is defined. It contains an English help text and is assigned to the domain USER. It is implemented by means of the procedure appearing in the file $SDFUSR.TYPE.FILE.
7. The first operand of the TYPE-FILE command is defined. Its name is NAME. It contains an English help text. It comes first in the string to be passed to the procedure.
8. It is defined that the value of the operand NAME must be of the data type FILENAME.
9. The second operand of the TYPE-FILE command is defined. Its name is ACCESS-METHOD. Its default value is ISAM. In the string to be passed to the procedure, it occupies the second position.

```
%//add-value *keyw,struct=*y,value='isam' ──────────────────────────── (10)
%//add-oper key-length,def='8',res-oper-name=*pos(3) ───────────────── (11)
%//add-value *integ(1,8) ───────────────────────────────────────────── (12)
%//close-struct ────────────────────────────────────────────────────── (13)
%//add-value *keyw,value='sam' ─────────────────────────────────────── (14)
%//add-oper record-size,def='variable',res-oper-name=*pos(4) ───────── (15)
%//add-value *keyw,value='variable' ────────────────────────────────── (16)
%//add-value *integ(1,256) ─────────────────────────────────────────── (17)
%//close-cmd ────────────────────────────────────────────────────────── (18)
```

10. It is defined that the keyword ISAM is a permissible value of the ACCESS-METHOD operand. ISAM introduces a structure.
11. The first operand of the structure ISAM is defined. Its name is KEY-LENGTH. Its default value is the integer 8. It appears third in the string to be passed to the procedure.
12. It is defined that the value of the KEY-LENGTH operand must be of the data type INTEGER. 1 is defined as its lower limit, 8 as its upper limit.
13. The structure ISAM just edited is closed.
14. It is defined that the keyword SAM is a permissible value of the ACCESS-METHOD operand.
15. The third operand of the TYPE-FILE command is defined. Its name is RECORD-SIZE. Its default value is VARIABLE. It occupies the fourth position in the string to be passed to the procedure.
16. It is defined that the keyword VARIABLE is a permissible value of the RECORD-SIZE operand.
17. It is defined that the value of the RECORD-SIZE operand may be of the data type INTEGER. 1 is defined as its lower limit, 256 as its upper limit.
18. The definition of the TYPE-FILE command is terminated.

```
%//show *com(type-f),siz=*max ——————————————————————————————————————————— (19)
TYPE-FILE
     Outputs the contents of an ISAM or SAM file to SYSOUT.
     NAME =
          filename_1..54
          Name of file to be output
     ACCESS-METHOD = *ISAM
          *ISAM() or *SAM
          STRUCTURE: *ISAM
               KEY-LENGTH = 8
                    integer_1..8
     RECORD-SIZE = *VARIABLE
          *VARIABLE or integer_1..256
%//end
/type-file demo.1 —————————————————————————————————————————————————————— (20)
%  CMD0186 OPERATION NAME 'TYPE-FILE' UNKNOWN
/mod-sdf-opt synt-file=*add(*std) ——————————————————————————————————————— (21)
/type-f demo.1 ————————————————————————————————————————————————————————— (22)
Contents of ISAM file DEMO.1
 .
 .
/type-f demo.2,rec-siz=52
Contents of ISAM file DEMO.2
 .
 .
/type-f demo.3,isam(6)
Contents of ISAM file DEMO.3
 .
 .
```

19. The definition of the TYPE-FILE command, generated in the user syntax file SDF.USER.SYNTAX, is displayed in its most detailed form. By default, SDF-A has defined a minimum length of 1 and a maximum length of 54 for FILENAME.
20. SDF rejects the TYPE-FILE command, since it is not defined in an activated syntax file.
21. The user syntax file SDF.USER.SYNTAX is activated.
22. Various ISAM files are displayed with the TYPE-FILE command:
    DEMO.1: the ISAM key is 8 bytes long, the record length is variable
    DEMO.2: the ISAM key is 8 bytes long, the records length is 52 bytes
    DEMO.3: the ISAM key is 6 bytes long, the record length is variable

```
/type-f demo.4,sam,rec-siz=60 ------------------------------------------ (23)
Contents of SAM file DEMO.4
 .
 .
/type-f? --------------------------------------------------------------- (24)
```

```
COMMAND  : TYPE-FILE


--------------------------------------------------------------------------------
NAME                  = ?
ACCESS-METHOD         = *ISAM(KEY-LENGTH=8)
RECORD-SIZE           = ?VARIABLE















--------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

23. The SAM file DEMO.4 is displayed with the TYPE-FILE command. This file has a fixed record length of 60 bytes.
24. Guided dialog is temporarily introduced for the input of the TYPE-FILE command.

    Further information is requested via the operands NAME and RECORD-SIZE.

```
COMMAND  : TYPE-FILE
OPERANDS : NAME=?,RECORD-SIZE=*VARIABLE

-------------------------------------------------------------------------------
NAME                   = demo.5
                         filename_1..54
                         Name of the file to be output
ACCESS-METHOD          = *ISAM(KEY-LENGTH=8)
RECORD-SIZE            = 55
                         *VARIABLE or integer_1..256









-------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

```
Contents of ISAM file DEMO.5
 .
 .
/exit-job
```

The name of the file to be output is specified here as demo.5, and the record size is given as a fixed length of 55 bytes. The access method *ISAM(KEY-LENGTH=8) is the default value and is therefore preset.

The command is then executed. After the command has been executed the task is terminated.

4.2.3 Notes

If a command implemented by means of a procedure is to be generally available:

– the procedure must be shareable and

– the user ID under which the procedure file is cataloged must be specified as part of the file name in the command definition (see ADD-CMD).

In the operand list that SDF passes to the procedure the operands may be defined as keyword operands or as positional operands. In the case of a keyword operand, a different name may be passed than the one that appears in the command syntax (see ADD-OPERAND ...,RESULT-OPERAND-NAME=).

Operand definitions	**Command input**	**Parameter list passed to the procedure**

//ADD-OPER OP1,...,RES-OPER-N=*POS(1)
//ADD-OPER OP2,...,RES-OPER-N=*POS(2)
//ADD-OPER OP3,...,RES-OPER-N=*SAME
//ADD-OPER OP4,...,RES-OPER-N=PARAM
	/CMD W,X,Y,Z
	/CMD W,OP4=Z,
	OP3=Y,OP2=X
		('W','X',OP3='Y',
		PARAM='Z')

Several operands for a command may be concatenated to form a single operand and passed to the procedure in this form (see ADD-OPERAND ...,RESULT-OPERAND-NAME=,CONCATENATION-POS=).

Operand definitions	**Command input**	**Parameter list passed to the procedure**

//ADD-OPER OP1,...,RES-OPER-N=*POS(2),
CONC-POS=2
//ADD-OPER OP2,...,RES-OPER-N=*POS(2)
CONC-POS=1
//ADD-OPER OP3,...,RES-OPER-N=*POS(1)
	/CMD X,Y,Z
	
	/CMD X,OP3=Z,
	OP2=Y
		('Z','YX')

//ADD-OPER OP1,...,RES-OPER-N=OP2,
CONC-POS=2
//ADD-OPER OP2,...,RES-OPER-N=*SAME
CONC-POS=1
//ADD-OPER OP3,...,RES-OPER-N=*POS(1)
	/CMD X,Y,Z
	
	/CMD X,OP3=Z,
	OP2=Y
		('Z',OP2='YX')

An operand for which there is only one permissible value may be so defined that, while it does get passed to the procedure, it does not appear in the command syntax (see ADD-OPERAND ...,PRESENCE=*INTERNAL-ONLY). If the operand constitutes the only operand in a structure, then the structure will also be invisible in the command syntax. This can be used, among other things, for passing branch destinations to the procedure, to which processing within the procedure may branch depending on input alternatives (see "Example 1: Assembly command" on page 91).

On the other hand, an operand which, for example, is needed only for structuring the command syntax, and not for the implementation, may be suppressed when operands are passed to the procedure (see ADD-OPERAND ..., PRESENCE=*EXTERNAL-ONLY). ADD-VALUE ...,VALUE=<c-string>(...,OUTPUT=...,...),... can be used to direct SDF to convert defined single values to other values before passing them on to the procedure or to suppress transfer of the values. For example, it can be specified that SDF is to pass on the value 'ISD' to the procedure instead of the value 'YES' defined for the input (see example 1).

ADD-VALUE ...,OUTPUT=NORMAL(STRING-LITERALS=...) can be used to specify whether SDF is to recode an operand value before passing it on to the procedure (<c-string> to <x-string> or vice versa).

ADD-VALUE ...,STRUCTURE=*YES(SIZE=...,...),... is used to determine whether a structure will be integrated into the operand form at the MINIMUM and MEDIUM level of guided dialog or whether SDF will display a separate subform for this structure.

Normally, the default values of the ADD statements can be used extensively when defining a command. However, these statements also provide a number of means for tailoring a command definition to the user's individual requirements. Further information on this can be found in the descriptions of the ADD statements.

4.3 Definition and implementation of statements

4.3.1 Example: Program for copying files

A program is to be written to copy ISAM and SAM files. Using the program, it should be possible to change the following:

– the access method
– the record length
– the ISAM key.

The program is to execute as a main program.

If the file to be processed is protected by a password, it should be possible to enter the password without it appearing on the screen. If the password is omitted, it is to be requested in an error dialog.

In addition to the standard SDF statements, the program has the following statement:

```
COPY-FILE
------------------------------------------------------------------
FROM-FILE = <filename 1..54>
,TO-FILE = <filename 1..54 without-gen-vers>(...)
   <filename 1..54 without-gen-vers>(...)
     |  ACCESS-METHOD = *SAME / *ISAM(...) / *SAM
     |     *ISAM(...)
     |       |  KEY-LENGTH = *STD / <integer 1..50>
     |  ,RECORD-SIZE = *SAME / *VARIABLE / <integer 1..2048>
,PASSWORD = *NONE / <c-string 1..4> / *SECRET-PROMPT
```

Defining the program in the user syntax file

The program KOP is defined in the user syntax file SDF.KOP.SYNTAX:

```
/set-logon-parameters sdfusr, ... ————————————————————————————————————  (1)
 .
 .
/start-sdf-a ——————————————————————————————————————————————————————————  (2)
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
%//open-syntax-file sdf.kop.syntax,,*create ———————————————————————————  (3)
%//add-program kop ————————————————————————————————————————————————————  (4)
%//add-stmt name=copy-file,prog=kop,intern-name=copyfi,stmt-version=1 —  (5)
%//add-oper from-file,res-oper-name=*pos(1) ———————————————————————————  (6)
%//add-val *filename ——————————————————————————————————————————————————  (7)
%//add-oper to-file,res-oper-name=*pos(2) —————————————————————————————  (8)
%//add-val *filename(gen=*n,vers=*n),structure=*y(siz=*small) —————————  (9)
%//add-oper access-method,default='same',res-oper-name=*pos(3) ———————— (10)
```

1. A task is initiated under the user ID SDFUSR.
2. SDF-A is loaded and started.
3. The user syntax file SDF.KOP.SYNTAX is created and opened.
4. The program KOP is defined. By default, SDF-A takes the first three letters (KOP) as its internal name.
5. The header of the COPY-FILE statement belonging to the program KOP is defined. Its internal name is COPYFI and it has the version number 1.
6. The first operand of the COPY-FILE statement is defined. Its name is FROM-FILE. It occupies the first position in the operand array of the transfer area.
7. It is defined that the value of the FROM-FILE operand must be of the data type FILENAME.
8. The second operand of the COPY-FILE statement is defined. Its name is TO-FILE. In the operand array of the transfer area it occupies the second position.
9. It is defined that the value of the TO-FILE operand must be of the data type FILENAME. Specification of a generation number or version number is not permitted. FILENAME introduces a structure.
10. The first operand in the structure FILENAME is defined. Its name is ACCESS-METHOD. Its default value is SAME. In the operand array of the transfer area it occupies the third position.

```
%//add-val *keyw,val='same' ──────────────────────────────────────────── (11)
%//add-val *keyw,val='isam',struct=*y ──────────────────────────────── (12)
%//add-oper key-length,default='std',res-oper-name=*pos(4) ─────────── (13)
%//add-val *keyw,val='std' ───────────────────────────────────────────── (14)
%//add-val *integer(1,50) ─────────────────────────────────────────────── (15)
%//close-structure ──────────────────────────────────────────────────────── (16)
%//add-val *keyw,val='sam' ───────────────────────────────────────────── (17)
%//add-oper record-size,default='same',res-oper-name=*pos(5) ───────── (18)
%//add-val *keyw,val='same' ──────────────────────────────────────────── (19)
%//add-val *keyw,val='variable' ──────────────────────────────────────── (20)
%//add-val *integer(1,2048) ──────────────────────────────────────────── (21)
%//close-struct ───────────────────────────────────────────────────────── (22)
```

11. It is defined that the keyword SAME is a permissible value for the ACCESS-METHOD operand.
12. It is defined that the keyword ISAM is a permissible value for the ACCESS-METHOD operand. ISAM introduces a structure.
13. The first operand in the ISAM structure is defined. Its name is KEY-LENGTH. Its default value is STD. In the operand array of the transfer area it occupies the fourth position.
14. It is defined that the keyword STD is a permissible value for the KEY-LENGTH operand.
15. It is defined that the value of the KEY-LENGTH operand may be of the data type INTEGER. 1 is defined as its lower limit, 50 as its upper limit.
16. The structure just edited (ISAM) is closed.
17. It is defined that the keyword SAM is a permissible value for the ACCESS-METHOD operand.
18. The second operand in the structure FILENAME is defined. Its name is RECORD-SIZE. Its default value is SAME. In the operand array of the transfer area it occupies the fifth position.
19. It is defined that the keyword SAME is a permissible value for the RECORD-SIZE operand.
20. It is defined that the keyword VARIABLE is a permissible value for the RECORD-SIZE operand.
21. It is defined that the value of the RECORD-SIZE operand may be of the data type INTEGER. 1 is defined as its lower limit, 2048 as its upper limit.
22. The structure just edited (FILENAME) is closed.

```
%//add-oper password,default='none',secret=*y,res-oper-name=*pos(6) ─── (23)
%//add-val *keyw,val='none' ─────────────────────────────────────────── (24)
%//add-val *c-string(1,4) ───────────────────────────────────────────── (25)
%//add-val *keyw,val='secret-prompt',out=*secret ────────────────────── (26)
%//close-cmd ────────────────────────────────────────────────────────── (27)
%//show object=*program(name=kop),size=*max ─────────────────────────── (28)
KOP
COPY-FILE
   FROM SDF.KOP.SYNTAX (USER)
     FROM-FILE =
          filename 1..54
     TO-FILE =
          filename 1..54_without-generation-version()
          STRUCTURE: filename
               ACCESS-METHOD = *SAME
                    *SAME or *ISAM() or *SAM
                    STRUCTURE: *ISAM
                         KEY-LENGTH = *STD
                              *STD or integer_1..50
               RECORD-SIZE = *SAME
                    *SAME or *VARIABLE or integer_1..2048
     PASSWORD =
          *NONE or c-string_1..4 or *SECRET-PROMPT -DEFAULT-: *NONE

%//end
 .
 .
%//exit-job
```

23. The third global operand of the COPY-FILE statement is defined. Its name is PASSWORD. Its default value is NONE. It is defined as a secret operand. In the operand array of the transfer area it occupies the eighth position.
24. It is defined that the keyword NONE is a permissible value for the PASSWORD operand.
25. It is defined that the value of the PASSWORD operand may be of the data type C-STRING. 1 is defined as its minimum length, 4 as its maximum length.
26. It is defined that the keyword SECRET-PROMPT is a permissible value for the PASSWORD operand. It is not passed to the implementation; instead, after it has been entered, SDF requests a value (not displayed) for PASSWORD.
27. The definition of the COPY-FILE statement is terminated.
28. The definition of the program KOP, with all of the statements that go with it, is displayed in its most detailed form.

After analyzing the COPY-FILE statement, SDF writes the following information into the transfer area. The entries enclosed in parentheses are written by SDF-A only when ACCESS-METHOD=*ISAM applies.

Byte	Length in bytes	Description	Value
0 to 7	8	Standard header	
8 to 11	4	Length of the transfer area	
12 to 19	8	Internal statement name	**C'COPYFI'**
20 to 23	4	Reserved	
24 to 26	3	Version of the statement	**C'001'**
27 to 35	9	Reserved	
36 to 37	2	Number of positions in the operand array	**6**
38 to 39	2	Reserved	
40	1	Additional information for FROM-FILE	**B'1... ....'**
41	1	Type description for FROM-FILE	**11**
42	1	Global syntax attributes for FROM-FILE (always 0, as wildcards are not permitted)	**B'.... ....'**
43	1	Syntax attributes for data type <filename>, if:	
		– a catalog ID is specified	**B'1... ....'**
		– a user ID is specified	**B'.1.. ....'**
		– a file generation is specified	**B'..1. ....'**
		– a version is specified	**B'...1 ....'**
		– the file is temporary	**B'.... 1...'**
44 to 47	4	Absolute address of the value of FROM-FILE	**aaaa**
48	1	Additional information for TO-FILE	**B'1... ....'**
49	1	Type description for TO-FILE	**11**
50	1	Global syntax attributes for TO-FILE (always 0, as wildcards are not permitted)	**B'.... ....'**
51	1	Syntax attributes for data type <filename>, if:	
		– a catalog ID is specified	**B'1... ....'**
		– a user ID is specified	**B'.1.. ....'**
		– the file is temporary	**B'.... 1...'**
52 to 55	4	Absolute address of the value of TO-FILE	**aaaa**
56	1	Additional information for ACCESS-METHOD	**B'1... ....'**
57	1	Type description for ACCESS-METHOD	**22**
58	1	Reserved	
59	1	Reserved	
60 to 63	4	Absolute address of the value of ACCESS-METHOD	**aaaa**

Continued ➠

Byte	Length in bytes	Description	Value
64	1	Additional information for KEY-LENGTH	**B'0... ....'** **or** **B'1... ....'**
65	1	Type description for KEY-LENGTH	**(2 or 22)**
66	1	Reserved	
67	1	Reserved	
68 to 71	4	Absolute address of the value of KEY-LENGTH	**(aaaa)**
72	1	Additional information for RECORD-SIZE	**B'1... ....'**
73	1	Type description for RECORD-SIZE	**2 or 22**
74	1	Reserved	
75	1	Reserved	
76 to 79	4	Absolute address of the value of RECORD-SIZE	**aaaa**
80	1	Additional information for PASSWORD	**B'1... ....'**
81	1	Type description for PASSWORD	**2 or 22**
82	1	Reserved	
83	1	Syntax attributes for data type <c-string>:	
		– if the password contains single quotes	**B'1... ....'**
84 to 87	4	Absolute address of the value of PASSWORD	**aaaa**
	2	Length specification	**ll**
	2	Reserved	
	≤ 54	Value of FROM-FILE	**xxx**
	2	Length specification	**ll**
	2	Reserved	
	≤ 54	Value of TO-FILE	**xxx**
	2	Length specification	**ll**
	2	Reserved	
	≤ 4	Value of ACCESS-METHOD	**xxx**
	(2)	(Length specification)	**(ll)**
	(2)	(Reserved)	
	(≤ 4)	(Value of KEY-LENGTH)	**(xxx)**
	2	Length specification	**ll**
	2	Reserved	
	≤ 8	Value of RECORD-SIZE	**xxx**
	2	Length specification	**ll**
	2	Reserved	
	≤ 4	Value of PASSWORD	**xxx**

The COPY-FILE statement occupies a maximum of 240 bytes in the transfer area. Consequently, the transfer area must be at least 240 bytes long.

Generating the program

The KOP program is shown below:

```
KOP      START
         TITLE 'Example of program using SDF macros'
*----------------------------------------------------------------------*
* The program reads and corrects the statement COPY-FILE with SDF,
* then executes it.
* The field identifiers used by this program are to be found in the
* SDF macros CMDTA for the transfer area, CMDMEM for the status.
*----------------------------------------------------------------------*
R0       EQU      0
R1       EQU      1
R2       EQU      2
R3       EQU      3
R4       EQU      4
R5       EQU      5
R6       EQU      6
R7       EQU      7
R8       EQU      8
R9       EQU      9
R10      EQU      10
R11      EQU      11
R12      EQU      12
R13      EQU      13
R14      EQU      14
R15      EQU      15
*----------------------------------------------------------------------*
FRFILE#  EQU      1                   position of operand FROM-FILE
TOFILE#  EQU      2                                       TO-FILE
ACCESS#  EQU      3                                       ACCESS-METH
KEYLEN#  EQU      4                                       KEY-LENGTH
RECSIZ#  EQU      5                                       RECORD-SIZE
PASSWR#  EQU      6                                       PASSWORD
         CMDANALY ,                   SDF return codes
TAD      CMDTA    MF=D                SDF transfer area
RSTD     CMDRST   MF=D                Read statements
CSTD     CMDCST   MF=D                Correct statement
*----------------------------------------------------------------------*
*        Register usage
*        R2       SDF output area for CMDSTA, CMDRST, CMDCST.
*        R3       current character in filename analysis
*        R3       current field in SDF output area
*
*        R5       help register
*        R10,R11  base registers
*        R15      return code
```

```
*----------------------------------------------------------------------*
KOP      CSECT    ,
BEGIN    BALR     R10,R0
         USING    *,R10,R11
         B        FOO
SLICE#   DC       F'4096'
FOO      LR       R11,R10
         A        R11,SLICE#
*----------------------------------------------------------------------*
*        Check if user syntax file is activated.
*----------------------------------------------------------------------*
         CMDMEM   D,P=XMD               Layout for CMDSTA output
*
KOP      CSECT    ,
*
         CMDSTA   OUTAREA=STA#OUT       Get SDF options
*
         USING    XMDMEM,R2
         LA       R2,STA#OUT
         LA       R3,XMDUSER
         CLI      0(R3),':'                Check catid
         BNE      NOCATID
CATIDL   LA       R3,1(R3)
         CLI      0(R3),':'
         BNE      CATIDL
         LA       R3,1(R3)
NOCATID  CLI      0(R3),'$'                Check userid
         BNE      NOUSERID
USERIDL  LA       R3,1(R3)
         CLI      0(R3),'.'
         BNE      USERIDL
         LA       R3,1(R3)
NOUSERID CLC      0(USF#L,R3),USF#NAME     Check user syntax file
         BNE      FC#ERROR
         DROP     R2
*----------------------------------------------------------------------*
*        Read SDF statement.
*----------------------------------------------------------------------*
READSTMT DS       0Y
         MVI      OWNFLAGS,0
         USING    RSTD,1            from CMDRST
         USING    TAD,R2            from CMDTA
         LA       R1,RSTPL
         LA       R2,OUTPUT
*
         CMDRST   MF=E,PARAM=RSTPL
*
         CLI      CMDRMR1,CMDRSUCCESSFUL  no error, continue...
         BE       F01
         CLI      CMDRMR1,CMDREOF         EOF reached
         BE       FC#EOF
         CLI      CMDRMR1,CMDREND_STMT    //END was read
         BE       FC#END
         B        FC#ERROR              otherwise error...
*----------------------------------------------------------------------*
*        Evaluate structured-description
*----------------------------------------------------------------------*
F01      DS       0H
         CLC      CMDINTN,COPYFILE      is this //COPY-FILE ?
         BNE      FC#ERROR              it can only be //COPY-FILE !
*----------------------------------------------------------------------*
*        operand FROM-FILE
*----------------------------------------------------------------------*
         LA       R3,CMDMAIN+(FRFILE#-1)*CMDHEAL   FROM-FILE
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(from-file value)
         MVI      FRFILE,' '
         MVC      FRFILE+1(L'FRFILE-1),FRFILE
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,R0
         EX       R5,EX#MVC1            mvc with l=R5
         B        F02                   skip ex#mvc1...
EX#MVC1  MVC      FRFILE(1),CMDAVAL
*----------------------------------------------------------------------*
*        operand PASSWORD
*----------------------------------------------------------------------*
F02      DS       0H
         LA       R3,CMDMAIN+(PASSWR#-1)*CMDHEAL   PASSWORD
         USING    CMDODES,R3
         CLI      CMDOTYP,CMDCSTR       C-string input ?
         BNE      NO#PASS               no: no password input
*        copy input password into own field
         MVI      PASSWR,' '
         MVC      PASSWR+1(L'PASSWR-1),PASSWR
         USING    CMDHEAD,R3            operand description
         L        R3,CMDOPTR            A(password value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,R0
         EX       R5,EX#MVC2            mvc with l=R5
         B        FC#OPEN               skip ex#mvc2...
EX#MVC2  MVC      PASSWR(1),CMDAVAL
NO#PASS  DS       0H
         OI       OWNFLAGS,NOPASS
FC#OPEN  DS       0H
```

```
         CLC      PASSWR,QUESTION       is password = '????'
         BE       PASS#ER               test corstmt part.
*        open of file FROM-FILE
*        ...
*        if error at open of FROM-FILE:
*        B        PASS#ER
*---------------------------------------------------------------*
*        operand TO-FILE
*---------------------------------------------------------------*
         LA       R3,CMDMAIN+(TOFILE#-1)*CMDHEAL   TO-FILE
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC       operand input ?
         BZ       FC#ERROR              no: error
         MVI      TOFILE,' '
         MVC      TOFILE+1(L'TOFILE-1),TOFILE
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(to-file value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         BCTR     R5,R0
         EX       R5,EX#MVC3            mvc with l=R5
         B        F03                   skip ex#mvc3...
EX#MVC3  MVC      TOFILE(1),CMDAVAL
*---------------------------------------------------------------*
*        operand ACCESS-METHOD
*---------------------------------------------------------------*
F03      DS       0H
         LA       R3,CMDMAIN+(ACCESS#-1)*CMDHEAL   ACCESS-METHOD
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       FC#ERROR              no: error
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(access-method value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         CH       R5,THREE              sam?
         BE       I#SAM
         CLI      CMDAVAL,'I'           isam?
         BE       I#SAM
*        access-method = same
         OI       OWNFLAGS,ACCSAME
         MVI      ACCESS,'X'
         B        F04                   skip sam/isam
*        access-method = sam / isam
I#SAM    DS       0H
         MVC      ACCESS,CMDAVAL        S:sam / I:isam
*---------------------------------------------------------------*
*        operand KEY-LENGTH
```

```
*----------------------------------------------------------------------*
F04      DS       0H
         LA       R3,CMDMAIN+(KEYLEN#-1)*CMDHEAL   KEY-LENGTH
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       F05                   no: not isam, next operand.
         CLI      CMDOTYP,CMDINT
         BE       KEYINT
*        key-length = std : keylen := 8
         MVI      KEYLEN,8
         B        F05
*        key-length = <integer_1..50>
KEYINT   DS       0H
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(key-length value)
         USING    CMDOVAL,R3            value field
         MVC      KEYLEN,CMDAVAL+3
         B        F05
*----------------------------------------------------------------------*
*        operand RECORD-SIZE
*----------------------------------------------------------------------*
F05      DS       0H
         LA       R3,CMDMAIN+(RECSIZ#-1)*CMDHEAL   RECORD-SIZE
         USING    CMDODES,R3
         TM       CMDGSTA,CMDOCC        operand input ?
         BZ       FC#ERROR              no: error
         CLI      CMDOTYP,CMDINT
         BE       RECINT
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(record-size value)
         USING    CMDOVAL,R3            value field
         LH       R5,CMDLVAL            value length
         CH       R5,FOUR
         BNE      F051
*        record-size = same
         OI       OWNFLAGS,RECSAME
         B        F06
F051     DS       0H
*        record-size = variable
         OI       OWNFLAGS,RECVAR
         B        F06
*        record-size = <integer_1..2048>
RECINT   DS       0H
         USING    CMDHEAD,R3
         L        R3,CMDOPTR            A(record-size value)
         USING    CMDOVAL,R3            value field
         MVC      RECSIZ,CMDAVAL+2
F06      DS       0H
```

```
*----------------------------------------------------------------------*
*   ...  Copy file ...
*        Output of copied values for test purpose (even password!)
*----------------------------------------------------------------------*
         WROUT    MESS1,FC#END
         WROUT    MESS2,FC#END
         TM       OWNFLAGS,NOPASS
         BZ       WMESS3
         WROUT    MESS11,FC#END
         B        WMESS4
WMESS3   DS       0H
         WROUT    MESS3,FC#END
WMESS4   DS       0H
         TM       OWNFLAGS,ACCSAME
         BZ       WMESS5
         WROUT    MESS4,FC#END          access given
         B        WMESS6
WMESS5   WROUT    MESS5,FC#END          access default
WMESS6   DS       0H
         TM       OWNFLAGS,KEYSTD
         BNZ      WMESS8
         UNPK     BUFF5(5),KEYHW(3)
         TR       BUFF5(4),CONVCHAR-XF0
         MVC      KEYCHAR,BUFF5
         WROUT    MESS6,FC#END
         B        WMESS7
WMESS8   WROUT    MESS8,FC#END
WMESS7   DS       0H
         TM       OWNFLAGS,RECSAME
         BNZ      WMESS9
         TM       OWNFLAGS,RECVAR
         BNZ      WMESS10
         UNPK     BUFF5(5),RECSIZ(3)
         TR       BUFF5(4),CONVCHAR-XF0
         MVC      RECCHAR,BUFF5
         WROUT    MESS7,FC#END
         B        REPEAT
WMESS9   WROUT    MESS9,FC#END
         B        REPEAT
WMESS10  WROUT    MESS10,FC#END
*
*        R E P E A T   READSTMT
*
REPEAT   DS       0H
         B        READSTMT
*----------------------------------------------------------------------*
*        Password handling routine
*----------------------------------------------------------------------*
```

```
PASS#ER  DS       0H
         LA       R3,CMDMAIN+(PASSWR#-1)*CMDHEAL   PASSWORD
         USING    CMDODES,R3
         OI       CMDGSTA,CMDERR        set operand erroneous
         LA       R1,CSTPL
         USING    CSTD,R1
         CMDCST   MF=M,MESSAGE=A(MESSAGE)
CORRSTMT CMDCST   MF=E,PARAM=CSTPL
         CLI      CMDCMR1,CMDCSUCCESSFUL  corstmt successful?
         BNE      READSTMT                no: new read...
         B        F01                     repeat analysis...
*
*        Game over.
*
         B        FC#END
*----------------------------------------------------------------------*
FC#ERROR DS       0H                    error handling routine
*        ...
FC#EOF   DS       0H                    EOF   handling routine
*        ...
*----------------------------------------------------------------------*
FC#END   TERM
*----------------------------------------------------------------------*
*        Parameter lists
*----------------------------------------------------------------------*
RSTPL    CMDRST   MF=L,PROGRAM='KOP',OUTPUT=A(OUTPUT)
CSTPL    CMDCST   MF=L,INOUT=A(OUTPUT),MESSAGE=A(0)
*                                      given at execution
*----------------------------------------------------------------------*
*        Used constants, variables and buffers
*----------------------------------------------------------------------*
RC       DS       X                     return code byte
USF#NAME DC       'SDF.KOP.SYNTAX'      syntax file name
USF#L    EQU      *-USF#NAME            user syntax file length
COPYFILE DC       'COPYFI  '            //COPY-FILE internal name
QUESTION DC       '????'
THREE    DC       H'3'
FOUR     DC       H'4'
EIGHT    DC       H'8'
CONVCHAR DC       C'0123456789ABCDEF'
XF0      EQU      X'F0'
*        Message for CORSTMT
MESSAGE  DS       0F
         DC       Y(MSGEND-MESSAGE)
         DS       XL2
         DC       C'PLEASE ENTER PASSWORD FOR INPUT FILE'
MSGEND   EQU      *
*----------------------------------------------------------------------*
```

```
*        Output fields for statement operands
*-----------------------------------------------------------------------*
MESS1    DS       0F
         DC       Y(END1-MESS1)
         DS       CL2
         DC       X'40'
         DC       C'FROM-FILE = '
FRFILE   DS       CL52' '                 own from-file
END1     EQU      *
MESS2    DS       0F
         DC       Y(END2-MESS2)
         DS       CL2
         DC       X'40'
         DC       C'TO-FILE   = '
TOFILE   DS       CL52' '                 own to-file
END2     EQU      *
MESS3    DS       0F
         DC       Y(END3-MESS3)
         DS       CL2
         DC       X'40'
         DC       C'PASSWORD  = '
PASSWR   DS       CL4                     own password
END3     EQU      *
MESS4    DS       0F
         DC       Y(END4-MESS4)
         DS       CL2
         DC       X'40'
         DC       C'ACCESS-METHOD IS SAME'
END4     EQU      *
MESS5    DS       0F
         DC       Y(END5-MESS5)
         DS       CL2
         DC       X'40'
         DC       C'ACCESS-METHOD = '
ACCESS   DS       X                       own access-method: Sam | Isam
END5     EQU      *
MESS6    DS       0F
         DC       Y(END6-MESS6)
         DS       CL2
         DC       X'40'
         DC       C'KEY-LENGTH = '
KEYCHAR  DS       CL4' '                  key-length converted into hexa
END6     EQU      *
MESS7    DS       0F
         DC       Y(END7-MESS7)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE   = '
```

```
RECCHAR  DS       CL4' '                 rec-size   converted into hexa
END7     EQU      *
MESS8    DS       0F
         DC       Y(END8-MESS8)
         DS       CL2
         DC       X'40'
         DC       C'KEY-LENGTH IS STD'
END8     EQU      *
MESS9    DS       0F
         DC       Y(END9-MESS9)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE IS SAME'
END9     EQU      *
MESS10   DS       0F
         DC       Y(END10-MESS10)
         DS       CL2
         DC       X'40'
         DC       C'REC-SIZE IS VARIABLE'
END10    EQU      *
MESS11   DS       0F
         DC       Y(END11-MESS11)
         DS       CL2
         DC       X'40'
         DC       C'PASSWORD IS NONE'
END11    EQU      *
KEYHW    DC       H'0'
         ORG      KEYHW
FIL1     DS       X
KEYLEN   DS       AL1                    own key-length
BUFF5    DS       CL5                    buffer for unpack
RECSIZ   DS       H                      own record-size
OWNFLAGS DS       X                      own flags
ACCSAME  EQU      X'80'                  access-method=same
KEYSTD   EQU      X'40'                  key-length=std
RECSAME  EQU      X'20'                  record-size=same
RECVAR   EQU      X'10'                             =variable
NOPASS   EQU      X'08'                  no password.
*-----------------------------------------------------------------------*
STA#OUT  DS       0F                     CMDSTA output
         ORG      *+XMDMEML              length from CMDMEM
OUTPUT   CMDTA    MF=L,MAXLEN=400        CMDRST output
*-----------------------------------------------------------------------*
         END      KOP
```

Compiling, linking and testing the program

The source program KOP shown above is located in the file KOP.SRC. In the following it is compiled, linked and tested.

```
/set-logon-parameters sdfusr,... ──────────────────────────────── (1)
/start-assembh
%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.2B00' OF '1998-04-24' LOADED
%  BLS0552 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 1990. ALL RIGHTS
RESERVED
%  ASS6010 V01.2B02 OF BS2000 ASSTRAN  READY
%//compile source=*lib-elem(lib=kop.lib,elem=kop.src),-          ┐
%//mod-lib=kop.lib(el=kop),listing=*parameters-                  │
%//(source-print=*with-object-code(print-statements=*accept),- ── ├ (2)
%//macro-print=*std,min-mes-w=*note,cross-ref=*std,layout=*std,- │
%//output=*syslst),macro-lib=($.syslib.sdf.041,$loader.v140.syslib) ┘
%  ASS6011 ASSEMBLY TIME: 4223 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 3166 MSEC
%//end
%  ASS6012 END OF ASSTRAN
/start-binder
%  BND0500 BINDER VERSION 'V02.1A30' STARTED
%//start-llm-creation internal-name=kop ──────────────────────── (3)
%//include-module *lib(lib=kop.lib,elem=kop,type=r) ──────────── (4)
%//resolve-by-autolink $.syslib.sdf.045
%//resolve-by-autolink $loader.sysoml.bs2000-ga.14.0a
```

1. A task is initiated under the user ID SDFUSR.
2. The program must be assembled with ASSEMBH as it contains the new macros CMDTA, CMDCST and CMDRST. The source program with the program segments shown above is located under the name KOP.SRC as a type S member in the library KOP.LIB. The SDF macros are in the macro library $SYSLIB.SDF.045. The generated object module is to be stored in the library KOP.LIB under the member name KOP.
3. The linkage editor BINDER is instructed to create a link and load module (LLM) with the internal name KOP.
4. The linkage editor is instructed to read the object module KOP generated in step 2 and include it in the LLM.

```
%//save-llm lib=kop.lib,elem=kop ──────────────────────────────────────── (5)
%  BND1501 LLM FORMAT: '1'
%//end
%  BND1101 BINDER NORMALLY TERMINATED. SEVERITY CLASS: 'OK'
/mod-sdf-options synt-file=*add(sdf.kop.syntax) ───────────────────────── (6)
/start-exe *lib-elem(lib=kop.lib,elem=kop) ────────────────────────────── (7)
%  BLS0524 LLM 'KOP', VERSION ' ' OF '2001-10-10 14:00:22' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2001. ALL RIGHTS
RESERVED
%//mod-sdf-opt guid=*n ────────────────────────────────────────────────── (8)
%STMT:cop-f testdat.sv,pass=*secr ─────────────────────────────────────── (9)
%ENTER SECRET OPERAND (PASSWORD):......
%  CMD0051 INVALID OPERAND 'TO-FILE'
%  CMD0099 MANDATORY OPERAND INVALID OR MISSING
%ENTER OPERANDS:
to-f=testdat.cop.1(isam,50)
%STMT:cop-f testdat.if,testdat.cop.2(acc-m=?) ─────────────────────────── (10)
% CMD0090 EXPLANATION OF THE OPERAND ' TO-FILE=TESTDAT.COP.2:ACCESS-METHOD ':
%SAME or ISAM() or SAM -DEFAULT-: SAME
%ENTER OPERANDS:
%TESTDAT.SV,TESTDAT.COP.2(ACC-M=?)
acc-m=sam
%  CMD0051 INVALID OPERAND 'PASSWORD'
% PLEASE ENTER PASSWORD FOR INPUT FILE
%ENTER SECRET OPERAND (PASSWORD):......
```

5. The executable LLM is stored under the name KOP (as a member of type L) in the library KOP.LIB.
6. The user syntax file SDF.KOP.SYNTAX, in which the statements for the program KOP are defined, is activated.
7. The program KOP is started.The command used here, START-EXECUTABLE-PROGRAM, is available in BLSSERV V2.3 and higher (the START-PROGRAM command with RUN-MODE=*ADVANCED is to be used if necessary).
8. User guidance is set to NO.
9. Case 1: The COPY-FILE statement is entered without the mandatory operand TO-FILE. Input of the password is to be blanked. SDF requests input of the password and the missing operand.
10. Case 2: The COPY-FILE statement is entered without the required password. Further information on the ACCESS-METHOD operand is requested. SDF provides the information and requests entry of the password.

```
%STMT:cop-f testdat.iv,testdat.cop.3(?) ———————————————————————————— (11)
```

```
PROGRAM  : KOP                    STATEMENT  : COPY-FILE
OPERANDS : FROM-FILE=TESTDAT.IV,TO-FILE=TESTDAT.COP.3

--------------------------------------------------------------------------------
FROM-FILE           = TESTDAT.IV
TO-FILE             = TESTDAT.COP.3(ACCESS-METHOD=SAME,RECORD-SIZE=60)
PASSWORD            =














--------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

```
PROGRAM  : KOP                    STATEMENT  : COPY-FILE
OPERANDS : FROM-FILE=TESTDAT.IV,TO-FILE=TESTDAT.COP.3(RECORD-SIZE=60)

--------------------------------------------------------------------------------
FROM-FILE           = TESTDAT.IV
TO-FILE             = TESTDAT.COP.3(ACCESS-METHOD=SAME,RECORD-SIZE=60)
PASSWORD____________= ....














--------------------------------------------------------------------------------
NEXT = *CONTINUE
KEYS : F1=?   F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL

ERROR:  PLEASE ENTER PASSWORD FOR INPUT FILE
```

11. Case 3: The COPY-FILE statement is entered without the required password. Further information on the structure that introduces the file name of the output file is requested via the question mark. This causes a switch to temporarily guided dialog. SDF displays the desired information in the operand form and then requests the required password.

```
%STMT:?  ------------------------------------------------------------------ (12)

PROGRAM  : KOP
--------------------------------------------------------------------------------
AVAILABLE STATEMENTS:

  1  COPY-FILE                                8  RESET-INPUT-DEFAULTS
  2  END                           (!)        9  RESTORE-SDF-INPUT
  3  EXECUTE-SYSTEM-CMD                      10  SHOW-INPUT-DEFAULTS
  4  HELP-MSG-INFORMATION                    11  SHOW-INPUT-HISTORY
  5  HOLD-PROGRAM                  (!)       12  SHOW-SDF-OPTIONS
  6  MODIFY-SDF-OPTIONS                      13  SHOW-STMT
  7  REMARK                                  14  WRITE-TEXT







--------------------------------------------------------------------------------
NEXT =
KEYS : F1=?  F3=*EXIT  F5=*REFRESH  F6=*EXIT-ALL  F9=REST-SDF-IN  F12=*CANCEL

** NORMAL PROGRAM END **
%CMD:exit-job
```

12. The question mark indicates the switch to temporarily guided dialog. SDF displays a menu listing the available statements. These include the standard statements. The STEP statement is not available in interactive mode. Entering the digit 2 selects the END statement and thus terminates the program.

4.3.2 Notes on the definition of statements

Standard statements

The standard statements (see page 127ff) are not specific to SDF-A. They are offered system-globally for user programs that read their statements via SDF. These statements are not to be implemented in the user program. Information on an END or STEP statement that has been read in is passed to the user program via register 15 (old interface) or the return code in the standard header (new interface).

Operand position

In the transfer area into which SDF writes the analyzed statement, operands can be identified only on the basis of their position in the operand array (see section “Format of the standardized transfer area” on page 365ff). This position can be defined when defining the operands (see ADD-OPERAND..., RESULT-OPERAND-NAME=*POS(...)).

Structure

When defining a value that introduces a structure (see ADD-VALUE ..., STRUCTURE=*YES(...,FORM...)) and when defining the operands within the structure (see ADD-OPERAND ...,RESULT-OPERAND-LEVEL=), the user specifies whether the structure is preserved in the transfer area or is linearized. If the default values of the statements ADD-VALUE and ADD-OPERAND are used, the structure is linearized. ADD-VALUE ...,STRUCTURE=*YES(SIZE=...,...),... is used to determine whether a structure will be integrated in the higher-ranking operand form at the MINIMUM and MEDIUM level of guided dialog or whether SDF will display a separate subform for this structure. By default, the structure is so defined that SDF integrates it in the higher-ranking form.

Passing an operand

An operand needed, for example, only for structuring the statement syntax and not for the implementation may be omitted when operands are passed to the program (see ADD-OPERAND ...,PRESENCE=*EXTERNAL-ONLY).

On the other hand, an operand for which there is only one single permissible value may be so defined that, while it does get passed to the program, it does not appear in the statement syntax (see ADD-OPERAND ..., PRESENCE=*INTERNAL-ONLY). If the operand constitutes the only operand in a structure, then the structure will also be invisible in the statement syntax.

Defining an operand value

When an operand value is defined, the user must specify how SDF is to write the value into the transfer area. If the default values of the ADD-VALUE statement are used, SDF will write any operand value input into the transfer area unchanged.

ADD-VALUE...,VALUE=<c-string>(...,OUTPUT=...,...) can be used to direct SDF to convert defined single values to different values before passing them on to the program or to suppress transfer of the values. For example, it can be specified that SDF passes on the value 'ISD' to the program instead of the value 'YES' defined for the input (see example 1 on page 91).

The default value of an operand defined with ADD-OPERAND ..., OVERWRITE-POSSIBLE=*YES can be replaced by the user program with another value (internal default) such that, in guided dialog, the internal default value is visible in the operand form (see the DEFAULT operand for the CMDCST, CMDRST and CMDTST macros).

An entered operand value can be replaced by the user program with another value if the value to be replaced has been defined using ADD-VALUE ..., VALUE=<c-string> (...,OVERWRITE-POSSIBLE=*YES),...) The following applies:

– The internal default value must be covered by an operand value defined with ADD-VALUE.

– Not more than one internal default value can be defined per operand. However, it is possible to assign the same internal default value to several input alternatives of an operand.

– If an internal default value is assigned to an input alternative, and if another input alternative has a structure attached to it which is also to have an internal default value, this requires that the structure be linearized in the transfer area (and thus also in the conversion description for internal default values).

– Defining an internal default value in alternative structures is possible only if the structures are linearized in the transfer area (one of the alternative structures may be non-linearized).

– If the operands to be defaulted belong to a structure whose introductory value is defined using LIST-ALLOWED=YES (see ADD-VALUE), the following may occur: The conversion description contains several list items to which a structure with operands to be defaulted is attached. At the same time the user also enters several list items to which a structure with operands to be defaulted is attached. SDF attempts first to allocate the structures entered by the user to those specified in the conversion description via the value that introduces the structure. If this value does not allow an unambiguous allocation because none of the values entered corresponds to any of those in the conversion description or because the user has entered the corresponding value several times, the allocation is made via the position in the list of the value introducing the structure.

Normally, the default values of the ADD statements can be used extensively when defining a statement. However, these statements also provide a number of means of tailoring a statement definition to the user's individual requirements. Further information on this may be found in the descriptions of the ADD statements.

4.3.3 Notes on the use of macros

SDF ensures that only statements without syntactical errors are read. This does not exclude the possibility that an operand value read in may be invalid in a specific situation. When the user program checks the validity of the operand values read in, the user can use the CMDCST macro to define a correction dialog (see example on page 116) to deal with errors.

If users want their programs to execute as a subprogram to which the calling program passes the statements, they can use the CMDCST macro to direct SDF to perform the statement analysis also.

By means of the STMT operand (cf. the CMDRST and CMDTST macros) the user can limit the number of permissible statements. For example, immediately after initiation of SDF-A, only the OPEN-SYNTAX-FILE, DEFINE-ENVIRONMENT and COMPARE-SYNTAX-FILE statements are permissible in addition to the standard statements. By means of the PREFER operand (see CMDRST) the user can specify that, in guided dialog, the form of the statement expected by the program is displayed instead of the statement menu. This does not, however, prevent the user from entering a statement different from that which is expected. For example, immediately after an EDIT statement SDF-A expects a corresponding MODIFY statement, but the user may enter any other SDF-A statement instead of the expected MODIFY statement.

With the aid of the SPIN operand (see the CMDRST macro) the user can specify that, in batch mode or in procedures, all statements up to the next STEP or END statement are skipped.

The CMDSTA macro causes the user program to be supplied with information on the activated syntax files and the definitions applicable for the command/statement input. When interpreting this information it should be borne in mind that the user ID is specified as part of the name of a syntax file only if the syntax file is cataloged under a different user ID (see example).

5 SDF-A statements

The statements to SDF-A are defined in syntax files furnished by Siemens Nixdorf. They are entered via the SDF command processor.

The standard statements are not specific to SDF-A. They include:

- END
- EXECUTE-SYSTEM-CMD
- HELP-MSG-INFORMATION
- HOLD-PROGRAM
- MODIFY-SDF-OPTIONS
- REMARK
- RESET-INPUT-DEFAULTS
- RESTORE-SDF-INPUT
- SHOW-INPUT-DEFAULTS
- SHOW-INPUT-HISTORY
- SHOW-SDF-OPTIONS
- SHOW-STMT
- STEP
- WRITE-TEXT

They are available on a system-global basis for user programs that read their statements via SDF. As soon as the user has defined his/her own user program, SDF assigns these statements to it. The EXECUTE-SYSTEM-CMD and HOLD-PROGRAM statements are available as of SDF V4.0 and BS2000/OSD-BC V2.0. The HELP-MSG-INFORMATION, RESET-INPUT-DEFAULTS and SHOW-INPUT-DEFAULTS statements can be used as of SDF V4.1.

Every user has access to the full range of SDF-A functions. Certain operands, however, are reserved for Fujitsu Siemens Computers Software Development and are masked out in the guided dialog and not described here.

Comments can be provided for statements to SDF-A; comments must be enclosed in quotation marks (").

5.1 Starting SDF-A

SDF-A is started by means of the following command:

START-SDF-A	Abbreviation: **SDF-A**
VERSION = ***STD** / <product-version>	
,MONJV = ***NONE** / <filename 1..54 without-gen-vers>	
,CPU-LIMIT = ***JOB-REST** / <integer 1..32767 *seconds*>	

VERSION =
Allows the user to select the desired SDF-A version if multiple versions of SDF-A were installed with IMON. If the version is specified within single quotes, it may be preceded by the letter C (C-STRING syntax).
If the product was not installed using IMON or if the specified version does not exist, VERSION=*STD applies.

VERSION = *STD
Calls the SDF-A version with the highest version number.

VERSION = <product-version>
Specifies the SDF-A version in the format mm.n[a[so]] (see also "product-version" on page 15).

MONJV =
Specifies a monitoring job variable to monitor the SDF-A run.

MONJV = *NONE
No monitoring job variable is used.

MONJV = <filename 1..54 without-gen-vers>
Name of the job variable to be used.

CPU-LIMIT =
Maximum CPU time in seconds which the program may consume for execution.

CPU-LIMIT = *JOB-REST
The remaining CPU time available is to be used for the job.

CPU-LIMIT = <integer 1..32767 *seconds***>**
Only as much time as is specified is to be used.

Interrupting and resuming SDF-A

If SDF-A is interrupted with K2 during processing of a statement, it can be resumed in one of two ways:

– if SDF-A is resumed with RESUME-PROGRAM, processing resumes with the statement which was interrupted;

– if SDF-A is resumed with INFORM=PROGRAM (up to BS2000/OSD-BC V2.0: SEND-MSG TO=*PROGRAM), the statement which was interrupted is aborted and SDF-A waits for input of the next statement.

5.2 Functional overview

SDF standard statements

A detailed description of the standard SDF statements can be found in the “Introductory Guide to the SDF Dialog Interface” [1].

END	Terminate the SDF-A session
EXECUTE-SYSTEM-CMD	Execute a system command during the SDF-A run
HELP-MSG-INFORMATION	Text of a system message issued on SYSOUT
HOLD-PROGRAM	Hold the SDF-A run
MODIFY-SDF-OPTIONS	Activate or deactivate a user syntax file and modify SDF options
REMARK	Include comments in the sequence of statements to document program execution
RESET-INPUT-DEFAULTS	Reset task-specific default values
RESTORE-SDF-INPUT	Re-display previously entered statements or commands on the screen
SHOW-INPUT-DEFAULTS	Display task-specific default values
SHOW-INPUT-HISTORY	Display contents of input buffer
SHOW-SDF-OPTIONS	Display task-specific information about active syntax files and the definitions for input and processing of statements
SHOW-STMT	Show the syntax of a statement
STEP	Define a restart point within a sequence of SDF-A statements
WRITE-TEXT	Output a specified text to SYSOUT or SYSLST

Creating, processing and comparing syntax files

COMPARE-SYNTAX-FILE	Compare objects in two syntax files with one another
COPY	Copy the contents of another syntax file into the open syntax file
DEFINE-ENVIRONMENT	Define syntax file format and version
EDIT	Position to an object in an open syntax file
OPEN-SYNTAX-FILE	Open a syntax file for processing with SDF-A
REMOVE	Delete objects from the open syntax file
RESTORE	Restore objects from the syntax file
SET-GLOBALS	Modify general settings for input and processing of commands

Defining objects

ADD-CMD	Define a command in the open syntax file
ADD-DOMAIN	Define a domain in the open syntax file
ADD-OPERAND	Define an operand in the open syntax file
ADD-PROGRAM	Define a program in the open syntax file
ADD-STMT	Define a statement in the open syntax file
ADD-VALUE	Define an operand value in the open syntax file

Modifying objects

MODIFY-CMD	Modify a command definition in the open syntax file
MODIFY-CMD-ATTRIBUTES	Modify command attributes
MODIFY-DOMAIN	Modify a domain definition in the open syntax file
MODIFY-OPERAND	Modify an operand definition in the open syntax file
MODIFY-PROGRAM	Modify a program definition in the open syntax file
MODIFY-STMT	Modify a statement definition in the open syntax file
MODIFY-STMT-ATTRIBUTES	Modify statement attributes
MODIFY-VALUE	Modify an operand value definition in the open syntax file

Terminating processing of commands and structures

CLOSE-CMD-OR-STMT	Close a command or statement definition
CLOSE-STRUCTURE	Close structures
SHOW	Output the contents of an open syntax file on SYSOUT or SYSLST

Display functions

SHOW-CORRECTION-INFOR-MATION	Display correction information from the open syntax file (only for purposes of diagnosis)
SHOW-STATUS	Display the name of the open syntax file

5.3 Description of the statements

ADD-CMD
Define command

The ADD-CMD statement is used to define, in the syntax file being processed, the global characteristics for a command, such as

- the command name
- the help text
- the domain to which the command is to belong.

This command is then the “current” object. With the exception of RESULT-INTERNAL-NAME, all names assigned to the command must be unambiguously distinguishable within the overall set of commands.

BS2000 commands (implemented via system modules) can be defined only in a system or group syntax file.

The definitions of the operands and operand values pertaining to a command are placed in the syntax file by means of the ADD-OPERAND and ADD-VALUE statements, respectively, or by means of the COPY statement, rather than by using the ADD-CMD statement.

(part 1 of 3)

```
ADD-CMD

 NAME = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,RESULT-INTERNAL-NAME = *SAME / <alphanum-name 1..8>
,STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30>
,ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
,MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
,HELP = *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |  TEXT = <c-string 1..500 with-low>
,DOMAIN = *NO / list-poss(2000): <structured-name 1..30>
```

continued ➠

(part 2 of 3)

```
,IMPLEMENTOR = *PROCEDURE(...) / *TPR(...) / *APPLICATION(...) / *BY-TPR(...)

   *PROCEDURE(...)
   |    NAME = <c-string 1..280> / *BY-IMON(...)
   |       *BY-IMON(...)
   |       |    LOGICAL-ID = <filename 1..30 without-cat-user-gen-vers>
   |       |
   |       |   ,INSTALLATION-UNIT = <text 1..30 without-sep>
   |       |
   |       |   ,VERSION = *STD / <product-version>
   |       |
   |       |   ,DEFAULT-PATH-NAME = <filename 1..54>
   |       |
   |       |   ,ELEMENT = *NONE / <composed-name 1..64>
   |
   |   ,CALL-TYPE = *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
   |
   |   ,CALL-OPTIONS = *NONE / <c-string 1..1800 with-low>
   |
   |   ,UNLOAD-PROGRAM = *YES / *NO

   *TPR(...)
   |    ENTRY = <name 1..8>
   |
   |   ,INTERFACE = *ASS / *SPL / *ISL(...)
   |       *ISL(...)
   |       |    VERSION = 1 / <integer 1..2>
   |
   |   ,CMD-INTERFACE = *STRING(...) / *TRANSFER-AREA(...) / *NEW(...)
   |       *STRING(...)
   |       |    OUT-CMD-NAME = *SAME / <structured-name 1..30>
   |
   |       *TRANSFER-AREA(...)
   |       |    MAX-STRUC-OPERAND = *STD / <integer 1..3000>
   |       |
   |       |   ,CMD-VERSION = *NONE / <integer 1..999>
   |
   |       *NEW(...)
   |       |    MAX-STRUC-OPERAND = *STD / <integer 1..3000>
   |
   |   ,LOGGING = *BY-SDF / *BY-IMPLEMENTOR
   |
   |   ,INPUT-FORM = *NONE / *INVARIANT / *STANDARD
   |
   |   ,SCI = *NO / *YES

   *APPLICATION(...)
   |    LOGGING = *BY-SDF / *BY-IMPLEMENTOR

   *BY-TPR(...)
   |    TPR-CMD = <structured-name 1..30>

,REMOVE-POSSIBLE =  *YES / *NO
```

continued ➠

(part 3 of 3)

```
,DIALOG-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,DIALOG-PROC-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,GUIDED-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,BATCH-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,BATCH-PROC-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,CMD-ALLOWED = *YES(...) / *NO(...)

   *YES(...)
     |  UNLOAD = *NO / *YES
     |
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |  PRIVILEGE = *SAME / list-poss(64): <structured-name 1..30>

,NEXT-INPUT = *CMD / *STMT / *DATA / *ANY

,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>

   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = <structured-name 1..30>
(External) command name, to be specified when the command is entered. In contrast to STANDARD-NAME and ALIAS-NAME, the user may abbreviate this name when entering the command.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal command name. This name cannot be changed. With the help of the internal command name, SDF identifies a command defined in several syntax files under different external names as being the same command. If the command is implemented via system modules and transfer is formatted (IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...)), the executing module knows the command by its internal name. Unless otherwise specified, SDF-A takes as the internal name the first eight characters (omitting hyphens) of the external name entered for the NAME operand.

RESULT-INTERNAL-NAME = *SAME / <alphanum-name 1..8>
This operand is of significance only for some of the commands implemented via system modules. Implementation via system modules is reserved for Fujitsu Siemens Computers System Software Development. For this reason the RESULT-INTERNAL-NAME operand is not described here.

STANDARD-NAME = *NAME / *NO / list-poss: *NAME / <structured-name 1..30>
Additional external command name, which may be alternatively used when entering the command. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME cannot be deleted so long as the command with this name exists in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE).

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
Additional external command name, which can be alternatively used when the command is entered. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
Additional external command name which defines the shortest permissible abbreviation for the command. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other commands.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct command but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the command name (NAME).
3. The ALIAS-NAMEs and STANDARD-NAMEs of the command must not be shorter than the minimum abbreviation if they are an abbreviation of the command name.

4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the command, and if so, what those texts are.

HELP = <u>*NO</u>
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

> **TEXT = <c-string 1..500 with-low>**
> Help text.
> The help text can contain the special character string "\n" for the line break.

DOMAIN = <u>*NO</u> / list-poss(2000): <structured-name 1..30>
Specifies whether the command is assigned to a domain, and if so, to which one.

IMPLEMENTOR =
Specifies how the command is implemented.

IMPLEMENTOR = <u>*PROCEDURE</u>(...)
The command is implemented via a procedure. Entering the command causes the procedure to be called.

> **NAME=**
> Name of the procedure to be called.
>
> **NAME = <c-string 1..280>**
> Name of the file containing the procedure.
> If SDF-P is loaded, the name of a list variable containing the procedure may also be specified. A variable can be specified in the form '*VARIABLE(VARIABLE-NAME=varname)'.
> Library elements can be specified with
> '*LIBRARY-ELEMENT(LIBRARY=library,ELEMENT=element)'
> If 'library(element)' is specified, the value of CALL-OPTIONS is ignored. This notation should therefore no longer be used.
>
> It is the user's responsibility to ensure that the string to specify the container of the procedure is correctly constructed. Errors made at this position can only be detected when the newly defined command is first called.

NAME = *BY-IMON(...)
The name of the procedure or of the library that contains this procedure as a library element is provided by calling IMON-GPN the installation path manager (see the “IMON” manual [13]).

LOGICAL-ID = <filename 1..30 without-cat-user-gen-vers>
Logical name of the procedure or of the library that contains this procedure as a library element implementing the command, e.g. SYSSPR.

INSTALLATION-UNIT =<text 1..30 without-sep>
Name of the installation unit, e.g. SDF-A.

VERSION = *STD / <product-version>
Version of the installation unit (see also “product-version” on page 15 and page 181).
When *STD is specified, the version which was selected using the command SELECT-PRODUCT-VERSION is used. If this command has not yet been carried out for the relevant installation unit, the highest version is used.

DEFAULT-PATH-NAME = <filename 1..54>
Full file name of a procedure which is called if IMON-GPN is not available or if LOGICAL-ID, INSTALLATION-UNIT or VERSION is not recognized in the system. If the procedure is stored in a library element, the file name specified here designates the library from which the procedure specified in the ELEMENT operand is called.
In the case of other errors the command is rejected with an error message, i.e. the procedure specified here is not called.

ELEMENT = *NONE / <composed-name 1..64>
Specifies if the procedure is stored in a library element.

ELEMENT = <composed-name 1..64>
Name of the library element that contains the procedure. The element name is passed to IMON-GPN.

CALL-TYPE = *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
Defines whether the procedure is called with CALL-PROCEDURE or INCLUDE-PROCEDURE or ENTER-PROCEDURE. CALL-PROCEDURE and ENTER-PROCEDURE can be used to call S procedures as well as non S procedures; INCLUDE-PROCEDURE can only be used to call S procedures (see also the manuals “SDF-P” [12] and “Commands” [4]).

Example:
To call the procedure with
`CALL-PROCEDURE NAME=*LIB-ELEM(LIBRARY=xxx,ELEMENT=yyy)` the command must be defined as follows:

```
ADD-CMD ...,IMPLEMENTOR=*PROC(NAME='*LIB-ELEM(LIBRARY=xxx,
ELEMENT=yyy)',CALL-TYPE=*CALL-PROCEDURE...
```

CALL-OPTIONS = *NONE / <c-string 1..1800 with-low>
Specifies a character string containing additional operands (e.g. LOGGING) for the procedure call (using CALL-PROCEDURE, INCLUDE-PROCEDURE or ENTER-PROCEDURE) in the following format:
`CALL-OPTIONS='operandx=valuex,operandy=valuey,...'.`
This character string must not contain the PROCEDURE-PARAMETERS operand of the CALL-PROCEDURE, INCLUDE-PROCEDURE or ENTER-PROCEDURE command.

UNLOAD-PROGRAM = *YES / *NO
Specifies if a program is to be unloaded when the command defined in the NAME operand is executed in the program via the CMD macro.
The procedure called may not contain any commands defined with CMD-ALLOWED=*YES(UNLOAD=*YES), and in particular no TU program may be called in the procedure.

IMPLEMENTOR = *TPR(...)
The command is implemented via system modules. This alternative for implementing commands is reserved for Fujitsu Siemens Computers System Software Development. For this reason, the structure *TPR(...) is not described here.

IMPLEMENTOR = *APPLICATION(...)
The command is generated by a $CONSOLE application. This option is reserved for system administration. The prerequisites for its use are described in the manual “Introductory Guide to Systems Support” [6]. The required commands CONNECT-CMD-SERVER and DISCONNECT-CMD-SERVER are detailed in the “Commands” [4] manual.

LOGGING = *BY-SDF / *BY-IMPLEMENTOR
The operand is reserved for Fujitsu Siemens Computers System Software Development. For this reason, it is not described here.

IMPLEMENTOR = *BY-TPR(...)
An existing command serves as the basis for the new command.

TPR-CMD = <structured-name 1..30>
Name of a command defined with IMPLEMENTOR=*TPR(...) which is known in the syntax file hierarchy.

Example:
The user syntax file is opened and if necessary positioned using EDIT, and a new X-SET-PROCEDURE-DIALOG command is defined using the following statements:

```
//ADD-CMD X-SET-PROCEDURE-DIALOG,-
//IMPLEMENTOR=*BY-TPR(TPR-CMD=MODIFY-SDF-OPTIONS),PRIVIL=*STD-PROCESSING
//COPY *COMMAND(MODIFY-SDF-OPTIONS),FROM-FILE=$.SYSSDF.SDF.041(*SYSTEM),-
//ATTACHED-INFO=*ONLY
//EDIT *OPERAND(PROCEDURE-DIALOG,ORIGIN=*COMMAND(X-SET-PROCEDURE-DIALOG))
//MODIFY-OPERAND DEFAULT='*YES'
//END
```

After the user syntax file processed in this way, the user can use the new commands. The two following commands now act identically:

```
/X-SET-PROC-DIALOG
/MODIFY-SDF-OPTIONS PROCEDURE-DIALOG=*YES
```

REMOVE-POSSIBLE = *<u>YES</u>* / *NO
Specifies whether the command may be deleted (see the description of the REMOVE statement, page 301).

DIALOG-ALLOWED =
Specifies whether the command is allowed in interactive mode.

DIALOG-ALLOWED = <u>*YES</u>(...)
The command is allowed in interactive mode for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = <u>*SAME</u>
The command is allowed for user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is only allowed for user tasks that have the privileges defined in this list.

DIALOG-ALLOWED = *NO(...)
The command is not allowed in interactive mode for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

DIALOG-PROC-ALLOWED =
Specifies whether the command is allowed in interactive mode within a procedure.

DIALOG-PROC-ALLOWED = *YES(...)
The command is allowed in interactive mode within a procedure for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is only allowed for user tasks with the privileges defined in this list.

DIALOG-PROC-ALLOWED = *NO(...)
The command is not allowed in interactive mode within a procedure for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

GUIDED-ALLOWED =
Specifies whether the command is allowed in guided dialog.

GUIDED-ALLOWED = *YES(...)
The command is allowed in guided dialog for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is allowed only for the privileges defined in this list.

GUIDED-ALLOWED = *NO(...)
The command is not allowed in guided dialog for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

BATCH-ALLOWED =
Specifies whether the command is allowed in batch mode.

BATCH-ALLOWED = *YES(...)
The command is permitted in batch mode for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is only allowed for user tasks with the privileges defined in this list.

BATCH-ALLOWED = *NO(...)
The command is not permitted in batch mode for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = <u>*SAME</u>
The command is not allowed for user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

BATCH-PROC-ALLOWED =
Specifies whether the command is allowed in batch mode within a procedure.

BATCH-PROC-ALLOWED = <u>*YES</u>(...)
The command is permitted in batch mode within a procedure for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = <u>*SAME</u>
The command is allowed for user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is only allowed for user tasks with the privileges defined in this list.

BATCH-PROC-ALLOWED = *NO(...)
The command is not permitted in batch mode within a procedure for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE =
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = <u>*SAME</u>
The command is not allowed for user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

CMD-ALLOWED =
Specifies whether the command can be called with the CMD macro.

CMD-ALLOWED = <u>*YES</u>(...)
The command can be called with the CMD macro. Calling with the CMD macro is permitted for all user tasks which have at least one of the privileges specified for PRIVILEGE. Possible restrictions as to the permissible modes of operation (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED, BATCH-PROC-ALLOWED) do not apply when the command is called with the CMD macro.

> **UNLOAD = <u>*NO</u> / *YES**
> Specifies whether the calling program is to be unloaded. For commands implemented via a command procedure the calling program is always unloaded. Whether or not the calling program is unloaded is defined in this case in the UNLOAD-PROGRAM operand (see the IMPLEMENTOR= PROCEDURE(...) operand, page 137).
>
> **PRIVILEGE =**
> The command may be called by user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).
>
> **PRIVILEGE = <u>*SAME</u>**
> The command may be called by user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 145).
>
> **PRIVILEGE = list-poss(64): <structured-name 1..30>**
> The command is only allowed for user tasks with the privileges defined in this list.

CMD-ALLOWED = *NO(...)
Calling the command with the CMD macro is not permitted for user tasks which have only the privileges specified for PRIVILEGE.

> **PRIVILEGE =**
> The command cannot be called by user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).
>
> **PRIVILEGE = <u>*SAME</u>**
> The command cannot be called by user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 145).
>
> **PRIVILEGE = list-poss(64): <structured-name 1..30>**
> The command is not allowed only for the privileges defined in this list.

NEXT-INPUT =
Specifies what type of input is to follow the command. SDF needs this specification to conduct the guided dialog.

NEXT-INPUT = <u>*CMD</u>
A command is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a command.

NEXT-INPUT = *STMT
A statement is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a statement.
Example: A command implemented by means of a procedure starts a program whose first step is to read in a statement.

NEXT-INPUT = *DATA
Data is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as data.
Example: A command implemented by means of a procedure starts a program whose first step is to read in data.

NEXT-INPUT = *ANY
It is not possible to predict what type of input will follow.

PRIVILEGE =
Specifies the privileges assigned to the command.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the command.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the command.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the command.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the command.

ADD-DOMAIN
Define domain

The ADD-DOMAIN statement is used to define a domain in the syntax file being processed. The name given for the domain must be unambiguously distinguishable from all other domain names.

```
ADD-DOMAIN

 NAME = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,HELP = *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |   TEXT = <c-string 1..500 with-low>
,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |   EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = <structured-name 1..30>
Name of the domain which is to be used in the guided dialog.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal name of the domain. This cannot be changed. With the help of the internal name, SDF identifies a domain defined in several syntax files under different external names as being the same domain. Unless otherwise specified, SDF-A takes as the internal name the first eight characters (omitting hyphens) of the external name entered for the NAME operand.

HELP = *NO / list-poss(2000): <name 1..1>(...)
Specifies whether there are help texts for the domain, and if so, what those texts are.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = <c-string 1..500 with-low>
Help text.
The help text can contain the special character string “\n” for the line break.

PRIVILEGE =
Specifies the privileges assigned to the domain.

PRIVILEGE = <u>*ALL</u>
All privileges currently defined and all subsequently defined privileges are assigned to the domain.

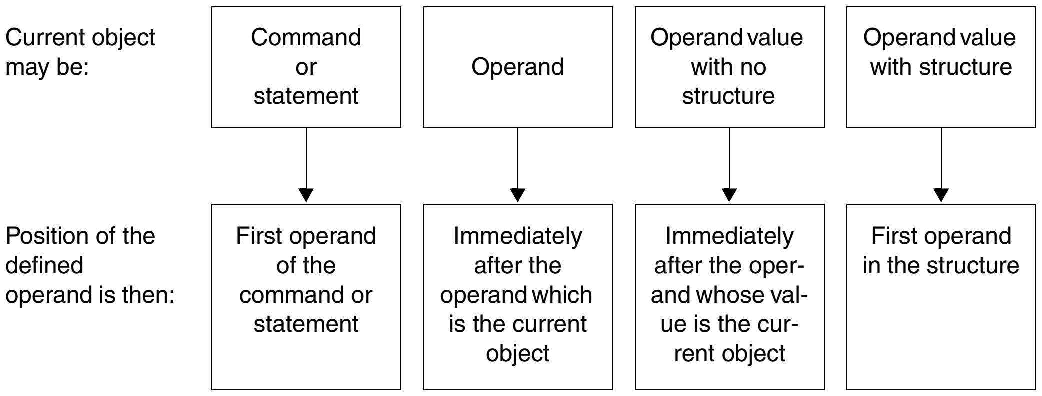
PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the domain.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the domain.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the domain.

ADD-OPERAND
Define operand

The ADD-OPERAND statement is used to define an operand in the open syntax file. The command or statement for which the operand is defined must have already been defined in the syntax file. The position the defined operand receives within this command or statement depends on what the “current object” is at the time ADD-OPERAND is entered. The defined operand is subsequently the current object.


Current object may be:
Command or statement
Operand
Operand value with no structure
Operand value with structure
Position of the defined operand is then:
First operand of the command or statement
Immediately after the operand which is the current object
Immediately after the operand whose value is the current object
First operand in the structure

All names given for the operand must be unique with regard to the other operands at the same level (or in the same structure).

An operand may be defined for a command implemented via system modules only when the command definition is in a group or system syntax file.

The definitions of the operand values belonging to the operand are placed in the syntax file with the ADD-VALUE or COPY statement, rather than the ADD-OPERAND statement.

(part 1 of 2)

ADD-OPERAND

```
 NAME = <structured-name 1..20>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..20>
,ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..20>
,MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
,HELP = *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |  TEXT = <c-string 1..500 with-low>
,DEFAULT = *NONE / *JV(...) / *VARIABLE(...) / <c-string 1..1800 with-low>(...)
    *JV(...)
      |  JV-NAME = <filename 1..54 without-gen-vers>
      | ,ALTERNATE-DEFAULT = *NONE / <c-string 1..1800 with-low>(...)
      |     <c-string 1..1800 with-low>(...)
      |       |  ANALYSE-DEFAULT = *YES / *NO
    *VARIABLE(...)
      |  VARIABLE-NAME = <composed-name 1..255>
      | ,ALTERNATE-DEFAULT = *NONE / <c-string 1..1800 with-low>(...)
      |     <c-string 1..1800 with-low>(...)
      |       |  ANALYSE-DEFAULT = *YES / *NO
    <c-string 1..1800 with-low>(...)
      |  ANALYSE-DEFAULT = *YES / *NO
,SECRET-PROMPT = *NO / *YES
,STRUCTURE-IMPLICIT = *NO / *YES
,REMOVE-POSSIBLE = *YES / *NO
,DIALOG-ALLOWED = *YES / *NO
,DIALOG-PROC-ALLOWED = *YES / *NO
,GUIDED-ALLOWED = *YES / *NO
,BATCH-ALLOWED = *YES / *NO
,BATCH-PROC-ALLOWED = *YES / *NO
```

continued ➠

(part 2 of 2)

```
,LIST-POSSIBLE = *NO / *YES(...)
   *YES(...)
     |  LIMIT = *STD / <integer 1..3000>
     | ,FORM = *NORMAL / *OR
,LINES-IN-FORM = 1 / <integer 1..15>
,PRESENCE = *NORMAL / *EXTERNAL-ONLY / *INTERNAL-ONLY
,RESULT-OPERAND-LEVEL = 1 / <integer 1..5>
,RESULT-OPERAND-NAME = *SAME / <structured-name 1..20> / *POSITION(...) / *LABEL /
                       *COMMAND-NAME
   *POSITION(...)
     |  POSITION = <integer 1..3000>
,CONCATENATION-POS = *NO / <integer 1..20>
,VALUE-OVERLAPPING = *NO / *YES
,OVERWRITE-POSSIBLE = *NO / *YES
,PRIVILEGE = *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = <structured-name 1..20>
(External) operand name, to be specified when the command or statement is entered (but see operand PRESENCE=*INTERNAL-ONLY). The user can abbreviate this name on input, in contrast to the STANDARD-NAME and ALIAS-NAME.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal operand name. With the help of the internal operand name, SDF identifies an operand defined in several syntax files under different external names as being the same operand. Unless otherwise specified, SDF-A takes as the internal operand name the first eight characters (omitting hyphens) of the external name entered for the NAME operand.

STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME /<structured-name 1..20>
Additional external operand name, which can be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted so long as the operand with this name exists in one of the assigned syntax files (see OPEN-SYNTAX-FILE).

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..20>
Additional external operand name, which can be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
Additional external operand name which defines the shortest permissible abbreviation for the operand. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other operands.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct operand but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the operand name (NAME).
3. The ALIAS-NAMEs and STANDARD-NAMEs of the operand must not be shorter than the minimum abbreviation if they are an abbreviation of the operand name.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the operand, and if so, what those texts are.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

> **TEXT = <c-string 1..500 with-low>**
> Help text.
> The help text can contain the special character string “\n” for the line break.

DEFAULT=
Specifies whether there is a default value for the operand.

DEFAULT= *NONE
There is no default value. The operand is mandatory.

DEFAULT= *JV(...)
The operand is optional; its default value is stored in the job variable whose name is specified here. If a job variable is used as a default value, this default value is always analyzed by SDF at execution time. If it is not possible to access the job variable, the alternate default value defined with ALTERNATE-DEFAULT is used. If no alternate default value exists, the operand must be regarded as mandatory (corresponding to DEFAULT=*NONE). Consequently, DEFAULT=*JV(...) must not be used together with PRESENCE=*INTERNAL-ONLY.

> **JV-NAME = <filename 1..54 without-gen-vers>**
> Name of the job variable.

ALTERNATE-DEFAULT =
Alternate default value to be used if errors occur when accessing the job variable.

ALTERNATE-DEFAULT = <u>*NONE</u>
There is no alternate default value.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternate default value, to be specified in accordance with the rules governing the input of operands. The alternate default may thus be given in the form of a list enclosed in parentheses and must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If this default value is contained in the keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = <u>*YES</u> / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

DEFAULT= *VARIABLE(...)
The operand is optional; its default value is stored in the S variable whose name is specified here (see the "SDF-P" [12] User Guide).
If an S variable is used as a default value, this default value is always analyzed by SDF at execution time. If it is not possible to access the S variable, the alternate default value defined with ALTERNATE-DEFAULT is used. If no alternate default value exists, this operand is considered mandatory (corresponding to DEFAULT = <u>*NONE</u>). Consequently, DEFAULT=*VARIABLE(...) must not be used together with PRESENCE=*INTERNAL-ONLY.

VARIABLE-NAME = <composed-name 1..255>
Name of the S variable.

ALTERNATE-DEFAULT =
Alternate default value to be used if errors occur when accessing the S variable.

ALTERNATE-DEFAULT = <u>*NONE</u>
There is no alternate default value.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternate default value, to be specified in accordance with the rules governing the input of operands. The alternate default may thus be given in the form of a list enclosed in parentheses and must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If this default value is contained in the keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = <u>*YES</u> / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

DEFAULT= <c-string 1..1800 with-low>(...)
The operand is optional and has the specified default value. This default value is subject to the same rules that govern the input of operands and may thus be entered in the form of a list enclosed in parentheses. The default value must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If it is contained in a keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = <u>*YES</u> / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

SECRET-PROMPT = <u>*NO</u> / *YES
Specifies whether the operand is treated as a secret operand. The input fields for values of secret operands are kept blank, and logging is suppressed (see also ADD-VALUE..., OUTPUT=*SECRET-PROMPT and ADD-VALUE...,SECRET-PROMPT=*SAME/*NO).

STRUCTURE-IMPLICIT =
Relevant only for an operand contained in a structure and specifies whether the structure containing the operand is implicitly selected via global specification of the operand name when the command or statement is entered.

STRUCTURE-IMPLICIT = <u>*NO</u>
The structure is only implicitly selected when the operand is specified if the operand name is unique throughout the command or statement or within an already selected structure.

STRUCTURE-IMPLICIT = *YES
The structure is implicitly selected when the operand is specified even if other operands with the same name exist. These must be defined using STRUCTURE-IMPLICIT=*NO.
This specification is permitted only for operands whose structure is introduced with the value KEYWORD or KEYWORD-NUMBER.
Example:
```
SHOW-FILE-ATTR ACCESS-METHOD=*ISAM
```
is the abbreviated form of
```
SHOW-FILE-ATTR SEL=*BY-ATTR(ACCESS-METHOD=*ISAM)
```
Simultaneous specification of the operand both within and outside the structure may lead to errors (such as SDF message `CMD0039: MORE THAN ONE VALUE HAS BEEN SPECIFIED FOR AN OPERAND. ONLY THE LAST ONE IS USED`).

REMOVE-POSSIBLE = *YES / *NO
Specifies whether the operand may be deleted (see the REMOVE statement, page 301).

DIALOG-ALLOWED = *YES / *NO
Specifies whether the operand is allowed in interactive mode. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in interactive mode.

DIALOG-PROC-ALLOWED = *YES / *NO
Specifies whether the operand is allowed in interactive mode within a procedure. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in interactive mode within a procedure.

GUIDED-ALLOWED = *YES / *NO
Specifies whether the operand is allowed in guided dialog. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in guided dialog.
GUIDED-ALLOWED=*NO is not suitable for security-related aspects, since operands defined with this setting are shown in the procedure error dialog as well as for /SHOW-CMD and //SHOW-STMT with FORM=*UNGUIDED .

BATCH-ALLOWED = *YES / *NO
Specifies whether the operand is allowed in batch mode. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in batch mode.

BATCH-PROC-ALLOWED = *YES / *NO
Specifies whether the operand is allowed in batch mode within a procedure. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure, are allowed in batch mode within a procedure.

LIST-POSSIBLE =
Specifies whether a list is allowed at the operand position. The ADD-VALUE statement is used to define for which of the operand values a list is allowed.

LIST-POSSIBLE = *NO
No list is allowed.

LIST-POSSIBLE = *YES(...)
A list is allowed.

LIMIT = *STD / <integer 1..3000>
Specifies the maximum number of list elements. Unless otherwise specified, SDF-A assumes the value 2000 (see also page 377).

FORM = *NORMAL / *OR
Specifies whether the list elements are to be addressed individually (NORMAL) or are to be passed to the implementation converted into a single value using logical OR (see section 6.3, “Format of the standardized transfer area”). The latter is appropriate only

for list elements of the data type KEYWORD, for which hexadecimal transfer values have been defined (see ADD-VALUE..,VALUE = <c-string>(..,OUTPUT=<x-string>...). FORM is relevant only when the defined operand pertains to a statement or to a command defined with IMPLEMENTOR=*TPR(..,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). The specification made here must be consistent with the transfer area defined in the implementation.

LINES-IN-FORM = 1 / <integer 1..15>
Specifies the number of input lines in the guided dialog form.

PRESENCE =
Specifies whether the operand is to be suppressed.

PRESENCE = *NORMAL
The operand is not suppressed.

PRESENCE = *EXTERNAL-ONLY
Transfer of the operand to the implementation is suppressed (e.g. an operand that is no longer needed but must be retained at the user interface for compatibility reasons, or an operand that serves merely to group further operands in a structure).

PRESENCE = *INTERNAL-ONLY
The operand is suppressed at the user interface. Together with the then mandatory definition of a default value (see operand DEFAULT=), a fixed value may be assigned to a parameter implemented in this way without an operand being visible to the user in the command or statement format. If a structure is attached to the operand, all of the sub-operands contained in the structure will be integrated into the higher level.
PRESENCE=*INTERNAL-ONLY must not be used together with DEFAULT=*JV(...) or DEFAULT=*VARIABLE(...).

RESULT-OPERAND-LEVEL = 1 / <integer 1..5>
Specifies the structure level at which the operand is to be passed to the implementation. For an operand not attached to a structure, this value must be 1. The following applies to an operand attached to a structure: The RESULT-OPERAND-LEVEL is equal to or less than the structure level on which the operand stands in the input format of the command or statement. It is lower than, equal to or 1 higher than the RESULT-OPERAND-LEVEL of the operand to which the operand value introducing the structure belongs.
For statements and for commands defined with IMPLEMENTOR=*TPR (...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...), see also ADD-VALUE...STRUCTURE=*YES(...,FORM=...).

RESULT-OPERAND-NAME =
Specifies how the implementation identifies the operand in the transfer area or string that SDF passes to it.
Note: SDF uses a transfer area (see page 365ff) for statements and for commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see

ADD-CMD). SDF passes a string in the case of commands defined with IMPLEMENTOR=*PROCEDURE or IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...) (see ADD-CMD).

RESULT-OPERAND-NAME = *SAME
Permissible only when operands are transferred by means of a string. In the string to be passed, the operand has the same name as the one given to it with NAME=.

RESULT-OPERAND-NAME = <structured-name 1..20>
Permissible only when operands are transferred by means of a string. In the string to be passed, the operand has the specified name.

RESULT-OPERAND-NAME = *POSITION(...)
In the transfer area (see page 365ff) or in the string to be passed, the operand has a specified position. A name is not assigned to it.

POSITION = <integer 1..3000>
Specifies the position. For operands attached to a structure, the position is specified relative to the structure, i.e. the first operand in the structure is assigned position 1 if the RESULT-OPERAND-LEVEL that was defined for the current operand is higher than that of the operand at the level above it.

RESULT-OPERAND-NAME = *LABEL
Permissible only for operands in commands defined with IMPLEMENTOR=*TPR(...,CALL=*OLD,...) (see ADD-CMD). This operand value is reserved for Fujitsu Siemens Computers Development and is therefore not described here.

RESULT-OPERAND-NAME = *COMMAND-NAME
Permissible only for operands in commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*STRING,...) (see ADD-CMD). This operand value is reserved for Fujitsu Siemens Computers Development and is therefore not described here.

CONCATENATION-POS = *NO / <integer 1..20>
Specifies whether and, if so, how the operand is to be put together with other input operands to form a single operand in the string to be passed to the implementation. The input operands are concatenated. The position they occupy when concatenated should be specified here. It is presupposed that the transfer to the implementation is in the form of a string (commands that are defined with IMPLEMENTOR=*PROCEDURE or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...); see ADD-CMD). All input operands to be concatenated must have the same RESULT-OPERAND-NAME. If the same position is specified for several input operands, SDF uses the first operand it encounters during analysis

VALUE-OVERLAPPING =
Specifies whether overlapping of data types is to be permitted in the definition of the operand values.

VALUE-OVERLAPPING = *NO
No overlapping is allowed (see section “Mutually exclusive data types” on page 623 for a list of mutually exclusive data types).

VALUE-OVERLAPPING = *YES
Overlapping is allowed.
When the command or statement is entered, SDF checks the operand value using the data type definitions as examples and in the order specified for the operand. SDF outputs an error message if there is no match between the data type and the value entered.
If the data type is KEYWORD(-NUMBER), SDF checks whether the value which has been entered is unique with respect to further definitions of this type. In addition, SDF checks the defined attributes (e.g. length, value range) of the value entered. If these attributes do not apply, the check is continued with the next defined data type.

OVERWRITE-POSSIBLE = *NO / *YES
This is only relevant for statements, and for commands defined with IMPLE-MENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...); see ADD-CMD.
OVERWRITE-POSSIBLE determines whether the operand default value can be replaced by a value created dynamically by the implementation (see the DEFAULT operand in the CMDCST, CMDRST and CMDTST macros). The program-generated value must be a valid operand value. In guided dialog, SDF shows the implementation-specific value in the form display.

PRIVILEGE =
Specifies the privileges assigned to the operand.

PRIVILEGE = *SAME
The operand is assigned the same privileges as those defined for the associated command or statement. If the operand is part of a structure, it is assigned the same privileges as the operand value which introduces the structure.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the operand.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the operand.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the operand.

ADD-PROGRAM
Define program

The ADD-PROGRAM statement is used to define a program in the syntax file being processed. The names given for the program must be unique with regard to all other names defined in the syntax file.

```
ADD-PROGRAM

 NAME = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |   EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
,COMMENT-LINE = *NONE / *STD / <c-string 1..50 with-low>
```

NAME = <structured-name 1..30>
(External) program name which is shown in guided dialog. This name is freely selectable (and need not agree with the module or phase name).

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal program name. This cannot be changed. The program specifies it to SDF when requesting statement input (see the CMDRST and CMDTST macros). Unless otherwise specified, SDF-A takes the first eight characters (omitting hyphens) of the external program name specified for the NAME operand.

PRIVILEGE =
Specifies the privileges assigned to the program.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the program.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the program.

> **EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>**
> Specifies the privileges that are not assigned to the program.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the program.

COMMENT-LINE =
Specifies which program comment lines are to be displayed in the guided dialog. The program comment line appears at the top of guided dialog forms.

COMMENT-LINE = *NONE
No program comment lines are displayed.

COMMENT-LINE = *STD
The version and creation date of the program are displayed in the program comment line. Object modules (elements of type R) have no internal version, so the execution date is shown instead of the creation date.

COMMENT-LINE = <c-string 1..50 with-low>
String to be output as the program comment line.

ADD-STMT
Define statement

The ADD-STMT statement is used to define, in the open syntax file, a statement for a program. This statement then becomes the “current object” (see page 148). The program must already be defined in the syntax file. All names given for the statement must be unique with regard to the set of statements pertaining to the program.
The definitions of the operands and operand values belonging to the statement are placed in the syntax file using the ADD-OPERAND and ADD-VALUE or COPY statements, rather than with the ADD-STMT statement.

ADD-STMT

```
 NAME = <structured-name 1..30>
,PROGRAM = <structured-name 1..30>
,INTERNAL-NAME = *STD / <alphanum-name 1..8>
,STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30>
,ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
,MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
,HELP = *NO / list-poss(2000): <name 1..1>(...)
   <name 1..1>(...)
     |  TEXT = <c-string 1..500 with-low>
,MAX-STRUC-OPERAND = *STD / <integer 1..3000>
,IMPLEMENTATION = *NORMAL / *RESTORE-SDF-INPUT / *SHOW-INPUT-HISTORY / *REMARK /
                  *WRITE-TEXT / *STEP / *END / *MODIFY-SDF / *SHOW-SDF / *EXECUTE / *HOLD /
                  *SHOW-INPUT-DEFAULTS / *RESET-INPUT-DEFAULTS / *HELP-MSG / *SHOW-STMT
,REMOVE-POSSIBLE = *YES / *NO
,DIALOG-ALLOWED = *YES / *NO
,DIALOG-PROC-ALLOWED = *YES / *NO
,GUIDED-ALLOWED = *YES / *NO
,BATCH-ALLOWED = *YES / *NO
,BATCH-PROC-ALLOWED = *YES / *NO
,NEXT-INPUT = *STMT / *DATA / *ANY
,PRIVILEGE = *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
,STMT-VERSION = *NONE / <integer 1..999>
```

NAME = <structured-name 1..30>
(External) statement name, to be specified when the statement is entered. The user can abbreviate this name on input, in contrast to the STANDARD-NAME and ALIAS-NAME.

PROGRAM = <structured-name 1..30>
External name of the program (see ADD-PROGRAM) to which the statement belongs.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal statement name. The program to which the statement belongs knows the statement by its internal name (cf. the CMDRST and CMDTST macros). This name cannot be changed. With the help of the internal statement name, SDF identifies a statement defined in several syntax files under different external names as being the same statement. Unless otherwise specified, SDF-A takes as the internal statement name the first eight characters (omitting hyphens) of the external name entered for the NAME operand.

STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30>
Additional external statement name, which can be alternatively used when entering the statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted so long as the statement with this name exists in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE). STANDARD-NAME is reserved for Fujitsu Siemens Computers Software Development and is therefore not described here.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
Additional external statement name, which can be alternatively used when entering the statement. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
Additional external statement name which defines the shortest permissible abbreviation for the statement. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other statements.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct statement but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.

2. The minimum abbreviation must be derived from the statement name (NAME).

3. The ALIAS-NAMEs and STANDARD-NAMEs of the statement must not be shorter than the minimum abbreviation if they are an abbreviation of the statement name.

4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the statement, and if so, what those texts are.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = <c-string 1..500 with-low>
Help text.
The help text can contain the special character string “\n” for the line break.

MAX-STRUC-OPERAND = *STD / <integer 1..3000>
Number of operand positions to be reserved at the top level in structured transfer. Unless otherwise specified, an operand array is created that is as large as required. However, the array may also be made larger for planned future expansions.

IMPLEMENTATION = *NORMAL / *RESTORE-SDF-INPUT / *SHOW-INPUT-HISTORY / *REMARK / *WRITE-TEXT / *STEP / *END / *MODIFY-SDF / *SHOW-SDF / *EXECUTE / *HOLD /*SHOW-INPUT-DEFAULTS / *RESET-INPUT-DEFAULTS / *HELP-MSG / *SHOW-STMT
This function is provided only for Fujitsu Siemens Computers System Software Development and is therefore not described here.

REMOVE-POSSIBLE = *YES / *NO
Specifies whether the statement may be deleted (cf. REMOVE).

DIALOG-ALLOWED = *YES / *NO
Specifies whether the statement is allowed in interactive mode.

DIALOG-PROC-ALLOWED = *YES / *NO
Specifies whether the statement is allowed in interactive mode within a procedure.

GUIDED-ALLOWED = *YES / *NO
Specifies whether the statement is allowed in guided dialog.

BATCH-ALLOWED = *YES / *NO
Specifies whether the statement is allowed in batch mode.

BATCH-PROC-ALLOWED = *YES / *NO
Specifies whether the statement is allowed in batch mode within a procedure.

NEXT-INPUT =
Specifies what type of input is expected following the statement. SDF needs this specification to conduct the guided dialog.

NEXT-INPUT = *STMT
A statement is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a statement.

NEXT-INPUT = *DATA
Data is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as data.

NEXT-INPUT = *ANY
It is not possible to predict what type of input will follow.

PRIVILEGE =
Specifies the privileges assigned to the statement.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the statement.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the statement.

> **EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>**
> Specifies the privileges that are not assigned to the statement.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the statement.

STMT-VERSION = *NONE / <integer 1..999>
Version of the statement. The value is transferred in the standardized transfer areas. *NONE means that a zero value is specified in the standardized transfer area. STMT-VERSION is ignored if the program to which the statement belongs does not work with the new macros CMDRST and CMDTST or still uses the old format for the standardized transfer area. More information on the format of standardized transfer areas and SDF macros can be found in section “Format of the standardized transfer area” on page 365ff and section “SDF macros” on page 379ff.

ADD-VALUE
Define operand value

The ADD-VALUE statement is used to define an operand value in the open syntax file. The operand for which this value is defined must already be defined in the syntax file.

The position at which the operand value is inserted depends on whether the operand itself or one of its previously defined values is the “current object” (see page 148):

– if the operand itself is the “current object”, the operand value defined will be inserted into the command or statement definition as the first value for this operand;

– if another, previously defined value for this operand is the “current object”, the newly defined operand value will be inserted immediately following this existing value.

In both cases, the newly defined operand value then becomes the “current object”.

All names given for the operand value must be unique with regard to its environment. If the names are not explicitly defined but are instead left up to SDF-A, it can happen, in the case of the data type KEYWORD, that the internal name assembled by SDF-A by default is invalid.

An operand value for a command implemented via system modules may be defined only if the command definition is in a group or system syntax file.

(part 1 of 6)

```
ADD-VALUE

TYPE = *ALPHANUMERIC-NAME(...) / *CAT-ID / *COMMAND-REST(...) / *COMPOSED-NAME(...) /
       *C-STRING(...) / *DATE(...) / *DEVICE(...) / *FIXED(...) / *FILENAME(...) / *INTEGER(...) /
       *KEYWORD(...) / *KEYWORD-NUMBER(...) / *LABEL(...) / *NAME(...) / *PARTIAL-FILENAME(...) /
        *POSIX-PATHNAME(...) / *POSIX-FILENAME(...) / *PRODUCT-VERSION(...) /
       *STRUCTURED-NAME(...) / *TEXT(...) / *TIME(...) / *VSN(...) / *X-STRING(...) / *X-TEXT(...)

   *ALPHANUMERIC-NAME(...)
     |  ,SHORTEST-LENGTH = *ANY / <integer 1..255>
     |  ,LONGEST-LENGTH = *ANY / <integer 1..255>
     |  ,WILDCARD = *NO / *YES(...)
     |     *YES(...)
     |       | TYPE = *SELECTOR / *CONSTRUCTOR
     |       |,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..255>
```

continued ➠

(part 2 of 6)

```
*COMMAND-REST(...)
   LOWER-CASE = *NO / *YES
*COMPOSED-NAME(...)
   SHORTEST-LENGTH = *ANY / <integer 1..1800>
  ,LONGEST-LENGTH = *ANY / <integer 1..1800>
  ,UNDERSCORE = *NO / *YES
  ,WILDCARD = *NO / *YES(...)
     *YES(...)
         TYPE = *SELECTOR / *CONSTRUCTOR
        ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..1800>
*C-STRING(...)
   SHORTEST-LENGTH = *ANY / <integer 1..1800>
  ,LONGEST-LENGTH = *ANY / <integer 1..1800>
  ,LOWER-CASE = *NO / *YES
*DATE(...)
   COMPLETION = *NO / *YES
*DEVICE(...)
   CLASS-TYPE = *DISK(...) / list-poss(2000): *DISK(...) / *TAPE(...)
     *DISK(...)
         EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
        ,SCOPE = *ALL / *STD-DISK
     *TAPE(...)
         EXCEPT = *NO / list-poss(50): <text 1..8 without-sep>
  ,ALIAS-ALLOWED = *YES / *NO
  ,VOLUME-TYPE-ONLY = *NO / *YES
  ,RESULT-VALUE = *BY-NAME / *BY-CODE
*FIXED(...)
   LOWEST = *ANY / <fixed -2147483648..2147483647>
  ,HIGHEST = *ANY / <fixed -2147483648..2147483647>
```

continued ➠

(part 3 of 6)

```
*FILENAME(...)
  | SHORTEST-LENGTH = *ANY / <integer 1..80>
  | ,LONGEST-LENGTH = *ANY / <integer 1..80>
  | ,CATALOG-ID = *YES / *NO
  | ,USER-ID = *YES / *NO
  | ,GENERATION = *YES / *NO
  | ,VERSION = *YES / *NO
  | ,WILDCARD = *NO / *YES(...)
  |     *YES(...)
  |       | TYPE = *SELECTOR / *CONSTRUCTOR
  |       | ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..80>
  | ,PATH-COMPLETION = *NO / *YES
  | ,TEMPORARY-FILE = *YES / *NO
*INTEGER(...)
  | LOWEST = *ANY / <integer -2147483648..2147483647>
  | ,HIGHEST = *ANY / <integer -2147483648..2147483647>
  | ,OUT-FORM = *STD / *BINARY / *PACKED / *UNPACKED / *CHAR / *STD
*KEYWORD(...)
  | STAR = *OPTIONAL / *MANDATORY
*KEYWORD-NUMBER(...)
  | STAR = *OPTIONAL / *MANDATORY
*LABEL(...)
  | SHORTEST-LENGTH = *ANY / <integer 1..8>
  | ,LONGEST-LENGTH = *ANY / <integer 1..8>
*NAME(...)
  | SHORTEST-LENGTH = *ANY / <integer 1..255>
  | ,LONGEST-LENGTH = *ANY / <integer 1..255>
  | ,UNDERSCORE = *NO / *YES
  | ,LOWER-CASE = *NO / *YES
```

continued ➠

(part 4 of 6)

```
   | ,WILDCARD = *NO / *YES(...)
   |
   |     *YES(...)
   |
   |       |  TYPE = *SELECTOR / *CONSTRUCTOR
   |       |
   |       | ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..255>

*PARTIAL-FILENAME(...)

   |  SHORTEST-LENGTH = *ANY / <integer 2..79>
   |
   | ,LONGEST-LENGTH = *ANY / <integer 2..79>
   |
   | ,CATALOG-ID = *YES / *NO
   |
   | ,USER-ID = *YES / *NO
   |
   | ,WILDCARD = *NO / *YES(...)
   |
   |     *YES(...)
   |
   |       |  TYPE = *SELECTOR / *CONSTRUCTOR
   |       |
   |       | ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 2..79>

*POSIX-PATHNAME(...)

   |  SHORTEST-LENGTH = *ANY / <integer 1..1023>
   |
   | ,LONGEST-LENGTH = *ANY / <integer 1..1023>
   |
   | ,WILDCARD = *YES / *NO
   |
   | ,QUOTES = *OPTIONAL / *MANDATORY

*POSIX-FILENAME(...)

   |  SHORTEST-LENGTH = *ANY / <integer 1..255>
   |
   | ,LONGEST-LENGTH = *ANY / <integer 1..255>
   |
   | ,WILDCARD = *YES / *NO
   |
   | ,QUOTES = *OPTIONAL / *MANDATORY

*PRODUCT-VERSION(...)

   |  USER-INTERFACE = *YES (...) / *NO / *ANY(...)
   |
   |     *YES(...)
   |
   |       |  CORRECTION-STATE = *YES / *NO / *ANY
   |
   |     *ANY(...)
   |
   |       |  CORRECTION-STATE = *ANY / *NO / *YES
```

continued ➠

(part 5 of 6)

```
     *STRUCTURED-NAME(...)
        SHORTEST-LENGTH = *ANY / <integer 1..255>
       ,LONGEST-LENGTH = *ANY / <integer 1..255>
       ,WILDCARD = *NO / *YES(...)
          *YES(...)
             TYPE = *SELECTOR / *CONSTRUCTOR
            ,LONGEST-LOGICAL-LEN = *LONGEST-LENGTH / <integer 1..255>
     *TEXT(...)
        SHORTEST-LENGTH = *ANY / <integer 1..1800>
       ,LONGEST-LENGTH = *ANY / <integer 1..1800>
       ,LOWER-CASE = *NO / *YES
       ,SEPARATORS = *YES / *NO
     *TIME(...)
        OUT-FORM = *STD / *BINARY / *CHAR
     *VSN(...)
        SHORTEST-LENGTH = *ANY / <integer 1..6>
       ,LONGEST-LENGTH = *ANY / <integer 1..6>
     *X-STRING(...)
        SHORTEST-LENGTH = *ANY / <integer 1..1800>
       ,LONGEST-LENGTH = *ANY / <integer 1..1800>
     *X-TEXT(...)
        SHORTEST-LENGTH = *ANY / <integer 1..3600>
       ,LONGEST-LENGTH = *ANY / <integer 1..3600>
       ,ODD-POSSIBLE = *YES / *NO
  ,INTERNAL-NAME = *STD / <alphanum-name 1..8>
  ,REMOVE-POSSIBLE = *YES / *NO
  ,SECRET-PROMPT = *SAME / *NO
  ,DIALOG-ALLOWED = *YES / *NO
  ,DIALOG-PROC-ALLOWED = *YES / *NO
  ,GUIDED-ALLOWED = *YES / *NO
  ,BATCH-ALLOWED = *YES / *NO
  ,BATCH-PROC-ALLOWED = *YES / *NO
```

continued ➠

(part 6 of 6)

```
,STRUCTURE = *NO / *YES(...)

   *YES(...)

    |   SIZE = *SMALL / *LARGE
    |
    |  ,FORM = *FLAT / *NORMAL
    |
    |  ,MAX-STRUC-OPERAND = *STD / <integer 1..3000>

,LIST-ALLOWED = *NO / *YES

,VALUE = *NO / list-poss(2000): <c-string 1..1800 with-low>(...)

   <c-string 1..1800 with-low>(...)

    |   STANDARD-NAME = *NAME / *NO / list-poss(2000): *NAME / <structured-name 1..30> /
    |                                     <c-string 1..30>
    |
    |  ,ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
    |
    |  ,GUIDED-ABBREVIATION = *NAME / <structured-name 1..30> / <c-string 1..30>
    |
    |  ,MINIMAL-ABBREVIATION = *NO / <structured-name 1..30> / <c-string 1..30>
    |
    |  ,NULL-ABBREVIATION = *NO / *YES
    |
    |  ,OUTPUT = *SAME / *EMPTY-STRING / *DROP-OPERAND / <c-string 1..1800> / <x-string 1..3600>
    |
    |  ,OUT-TYPE = *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST /
    |              *COMPOSED-NAME / *C-STRING /  *DATE / *DEVICE / *FIXED / *FILENAME /
    |              *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME /
    |              *PARTIAL-FILENAME /  *PRODUCT-VERSION /  *POSIX-PATHNAME /
    |              *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN /
    |              *X-STRING / *X-TEXT
    |
    |  ,OVERWRITE-POSSIBLE = *NO / *YES

,OUTPUT = *NORMAL(...) / *SECRET-PROMPT

   *NORMAL(...)

    |   AREA-LENGTH = *VARIABLE / <integer 1..3000>
    |
    |  ,LEFT-JUSTIFIED = *STD / *YES / *NO
    |
    |  ,FILLER = *STD / <c-string 1..1> / <x-string 1..2>
    |
    |  ,STRING-LITERALS = *NO / *HEX / *CHAR
    |
    |  ,HASH = *NO / *YES(...)
    |
    |      *YES(...)
    |       |   OUTPUT-LENGTH = <integer 2..32>

,PRIVILEGE = *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>

   *EXCEPT(...)
    |   EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

TYPE =
Specifies the data type of the operand value. The various values defined for an operand must not be mutually exclusive with respect to data type (see page 623). (If this is not possible in exceptional cases, the value YES must be specified for VALUE-OVERLAPPING in the ADD-OPERAND or MODIFY-OPERAND statement. Otherwise it is not possible, for example, to define a value of the type NAME and an alternative value of the type STRUCTURED-NAME for an operand. Only the data type KEYWORD may be specified in more than one alternative operand value definition. When a command or statement is entered, SDF checks whether an operand value entered is of the type defined for it. In the following descriptions of the data types, the term “alphanumeric character” is repeatedly used. This may be a letter (A, B, C, ..., Z), a digit (0, 1, 2, ..., 9) or a special character (@, #, $).

TYPE = *ALPHANUMERIC-NAME(...)
Specifies that the operand value is of the data type ALPHANUMERIC-NAME. This is defined as a sequence of alphanumeric characters.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE =
Specifies whether the string can be a wildcard selector ()search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or construction).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *CAT-ID
Specifies that the operand value is of the data type CAT-ID, defined as a sequence of up to 4 characters (without ":" delimiters) from the ranges A-Z and 0-9; the special characters $, @ and # are not permitted. A four-character catalog ID must not begin with the string 'PUB'.

TYPE = *COMMAND-REST(...)
Specifies that the operand value is of the data type COMMAND-REST. This data type is provided only for special purposes for Fujitsu Siemens Computers Software Development and is therefore not described here.

TYPE = *COMPOSED-NAME(...)
Specifies that the operand value is of the data type COMPOSED-NAME. This data type is very similar to the data type FILENAME, with the following differences:

1. The following characters are not permitted: ':' , '(' , ')'
2. Letters, digits and the following special characters are permitted: '@' , '$' , '#' , '.' , '-', and possibly '_' (depending on the value of the UNDERSCORE operand).
3. There are no restrictions for the characters '@' , '$' and '#'
4. The characters '-' and '.' may only be specified as separators between other characters. The following are not permitted, for example: '..' , '--' , '.-' or '-.'. Furthermore, '.' and '-' must not be used at the start or end of a composed name.
5. The length is not limited to 54 characters;
6. The name does not need to include a letter.

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

UNDERSCORE = *NO / *YES
Specifies whether the underscore character '_' is accepted.

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE =
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..1800>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *C-STRING(...)
Specifies that the operand value is of the data type C-STRING. This is defined as a sequence of EBCDIC characters, enclosed in single quotes. It may be prefixed with the letter C. A single quote as a value within the limiting apostrophes must be specified twice.

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

LOWER-CASE = *NO / *YES
Specifies whether lowercase letters within the apostrophes are retained.

TYPE = *DATE(...)
Specifies that the operand value is of the data type DATE. This is defined as a sequence of one four-digit number and two two-digit numbers, joined together by hyphens (<year>-<month>-<day>). The year may also be specified with two digits instead of four.

COMPLETION = <u>*NO</u> / *YES
Specifies whether a two-digit year specification is to be extended. If YES is specified, SDF extends two-digit year specifications (in the form yy-mm-dd) as follows:

– 20yy-mm-dd if yy < 60
– 19yy-mm-dd if yy ≥ 60.

TYPE = *DEVICE(...)
Specifies that the operand value is of the data type DEVICE. The user is offered a list of the disk or tape devices available in the system for operands whose value is defined with the data type DEVICE (see the manual “System Installation” [9]).

CLASS-TYPE = <u>*DISK</u> / list-poss(2000): *DISK(...) / *TAPE(...)
Specifies the class type of the device.

CLASS-TYPE = *DISK(...)
The class type of the device is disk.

EXCEPT = <u>*NO</u> / list-poss(50): <text 1..8 without-sep>
Specifies the disks that will not appear in the list of available devices.

SCOPE =
Specifies whether all disks appear in the list of available devices or only the standard disks specified by DMS (see the “Introductory Guide to DMS” [7]) SCOPE=*ALL is always valid in BS2000/OSD-BC < V3.0.

SCOPE = <u>*ALL</u>
All the disks appear in the list.

SCOPE = *STD-DISK
Only those disks which are specified as standard disks in DMS appear in the list.

CLASS-TYPE = *TAPE(...)
The class type of the device is tape.

EXCEPT = <u>*NO</u> / list-poss(50): <text 1..8 without-sep>
Specifies the devices that will not appear in the list of available devices.

ALIAS-ALLOWED = <u>*YES</u> / *NO
Specifies whether an alias is allowed for the device.

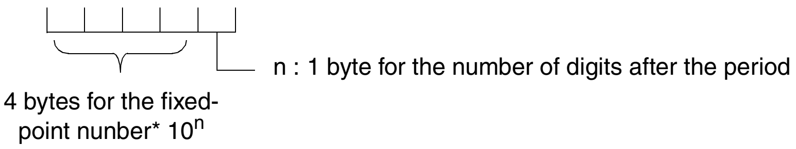
VOLUME-TYPE-ONLY = <u>*NO</u> / *YES
Specifies whether the volume type is accepted.

RESULT-VALUE =
Specifies the form in which SDF transfers information to the implementation.

RESULT-VALUE = *BY-NAME
SDF outputs the external device name. The external device name is 8 characters in length.

RESULT-VALUE = *BY-CODE
SDF outputs the internal device code. The internal device code is 2 bytes long.

TYPE = *FIXED(...)
Specifies that the operand value is of the data type FIXED. This is defined as follows:
[sign][digits].[digits]
[sign] is + or –
[digits] are 0..9
FIXED must consist of at least one digit; it must not exceed 10 characters (digits and a '.').
The value is stored in the standardized transfer area in the following format:


n : 1 byte for the number of digits after the period
4 bytes for the fixed-point nunber* 10^n

LOWEST = *ANY / <fixed -2 147 483648..2 147 483647>
Specifies whether there is a lower limit for the fixed value and, if so, what this limit is.

HIGHEST = *ANY / <fixed -2 147 483648..2 147 483647>
Specifies whether there is an upper limit for the fixed value and, if so, what this limit is.

TYPE = *FILENAME(...)
Specifies that the operand value is of the data type FILENAME. The definition of the character string to be entered is the one given in the BS2000 manual "Introductory Guide to DMS" [7] for fully qualified file names.

SHORTEST-LENGTH = *ANY / <integer 1..80>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..80>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

CATALOG-ID = *YES / *NO
Specifies whether the catalog ID may be specified as part of the file name.

USER-ID = *YES / *NO
Specifies whether the user ID may be specified as part of the file name.

GENERATION = *YES / *NO
Specifies whether a generation number may be specified as part of the file name. If a structure is appended to the file name, NO must be specified.

VERSION = <u>*YES</u> / *NO
Specifies whether a version identifier may be specified as part of the file name. If a structure is appended to the file name, NO must be specified.

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = <u>*NO</u>
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.
The data type <filename x..y with-wild> also contains partially qualified file names, i.e. it is not necessary to additionally define a value for the operand of the data type <partial-filename>.

TYPE =
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = <u>*SELECTOR</u>
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = <u>*LONGEST-LENGTH</u>
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..80>
Specifies the maximum length of the name matched by the wildcard.

PATH-COMPLETION = <u>*NO</u> / *YES
specifies whether the file name is extended to a full path name at the transfer to the implementation. This includes the substitution of aliases by the ACS function. PATH-COMPLETION = *YES can only be specified when CATALOG-ID and USER-ID are permitted and when wildcards in file names are not permitted.

TEMPORARY-FILE = *YES / *NO
specifies whether temporary file names are permitted.

TYPE = *INTEGER(...)
Specifies that the operand value is of the data type INTEGER. This is defined as a sequence of digits, which may be preceded by a sign.

LOWEST = *ANY / <integer -2 147 483648..2 147 483647>
Specifies whether there is a lower limit for the integer, and if so, what that lower limit is.

HIGHEST = *ANY / <integer -2 147 483648..2 147 483647>
Specifies whether there is an upper limit for the integer, and if so, what that upper limit is.

OUT-FORM = *STD / *BINARY / *PACKED / *UNPACKED / *CHAR
Specifies the form in which the integer is to be passed by SDF to the implementation. If passed in a transfer area (see section 6.3, “Format of the standardized transfer area”), SDF-A converts *SDT to *BINARY. If a string is passed (commands defined with IMPLEMENTOR=*PROCEDURE(...) or IMPLEMENTOR=*TPR (...,CMD-INTERFACE-*STRING,...); see ADD-CMD), SDF-A converts OUT-FORM=*STD to *CHAR. If the integer is passed in BINARY format, SDF creates a minimum of 4 bytes (see the OUTPUT operand). If the integer is passed in *PACKED format, SDF creates a minimum of 8 bytes. For *CHAR and *UNPACKED, at least 1 byte is created.

TYPE = *KEYWORD(...)
Specifies that the operand value is of the data type KEYWORD. This is defined as a sequence of alphanumeric characters. This character string may be subdivided into several substrings, joined together by hyphens. Substring sequences may contain periods. The periods must not be at the beginning or end of the substring sequence. The entire string, or, as the case may be, the first substring, must begin with a letter or special character. The value range for an operand value of the type KEYWORD is limited to one or a finite number of precisely defined individual values (see the VALUE operand of this statement).
From SDF-A/SDF Version 2.0 onwards, the value ’*...’ is also accepted. This value can be used where a list of keywords has to be defined for an operand (e.g. the definition for all external devices). This permits new keywords (e.g. new device names) to be inserted without having to modify the syntax file, since ’*...’ causes SDF to accept additional values and to analyze them as keywords. The data type KEYWORD may be up to 30 characters long.

STAR =
Specifies whether the string must be preceded by an asterisk when entered.

STAR = *OPTIONAL
An asterisk may be prefixed, but need not be. If an asterisk is prefixed, it is suppressed when the operand value is passed to the implementation.

STAR = *MANDATORY
An asterisk must be prefixed (necessary in order to distinguish between alternative data types).

TYPE = *KEYWORD-NUMBER(...)
Specifies that the operand value is of the data type KEYWORD-NUMBER. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *LABEL(...)
Specifies that the operand value is of the data type LABEL. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *NAME(...)
Specifies that the operand value is of the data type NAME. This is defined as a sequence of alphanumeric characters, beginning with a letter or special character.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

UNDERSCORE = *NO / *YES
Specifies whether an underscore character (_) is accepted.

LOWER-CASE = *NO / * YES
Specifies whether lowercase characters are to be retained.

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE =
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = <u>*LONGEST-LENGTH</u>
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *PARTIAL-FILENAME(...)
Specifies that the operand value is of the data type PARTIAL-FILENAME. The definition of the string to be entered is the one given in the BS2000 manual “Introductory Guide to DMS” [7] for partially qualified file names.

SHORTEST-LENGTH = <u>*ANY</u> / <integer 2..79>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = <u>*ANY</u> / <integer 2..79>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

CATALOG-ID = <u>*YES</u> / *NO
Specifies whether the catalog ID may be specified as part of the file name.

USER-ID = <u>*YES</u> / *NO
Specifies whether the user ID may be specified as part of the file name.

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = <u>*NO</u>
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE =
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 2..79>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *POSIX-PATHNAME(...)
Defines the data type of the operand value as POSIX-PATHNAME. The character string entered here must comply with the conventions below:

– The following characters are allowed: letters, digits, and the characters '_', '-', '.', and '/' (the '/' always serves as a delimiter between directories and file names). Control characters are not allowed.
– A POSIX-PATHNAME consists of POSIX-FILENAMEs, separated by '/'. The total length of a POSIX-PATHNAME must not exceed 1023 characters.
– A '\' (backslash) is used to escape metacharacters in all POSIX-specific names; the backslash character itself is not included when counting the length. The metacharacter * is likewise excluded from the count.

Metacharacters are characters used in wildcard patterns. The following metacharacters may be used:

*	matches zero, one, or any number of characters
?	matches any single character
[S]	matches any single character from the defined character set S,
[!S]	matches any single character that is not an element of the defined character set S,
where S is	a set of fixed characters (e.g. abcd) or a range of characters (e.g. a-d) or a combination of the above (set and range).

POSIX-PATHNAMEs which contain characters other than those listed above or which begin with a '?' or '!' must be enclosed within single quotes on input (as in the case of C-STRINGs). Since the single quotes are not a part of the file name, they are removed by

SDF in the standardized transfer area or the transferred string. Single quotes that are part of a file name must be duplicated.
To avoid compatibility problems, the C-string syntax should always be used in procedures.

SHORTEST-LENGTH = *ANY / <integer 1..1023>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..1023>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *YES / *NO
Defines whether metacharacters (see above) may be specified in the name (instead of characters/strings) when entering a command or statement.

QUOTES = *OPTIONAL / *MANDATORY
Specifies whether the path name can (*OPTIONAL) or must (*MANDATORY) be enclosed within single quotes on input.

TYPE = *POSIX-FILENAME(...)
Defines the data type of the operand value as POSIX-FILENAME. The string to be entered here must comply with the conventions listed for POSIX-PATHNAMEs (see page 179f), but with the following restrictions:
– the slash (/) is not allowed
– the maximum length is limited to 255 characters.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *YES / *NO
Defines whether metacharacters (see POSIX-PATHNAME) may be specified in the name (instead of characters/strings) when entering a command or statement.

QUOTES = *OPTIONAL / *MANDATORY
Specifies whether the file name can (*OPTIONAL) or must (*MANDATORY) be enclosed in single quotes.

TYPE = *PRODUCT-VERSION(...)
Specifies that the operand value is of the data type PRODUCT-VERSION. The product version has the following format:

```
[[C]'][V][m]m.naso[']        m, n: digit (0..9)
               | |           a: letter
               | |           s,o: digit
               | |
               | |________   Correction state
               |__________   Release status
```

C, V and the apostrophe are optional characters and are suppressed in the SDF transfer area.

USER-INTERFACE =
Indicates whether the release status of the user interface can or must be specified.

USER-INTERFACE = *YES(...)
The release status of the user interface must be specified.

CORRECTION-STATE =
Indicates whether the correction state can or must be specified.

CORRECTION-STATE = *YES
The correction state must be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]'][V][m]m.naso['].

CORRECTION-STATE = *NO
The correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]'][V][m]m.na['].

CORRECTION-STATE = *ANY
The correction state can be specified.

USER-INTERFACE = *NO
The release status of the user interface and the correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]'][V][m]m.n['].

USER-INTERFACE = *ANY(...)
The release status of the user interface can be specified.

CORRECTION-STATE =
Indicates whether the correction state can or must be specified.

CORRECTION-STATE = *ANY
The correction state can be specified.

CORRECTION-STATE = *NO
The correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.na[’].

CORRECTION-STATE = <u>*YES</u>
The correction state must be specified if the release status is specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.naso[’].

TYPE = *STRUCTURED-NAME(...)
Specifies that the operand value is of the data type STRUCTURED-NAME. This is defined as a sequence of alphanumeric characters. This string may be subdivided into several substrings, joined together by hyphens. The entire string, or, as the case may be, the first substring, must begin with a letter or special character.

SHORTEST-LENGTH = <u>*ANY</u> / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = <u>*ANY</u> / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD =
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = <u>*NO</u>
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE =
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = <u>*SELECTOR</u>
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN =
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *TEXT(...)
Specifies that the operand value is of the data type TEXT. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *TIME(...)
Specifies that the operand value is of the data type TIME. This is defined as a sequence of one, two or three unsigned numbers, joined together by colons (<hours>[:<minutes>[:<seconds>]]). The specifications for hours, minutes and seconds must not contain more than two digits. Numbers with less digits may be preceded by leading zeros. The value range for minutes and seconds lies between 0 and 59.

OUT-FORM = *STD / *BINARY / *CHAR
Specifies the format that SDF uses to represent numbers in the time specification passed to the implementation.

OUT-FORM = *STD
If the transfer is made in a transfer area (see the section “Format of the standardized transfer area” on page 365), then the time specification is passed in the binary format. When passing in a string (for commands that are defined with IMPLEMENTOR= *PROCEDURE(...) or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...), see ADD-CMD), the time specification is passed in the character format.

OUT-FORM = *BINARY
The time specification is passed in the binary format.

OUT-FORM = *CHAR
The time specification is passed in the character format.

TYPE = *VSN(...)
Specifies that the operand value is of data type VSN. SDF is able to distinguish between two types:

a) VSN with a period:
This VSN must consist of 6 characters. Apart from a single period, only the letters A-Z and digits 0-9 are accepted. Such a VSN has the format `pvsid.sequence-number`,

where: `pvsid` consists of 2 to 4 characters and `sequence-number` of 1 to 3 characters. This subordinate type of VSN must not be preceded by PUB: PUBA.0 or PUB.02 would be invalid. The period may be the third, fourth or fifth character of the VSN.

b) VSN without a period:
This VSN consists of a string of up to 6 characters. Only letters A-Z, digits 0-9 and special characters $, @ and # are allowed. Such a VSN may start with “PUB”. In this case, the subsequent characters must not be special characters (e.g. PUB@1 or PUB$## will be rejected). Furthermore, VSNs beginning with the string ’PUB’ must be 6 characters long.
SDF makes no further distinctions between public or private VSNs or PUBRES.

SHORTEST-LENGTH = *ANY / <integer 1..6>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..6>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

TYPE = *X-STRING(...)
Specifies that the operand value is of the data type X-STRING. This is defined as a sequence of hexadecimal characters (digits 0 through 9, capital letters A through F), enclosed in apostrophes. It is prefixed by the letter X.

SHORTEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

TYPE = *X-TEXT(...)
Specifies that the operand value is of the data type X-TEXT. This data type is very similar to the data type X-STRING, but it is not enclosed in single quotes and is not preceded by the letter ’X’.

SHORTEST-LENGTH = *ANY / <integer 1..3600>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..3600>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

ODD-POSSIBLE = *YES / *NO
Specifies whether an odd number of characters is accepted.

INTERNAL-NAME = *STD / <alphanum-name 1..8>
Internal operand value name. SDF identifies an operand value by means of this name. Unless otherwise specified, SDF-A takes the first eight characters (omitting hyphens) of the data type specified for the TYPE operand. For operand values of the data type KEYWORD, the default value for SDF-A is the first eight characters (omitting hyphens) of the first individual value specified for the VALUE operand.

SECRET-PROMPT = *SAME / *NO
Specifies whether the operand value is to be treated as secret. SECRET-PROMPT= *SAME assumes the setting of the operand to which the defined operand value belongs (see ADD-OPERAND ...,SECRET-PROMPT= , page 153).The input fields for values of secret operands are kept blank, and logging is suppressed.
If SECRET-PROMPT=*NO is specified, the operand value is not treated as secret. If a secret operand is not preset with a secret value, the input field is not kept blank. The input field can be kept blank by entering the ^ character or a value defined using OUTPUT=*SECRET-PROMPT.

REMOVE-POSSIBLE = *YES / *NO
Specifies whether the operand value may be deleted (see REMOVE).

DIALOG-ALLOWED = *YES / *NO
Specifies whether the operand value is allowed in interactive mode. Specifying YES presupposes that the operand to which the value pertains is allowed in interactive mode.

DIALOG-PROC-ALLOWED = *YES / *NO
Specifies whether the operand value is allowed in interactive mode within a procedure. Specifying YES presupposes that the operand to which the value pertains is allowed in interactive mode within a procedure.

GUIDED-ALLOWED = *YES / *NO
Specifies whether the operand value is allowed in guided dialog. Specifying YES presupposes that the operand to which the value pertains is allowed in guided dialog.

BATCH-ALLOWED = *YES / *NO
Specifies whether the operand value is allowed in batch mode. Specifying YES presupposes that the operand to which the value pertains is allowed in batch mode.

BATCH-PROC-ALLOWED = *YES / *NO
Specifies whether the operand value is offered in batch mode within a procedure. Specifying YES presupposes that the operand to which the value pertains is allowed in batch mode within a procedure.

STRUCTURE =
Specifies whether the operand value introduces a structure.

STRUCTURE = *NO
The operand value does not introduce a structure.

STRUCTURE = *YES(...)
The operand value introduces a structure.

SIZE = <u>*SMALL</u> / *LARGE
Specifies whether, at the MINIMUM or MEDIUM level of guided dialog, the structure is to be integrated into the higher-level form (SMALL), or if a separate form is to be created for it (LARGE).

FORM = <u>*FLAT</u> / *NORMAL
Relevant only for statements and for commands defined with IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). If LIST-ALLOWED=*YES is specified, FORM=NORMAL must also be specified. In the default case (*FLAT), the structure is linearized for the implementation in the transfer area, and the operands for the structure are integrated into a higher-ranking operand array. In the NORMAL case, the structure receives its own operand array. In it, the operands are passed for which a higher structure level is defined for RESULT-OPERAND-LEVEL than for the operand to which the defined operand value that introduces a structure pertains (see ADD-OPERAND ...,RESULT-OPERAND-LEVEL=,... and section “Format of the standardized transfer area” on page 365).

MAX-STRUC-OPERAND = <u>*STD</u> / <integer 1..3000>
Number of operand positions to be reserved in the structured transfer. Unless otherwise specified, the operand array will be made as large as necessary for the structure. However, it may also be made larger to accommodate planned expansions. This operand refers to the structure introduced by the operand value, and is only relevant when NORMAL was specified for the preceding operand.

LIST-ALLOWED = <u>*NO</u> / *YES
Specifies whether a list may be specified for the operand value when the command or statement is entered. This presupposes that the operand to which the value pertains was defined with LIST-POSSIBLE=*YES (see ADD-OPERAND). For statements and commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) and STRUCTURE=*YES with FORM=*FLAT, only LIST-ALLOWED=*NO may be specified.

VALUE =
Specifies which values are permitted as input.

VALUE = <u>*NO</u>
All values corresponding to the operand type are permitted. Limitations exist only insofar as these have been specified in the definition of the operand type (e.g. length restrictions). For operands of the type KEYWORD, NO is not permitted.

VALUE = list-poss(2000): <c-string 1..1800 with-low>(...)
The permissible values are limited to the specified values. In contrast to STANDARD-NAME and ALIAS-NAME, the user may abbreviate the specified values of the type KEYWORD during input. For values of the type KEYWORD, a list of single values is not permissible (a separate ADD-VALUE must be issued for each individual value).

STANDARD-NAME = *NAME / *NO / list-poss(2000):<structured-name 1..30>/ <c-string 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the standard name of the operand value, and may be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted and is reserved for Fujitsu Siemens Computers Software Development.

ALIAS-NAME = *NO / list-poss(2000): <structured-name 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the alias name for the operand value, which may be alternatively used when the command or statement is entered. It must not be abbreviated when entered. In contrast to STANDARD-NAME, an ALIAS-NAME may be deleted.

GUIDED-ABBREVIATION = *NAME / <structured-name 1..30> / <c-string 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the name by which SDF identifies the operand value at the medium help level of guided dialog. Unless otherwise specified, SDF-A takes as GUIDED-ABBREVIATION the individual value entered for the VALUE operand.

MINIMAL-ABBREVATION = *NO / <structured-name 1..30> / <c-string 1..30>
Only for operand values of the type KEYWORD:
Determines the shortest permissible abbreviation for the operand value. Any shorter abbreviation will not be accepted by SDF.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct operand value but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the KEYWORD.
3. The ALIAS-NAMEs and STANDARD-NAMEs of the operand value must not be shorter than the minimum abbreviation if they are an abbreviation of the operand value.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

NULL-ABBREVIATION = *NO / *YES
Relevant only for operand values of the type KEYWORD defined with STRUCTURE=*YES. It specifies whether the operand value may be omitted in front of the opening structure parenthesis when the command or statement is entered, e.g. an operand value introducing a structure, when there are no alternative values for the operand.

OUTPUT =
Specifies which value is passed to the implementation when OUTPUT=*NORMAL applies (see below).

OUTPUT = *SAME
The value specified for the VALUE operand is passed.

OUTPUT = *EMPTY-STRING
An empty string is passed.

OUTPUT = *DROP-OPERAND
Transfer of the operand is suppressed.

OUTPUT = <c-string 1..1800>
The value specified here is passed.

OUTPUT = <x-string 1..3600>
The value specified here is passed.

OUT-TYPE = *SAME/*ALPHANUMERIC-NAME/*CAT-ID/*COMMAND-REST/ *COMPOSED-NAME/*C-STRING/*DATE/*DEVICE/*IXED/*FILENAME/ *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME / *PARTIAL-FILENAME / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN / *X-STRING / *X-TEXT
Relevant only for statements and for commands defined with IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). OUT-TYPE specifies whether SDF changes the data type of the operand value, and if so, in what way, when the value is stored in the transfer area for the implementation (see section “Format of the standardized transfer area” on page 365). If not otherwise specified, SDF does not change the data type.

OVERWRITE-POSSIBLE = *NO / *YES
Relevant only for statements and for commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). OVERWRITE-POSSIBLE specifies whether the operand value entered is overwritten by a value dynamically generated by the implementation (see the DEFAULT operand in the CMDRST and CMDTST macros). The program-generated value must represent a valid operand value. In guided dialog, SDF shows the implementation-specific value in the form.
Example: The value UNCHANGED in MODIFY statements for SDF-A is overwritten by SDF-A with the current value.

OUTPUT =
Specifies whether, and if so, in what way SDF is to pass the operand value entered to the implementation.

OUTPUT = <u>*NORMAL</u>(...)
SDF passes the operand value to the implementation. From SDF-A Version 2.0 onwards, the specifications AREA-LENGTH=, LEFT-JUSTIFIED= and FILLER= are no longer restricted to certain operand values.

AREA-LENGTH = <u>*VARIABLE</u> / <integer 1..3000>
Specifies the length of the field in which SDF stores the operand value for the implementation. The field must be large enough to hold the maximum value which can be entered during execution. If the value specified for AREA-LENGTH is less than the value defined for LONGEST-LENGTH, SDF issues a warning and accepts the value specified for AREA-LENGTH.

There are two possible cases when a value that is greater than AREA-LENGTH and less than LONGEST-LENGTH is analyzed:

1. Values that are of type C-STRING with LONGEST-LENGTH 32 and which are part of a secret operand are compressed by SDF and stored in a hexadecimal string with the length defined for AREA-LENGTH. This behavior is typically used for passwords. The passwords are compressed with the aid of a hash algorithm and are protected against unauthorized access by their hexadecimal storage format.

   Notes:
   – The same hash algorithm is used as in the HASH-STRING function provided in SDF-P.
   – The command server or the program that processes the value analyzed by SDF may need to be adapted if the password definition was changed in order to support hash passwords. The hash value returned by SDF may otherwise be rejected by the semantic analysis module of the program or command server.

2. In all other cases, i.e. those which deviate from case 1, the value is truncated to the length specified for AREA-LENGTH.

LEFT-JUSTIFIED = <u>*STD</u> / *YES / *NO
Relevant only when a fixed length has been defined for the field in which the operand value is stored. LEFT-JUSTIFIED specifies whether SDF stores the operand value in the field left-justified or right-justified. SDF-A changes STD to NO for numeric values and to YES for all other values.

FILLER = <u>*STD</u> / <c-string 1..1> / <x-string 1..2>
Relevant only when a fixed length has been defined for the field in which the operand value is stored. FILLER specifies the character with which SDF pads the field when necessary. SDF-A changes *STD to X’00’ for values of the type X-STRING or INTEGER, and to C’␣’ (blank) for all other values.

STRING-LITERALS = *NO / *HEX / *CHAR
Specifies whether SDF converts the operand value into the data type X-STRING or C-STRING to be passed on to the implementation. In the default case, SDF does not change the data type. It must then be kept in mind that, for operand values of the type C-STRING, SDF transfers only the content of the string (without the apostrophes and the prefixed C). This operand is valid only if VALUE=*NO is specified.

HASH = *NO / *YES(...)
Specifies whether the input value can be converted to a value with a defined length using a hash algorithm.

HASH = *YES(...)
Is only permitted for operand values of the data type C-STRING which are defined with LONGEST-LENGTH ≤ 32.
The other operands in the structure OUTPUT=*NORMAL(..) do not format the value until after the hash function has been carried out. The value then has the data type X-STRING and may then contain unprintable characters.

OUTPUT-LENGTH = <integer 2..32>
Length of the value into which the input value can be converted.

OUTPUT = *SECRET-PROMPT
The operand value is not passed to the implementation, but instead causes SDF to request the user to enter one of the alternative values for the operand. The input that follows is then not displayed and is not logged. Prerequisites are:
– the operand is defined as secret (see ADD-OPERAND ...,SECRET-PROMPT=*YES)
– input is made in unguided dialog or in a foreground procedure
– a single value is specified as permissible input for the operand value defined with SECRET-PROMPT (see the operand VALUE=<c-string>; in the normal case it is of the type KEYWORD).

The following case occurs in an application with guided dialog:
The input field for a secret operand set to a value which is not secret, is not kept blank. The input field can be kept blank by entering the defined value with OUTPUT=*SECRET-PROMPT.

PRIVILEGE =
Specifies the privileges assigned to the operand value.

PRIVILEGE = *SAME
The operand value is assigned the same privileges as the operand for which this operand value is defined.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined by *EXCEPT(...), the operand value is assigned all currently defined privileges and all privileges defined subsequently.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies those privileges that are not assigned to the operand value.

PRIVILEGE = list-poss(64): <structured-name 1..30>
The operand value is assigned only those privileges specified in this list.

CLOSE-CMD-OR-STMT
Conclude definition of command or statement

The CLOSE-CMD-OR-STMT statement is used to conclude editing of a command or statement definition, and causes SDF-A to check the definition for consistency. Any structures that have not yet been closed are closed. If errors are detected, SDF-A issues appropriate messages and takes standard measures to correct the errors.

When the editing of a command of statement definition is terminated with a different statement, e.g. ADD-CMD, ADD-STMT or END, the same consistency checks are thereby implicitly initiated. However, any error messages regarding the completed definition are then output, mixed with any messages arising from the statement entered.

CLOSE-CMD-OR-STMT

This statement has no operands.

CLOSE-STRUCTURE
Close structure

The CLOSE-STRUCTURE statement is used to close structures in command or statement definitions.

CLOSE-STRUCTURE
LEVEL = ***CUR**RENT / ***ALL**

LEVEL =
Specifies which structures are closed.

LEVEL = *CURRENT
The structure being edited is closed after the “current object”. The operand value introducing the structure becomes the new “current object”.

LEVEL = *ALL
The structure being edited is closed after the “current object”. Any structures ranked above this structure which are still open are likewise closed. The operand value introducing the lowest of all these structures becomes the new “current object”.

COMPARE-SYNTAX-FILE
Compare objects from two syntax files

Objects in two different syntax files can be compared using the COMPARE-SYNTAX-FILE statement.
First of all, SDF-A searches for the object to be compared using the external name in the first syntax file. If it exists, SDF-A searches for this object using its internal name in the second syntax file. Therefore only objects with the same internal names can be compared.
In the comparison log, SDF-A always uses the first default name of the object, i.e. the output can display two different external names for objects with the same internal names.

(part 1 of 4)

```
COMPARE-SYNTAX-FILE

OBJECT = *GLOBAL-INFORMATION / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) /
            *STATEMENT(...) / *OPERAND(...) / *VALUE(...)

   *DOMAIN(...)
     |  NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

   *COMMAND(...)
     |  NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

   *PROGRAM(...)
     |  NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

   *STATEMENT(...)
     |  NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
     |
     | ,PROGRAM = <structured-name 1..30>

   *OPERAND(...)
     |  OPERAND-L1 = <structured-name 1..20>
     |
     | ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
     |              *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
     |              *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
     |              *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
     |              *POSIX-PATHNAME / *POSIX-FILENAME /
     |              <composed-name 1..30>
```

continued ➠

(part 2 of 4)

```
          ,OPERAND-L2 = *NO / <structured-name 1..20>

          ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                      *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                      *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                      *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                      *POSIX-PATHNAME / *POSIX-FILENAME /
                      <composed-name 1..30>

          ,OPERAND-L3 = *NO / <structured-name 1..20>

          ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                      *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                      *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                      *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                      *POSIX-PATHNAME / *POSIX-FILENAME /
                      <composed-name 1..30>

          ,OPERAND-L4 = *NO / <structured-name 1..20>

          ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                      *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                      *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                      *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                      *POSIX-PATHNAME / *POSIX-FILENAME /
                      <composed-name 1..30>

          ,OPERAND-L5 = *NO / <structured-name 1..20>

          ,ORIGIN = *COMMAND(...) / *STATEMENT(...)

             *COMMAND(...)
                NAME = <structured-name 1..30>

             *STATEMENT(...)
                NAME = <structured-name 1..30>

               ,PROGRAM = <structured-name 1..30>

       *VALUE(...)

           OPERAND-L1 = <structured-name 1..20>

          ,VALUE-L1 = *NO / COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                      *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                      *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                      *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                      *POSIX-PATHNAME / *POSIX-FILENAME /
                      <composed-name 1..30>
```

continued ➠

(part 3 of 4)

```
,OPERAND-L2 = *NO / <structured-name 1..20>

,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME /
                  <composed-name 1..30>

,OPERAND-L3 = *NO / <structured-name 1..20>

,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME /
                  <composed-name 1..30>

,OPERAND-L4 = *NO / <structured-name 1..20>

,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME /
                  <composed-name 1..30>

,OPERAND-L5 = *NO / <structured-name 1..20>

,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME /
                  <composed-name 1..30>

,ORIGIN = *COMMAND(...) / *STATEMENT(...)

   *COMMAND(...)
    |  NAME = <structured-name 1..30>
   *STATEMENT(...)
    |  NAME = <structured-name 1..30>
    |
    | ,PROGRAM = <structured-name 1..30>
```

continued ➠

(part 4 of 4)

```
,INPUT-FILE = <filename 1..54 without-gen-vers>(...)

   <filename1..54 without-gen-vers>(...)

      |  TYPE = *SYSTEM / *USER / *GROUP

,COMPARE-FILE = <filename 1..54 without-gen-vers>(...)

   <filename 1..54 without-gen-vers>(...)

      |  TYPE = *SYSTEM / *USER / *GROUP

,ATTACHED-INFORMATION = *YES / *NO

,OUTPUT = *SYSOUT / *SYSLST(...)

   *SYSLST(...)

      |  SYSLST-NUMBER = *STD / <integer 1..99>

,COMPARE-ATTRIBUTES = *ALL / *USER-INTERFACE / *PRIVILEGES-ALLOWNESS
```

OBJECT =
Type of object whose definition is to be compared.

OBJECT = *GLOBAL-INFORMATION
Specifies that the global information of syntax files is to be compared.

OBJECT = *DOMAIN(...)
Specifies that the definitions of domains are to be compared.

> **NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>**
> The definitions of all domains or of the domains named are compared.

OBJECT = *COMMAND(...)
Specifies that the definitions of commands are to be compared.

> **NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>**
> The definitions of all commands or of the commands named are compared.

OBJECT = *PROGRAM(...)
Specifies that the definitions of programs are to be compared.

> **NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>**
> The definitions of all programs or of the programs named are compared.

OBJECT = *STATEMENT(...)
Specifies that the definitions of statements are to be compared.

NAME = *ALL / <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of all statements or of the statements named are compared.

PROGRAM = <structured-name 1..30>
Name of the program to which the statements belong.

OBJECT = *OPERAND(...)
Specifies that the definition of an operand is to be compared. If this operand is included in a structure, it is specified by the path leading to it, i.e. by specifying the operands and operand values that introduce the structure in this path. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified (it can even be omitted). An operand which is not absolutely essential for the identification of the operand definition to be compared, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first VALUE-Li = *NO, SDF-A takes the operand defined by OPERAND-Li as the one whose definition is to be compared. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for VALUE-Li, the value specified for OPERAND-Li + 1 must also be other than *NO.

OPERAND-L1 = <structured-name 1..20>
Specifies the operand whose definition is to be compared (VALUE-L1 = *NO) or an operand in the path leading to it (VALUE-L1 ≠ *NO). <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *NO/*COMMAND-REST/*INTEGER/*X-STRING/*C-STRING/ *NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/*FILENAME/ *PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/ *COMPOSED-NAME / *X-TEXT / *FIXED / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30> /
*NO means that the definition of OPERAND-L1 is to be compared. Otherwise, an operand value that introduces a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be compared. If the operand value introducing the structure is of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE= <c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value introducing the structure is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO /<structured-name 1..20>
*NO means that OPERAND-L2 is irrelevant for the specification of the operand whose definition is to be compared. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L1 is to be specified. This operand is either the one whose definition is to be compared (VALUE-L2 = *NO) or one that is in the path leading to it (VALUE-L2 ≠ *NO).

VALUE-L2 = analogous to VALUE-L1
*NO means that VALUE-L2 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be copied. For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand whose definition is to be compared. Otherwise the name of an operand that is unique within the structure determined by VALUE-L2 is to be specified. This operand is either the one whose definition is to be compared (VALUE-L3 = *NO) or one that is in the path leading to it (VALUE-L3 ≠ *NO).

VALUE-L3= analogous to VALUE-L1
*NO means that VALUE-L3 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be compared. For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L2.

VALUE-L4 = analogous to VALUE-L1
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L2.

ORIGIN =
Specifies the command or statement to which the operand definition to be compared pertains.

ORIGIN = *COMMAND (...)
The operand definition pertains to a command.

NAME = <structured-name 1..30>
Name of the command.

ORIGIN = *STATEMENT(...)
The operand definition pertains to a statement.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *VALUE(...)
Specifies that the definition of an operand value is to be compared. This operand value is specified by the path leading to it, i.e. by specifying the operands and operand values introducing the structure in this path. If the operand value pertains to an operand that is not attached to any structure, the path contains only this operand. If the operand value does pertain to an operand attached to a structure, the path also includes the higher-ranking operands as well as the associated operand values introducing the structure. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified. An operand that is not absolutely essential to identify the operand value definition to be compared, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first OPERAND-Li+1=*NO,
SDF-A takes the operand value defined by VALUE-Li as the one whose definition is to be compared. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for OPERAND-Li, the value specified for VALUE-Li must also be other than *NO.

OPERAND-L1 = <structured-name 1..20>
Specifies the operand to which the operand value whose definition is to be compared pertains (OPERAND-L2 = *NO) or an operand in the path leading to this operand value (OPERAND-L2 ≠ *NO).

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING/ *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME/ *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN/ *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION/ *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
Specifies the operand value whose definition is to be compared (OPERAND-L2=*NO) or an operand value, introducing a structure, in the path leading to it (OPERAND-L2≠*NO). If VALUE-L1 is of data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE= <c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that the definition of VALUE-L1 is to be compared. Otherwise, the name of the operand to which the operand value whose definition is to be compared pertains (OPERAND-L3 = *NO) or the name of an operand in the path leading to this operand value (OPERAND-L3 ≠ *NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L1.

VALUE-L2 = *NO/*COMMAND-REST/*INTEGER/*X-STRING/*C-STRING/ *NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/*FILENAME/ *PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/ *COMPOSED-NAME/*X-TEXT/*FIXED/*DEVICE/*PRODUCT-VERSION/ *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30> /
*NO means that the VALUE-L2 is irrelevant for the specification of the operand value to be compared. Otherwise, an operand value is to be specified. This operand value is either the one whose definition is to be compared (OPERAND-L3=*NO) or an operand value introducing a structure in the path leading to it (OPERAND-L3≠*NO). For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the definition of the operand value to be compared. Otherwise, the name of the operand to which the operand value whose definition is to be compared pertains (OPERAND-L4=*NO) or the name of an operand in the path leading to this operand value (OPERAND-L4≠*NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L2.

VALUE-L3 = analogous to VALUE-L2
*NO means that VALUE-L3 is irrelevant for the specification of the definition of the operand value to be compared. Otherwise an operand value is to be specified. This operand value is either the one whose definition is to be compared (OPERAND-L4 = *NO) or an operand value, introducing a structure, in the path leading to it (OPERAND-L4≠*NO). For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L4 = analogous to VALUE-2
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L5= analogous to VALUE-2
see VALUE-L2.

ORIGIN =
Specifies the command or statement to which the operand value definition to be compared pertains.

ORIGIN = *COMMAND(...)
The operand value definition pertains to a command.

NAME = <structured-name 1..30>
Name of the command.

ORIGIN = *STATEMENT(...)
The operand value definition pertains to a statement.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

INPUT-FILE = <filename 1..54 without-gen-vers>(...)
Syntax file containing the definitions or global information to be compared.

TYPE = *SYSTEM / *GROUP / *USER
The type of syntax file containing the definitions or the global information to be compared: system syntax file, group syntax file or user syntax file.

COMPARE-FILE = <filename 1..54 without-gen-vers>(...)
Syntax file with which the definitions or global information from INPUT-FILE are to be compared.

TYPE = *SYSTEM / *GROUP / *USER
Type of syntax file:
system syntax file, group syntax file or user syntax file.

ATTACHED-INFO =
Specifies which of the definitions pertaining to the particular object are to be output.

ATTACHED-INFO = <u>*YES</u>
The definition of the particular object is output, along with the definitions of all objects associated with the particular object (in other words: domain with associated commands, program with associated statements, command or statement with associated operand values, operand value with associated structure, global information with language-dependent texts).

ATTACHED-INFO = *NO
The definition of the specified object is output without the definitions of the objects associated with the specified object (in other words: domain without associated commands, program without associated statements, command or statement without associated operands, operand without associated operand values, operand value without associated structure, global information without language-dependent texts).

OUTPUT =
Specifies the output medium for the comparison log.

OUTPUT = *SYSOUT
Output is directed to the logical system file SYSOUT, i.e. in interactive mode usually to the screen.

OUTPUT = *SYSLST(...)
Output is directed to the logical system file SYSLST.

SYSLST-NUMBER = *STD / <integer 1..99>
Specifies the number of the logical system file SYSLST. If *STD is specified, the logical system file SYSLST is not numbered.

COMPARE-ATTRIBUTES =
Specifies the comparison attributes.

COMPARE-ATTRIBUTES = <u>*ALL</u>
The complete object definition is compared.

COMPARE-ATTRIBUTES = *USER-INTERFACE
The following are compared:
– the help texts of the command, the statement, the domain or the operand
– the additions to the data type of an operand value (e.g. <integer 1..99>)
– the commentary line assigned to a program.

COMPARE-ATTRIBUTES = *PRIVILEGES-ALLOWNESS
The following are compared:
– the input modes (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED etc.)
– the privileges which are assigned to the object.

Outputs of the COMPARE-SYNTAX-FILE statement

If the definitions of the objects to be compared are identical, only the names of the two syntax files and then the message `END OF COMPARISON` are output.

If the definitions of the objects to be compared are not identical, SDF-A outputs the following information:

- Name of both syntax files
- Identification of the objects which are defined differently
- Differences in the definition

Names of the two syntax files

The figure in the first column identifies the syntax file:

1 : Syntax file which was specified with INPUT-FILE
2 : Syntax file which was specified with COMPARE-FILE

If the digit in the first column is missing, the information following it is valid for both syntax files.

Identifying the objects which are defined differently

SDF-A searches for objects with the external name in syntax file 1 (INPUT-FILE). If the name exists, SDF-A searches for this object by its internal name in syntax file 2 (COMPARE-FILE).

The identifier in the comparison log always begins with the object type, followed by a colon and the exact path to the object. The meanings of the terms in the following text are:

<domain-name>	Name of the domain
<command-name>	First STANDARD-NAME of the command (see ADD-CMD, page 133)
<program-name>	Name of the program
<statement-name>	First STANDARD-NAME of the statement (see ADD-STMT, page 160)
<operand-name>	First STANDARD-NAME of the operand (see ADD-OPERAND, page 148)
<value-name>	First STANDARD-NAME of the operand value, if it is of the data type KEYWORD(-NUMBER); otherwise the data type of the operand value in lowercase letters.

DOMAIN:<domain-name>
 Differences in the definition of a domain.

COMMAND:<domain-name>.<command-name>
 Differences in the command definition, and the comparison was carried out using OBJECT=*DOMAIN(...).

COMMAND:<command-name>
 Differences in the command definition, and the comparison was carried out using OBJECT=*COMMAND(...).

PROGRAM:<program-name>
 Differences in the program definition.

STATEMENT:<program-name>.<statement-name>
 Differences in the statement definition.

OPERAND:<domain-name>.<command-name>.<operand-name>
 Differences in the operand definition of a command, and the comparison was carried out using OBJECT=*DOMAIN(...).

OPERAND:<command-name>.<operand-name>
 Differences in the operand definition of a command, and the comparison was carried out using OBJECT=*COMMAND(...).

OPERAND:<program-name>.<statement-name>.<operand-name>
 Differences in the operand definition of a statement and the comparison was carried out using OBJECT=*PROGRAM(...).

VALUE:<domain-name>.<command-name>.<operand-name>.<value-name>
 Differences in the operand value definition of a command, and the comparison was carried out using OBJECT=*DOMAIN(...).

VALUE:<command-name>.<operand-name>.<value-name>
 Differences in the operand value definition of a command, and the comparison was carried out using OBJECT=*COMMAND(...).

VALUE:<program-name>.<statement-name>.<operand-name>.<value-name>
 Differences in the operand value definition of a command, and the comparison was carried out using OBJECT=*PROGRAM(...).

Differences between the compared objects

Directly after the object identifier, the results of the comparison are output.
Lines without the digit 1 or 2 in the first column are relevant to both syntax files and contain, for example, the text 'NOT DEFINED' as an indication that the object to be defined is not defined in both syntax files. The following are examples of output lines in a comparison log and with a short explanation.

Examples of undefined objects

```
PROGRAM NOT DEFINED
```

No program is defined in either of the syntax files. OBJECT=*PROGRAM(NAME=*ALL) was specified for the comparison.

```
PROGRAM:SDF-A NOT DEFINED
```

The program SDF-A is defined neither in syntax file 1 (INPUT-FILE) nor in syntax file 2 (COMPARE-FILE).

```
1 OPERAND:MODIFY-SDF-OPTIONS.MODE.TEST.CHECK-PRIVILEGES NOT DEFINED
```

In syntax file 1, the CHECK-PRIVILEGE operand is not defined in the structure MODE=*TEST(...) in the command MODIFY-SDF-OPTIONS.

```
2 COMMAND:SDF.RESET-INPUT-DEFAULTS NOT DEFINED
```

In syntax file 2, the RESET-INPUT-DEFAULTS command is not defined in the domain SDF.

```
1 COMMAND:MODIFY-SDF-OPTIONS NOT DEFINED
2 COMMAND:MODIFY-SDF-OPTIONS NOT DEFINED
```

The MODIFY-SDF-OPTIONS command does not have the same internal name in syntax file 2 as in syntax file 1.

Examples of different objects

Differences are presented in the form of the part of the SDF-A statement with which the effected object was defined; e.g. ADD-CMD for differences between commands, ADD-DOMAIN for differences between domains, or SET-GLOBALS for different global information.

```
1 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from input-file'))
2 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from compare-file'))
```

The help texts for the MODIFY-SDF-OPTIONS command are different.
The difference is presented as a part of the statement ADD-CMD NAME=MODIFY-SDF-OPTIONS,...,HELP=(E('...')).

```
1 COMMAND:MODIFY-SDF-OPTIONS HELP=(E('help from compare-file')) NOT DEFINED
```

There is no English help text defined in syntax file 1 for the MODIFY-SDF-OPTIONS command (the help text 'help from compare-file' is from syntax file 2 and is missing in syntax file 1).

```
1 VALUE:SDF.MODIFY-SDF-OPTIONS.MODE.TEST STRUCTURE=YES
2 VALUE:SDF.MODIFY-SDF-OPTIONS.MODE.TEST STRUCTURE=NO
```

The TEST value in the MODE operand of the MODIFY-SDF-OPTIONS command (SDF domain) is defined in syntax file 1 with STRUCTURE=*YES, and in syntax file 2 with STRUCTURE=*NO.

COPY
Copy contents of syntax file

The COPY statement is used to copy the contents of a syntax file. Information on the locking of objects (see REMOVE) is not copied. SDF-A inserts the copies into the syntax file being processed.

Definitions of BS2000 commands (implemented via system modules) may be copied into a user syntax file only when they are fully covered by the reference syntax files assigned (see OPEN-SYNTAX-FILE). The same applies to the case where an operand or operand value definition is being copied into the definition of a command implemented via system modules.

If the global information is copied, SDF-A overwrites the existing global information in the syntax file being processed.

The SDF default statements neither can nor need to be copied into the user syntax file. SDF provides these statements automatically for every program which has statements defined in syntax files. All standard statements released by Fujitsu Siemens Computers are stored centrally in the syntax file of SDF as of V4.0A.

Before the definition of an operand or operand value is copied, it must be ensured that the object is the current one (e.g. another operand) after whose definition the copy is to be inserted into a command or statement definition (see ADD-OPERAND and ADD-VALUE). The operand or operand value must not yet have been defined in this environment. Otherwise, SDF-A refuses to perform the copy operation and issues an appropriate message.

The SHOW-SYNTAX-VERSIONS command supplied with earlier versions of SDF up to V3.0A cannot be copied to a user syntax file with COPY OBJECT=*COMMAND (NAME=*ALL). As of SDF V4.0A, the SHOW-SYNTAX-VERSIONS command is treated like any other command and can thus be copied to a user syntax file.

Certain operand values are context-sensitive. For example, a value defined with TYPE=*INTEGER(OUT-FORM=*STD) can be converted in different ways, depending on the implementation form:

– If a command is defined with IMPLEMENTOR=*PROCEDURE or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...), OUT-FORM=*STD is converted to OUT-FORM=*CHAR.

– If a command is defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...), OUT-FORM=*STD is converted to OUT-FORM=*BINARY.

If an object of this nature is copied from one context (e.g. ...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) to another (e.g. ...,CMD-INTERFACE=*STRING,...), the user must adapt it to the context. This entails using the EDIT statement to position to the object in question and using the MODIFY statement to modify the object.

(part 1 of 3)

```
COPY

OBJECT = *GLOBAL-INFORMATION / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) /
            *STATEMENT(...) / *OPERAND(...) / *VALUE(...)

  *DOMAIN(...)

  |   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
  |     *ALL(...)
  |     |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
  |     |               list-poss(2000): <structured-name 1..30>

  *COMMAND(...)

  |   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
  |     *ALL(...)
  |     |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
  |     |               list-poss(2000): <structured-name 1..30>

  *PROGRAM(...)

  |   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
  |     *ALL(...)
  |     |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
  |     |               list-poss(2000): <structured-name 1..30>

  *STATEMENT(...)

  |   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
  |     *ALL(...)
  |     |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
  |     |               list-poss(2000): <structured-name 1..30>
  |  ,PROGRAM = <structured-name 1..30>

  *OPERAND(...)

  |   OPERAND-L1 = *CURRENT / <structured-name 1..20>
  |  ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
  |               *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
  |               *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
  |               *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
  |               *POSIX-PATHNAME / *POSIX-FILENAME /
  |               <composed-name 1..30>
  |  ,OPERAND-L2 = *NO / <structured-name 1..20>
```

continued ➠

(part 2 of 3)

```
  ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                   *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                   *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                   *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                   *POSIX-PATHNAME / *POSIX-FILENAME /
                   <composed-name 1..30>

  ,OPERAND-L3 = *NO / <structured-name 1..20>

  ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                   *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                   *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                   *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                   *POSIX-PATHNAME / *POSIX-FILENAME /
                   <composed-name 1..30>

  ,OPERAND-L4 = *NO / <structured-name 1..20>

  ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                   *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                   *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                   *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                   *POSIX-PATHNAME / *POSIX-FILENAME /
                   <composed-name 1..30>

  ,OPERAND-L5 = *NO / <structured-name 1..20>

  ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)

     *COMMAND(...)
        NAME = <structured-name 1..30>

     *STATEMENT(...)
        NAME = <structured-name 1..30>

       ,PROGRAM = <structured-name 1..30>

*VALUE(...)

   OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>

  ,VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                   *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                   *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                   *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                   *POSIX-PATHNAME / *POSIX-FILENAME /
                   <composed-name 1..30>

  ,OPERAND-L2 = *NO / <structured-name 1..20>
```

continued ➠

(part 3 of 3)

,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L3 = *NO / <structured-name 1..20>

,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L4 = *NO / <structured-name 1..20>

,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L5 = *NO / <structured-name 1..20>

,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)

*COMMAND(...)
| NAME = <structured-name 1..30>

*STATEMENT(...)
| NAME = <structured-name 1..30>
| ,PROGRAM = <structured-name 1..30>

,FROM-FILE = *CURRENT / *TASK-HIERARCHY / <filename 1..54 without-gen-vers>(...)

<filename 1..54 without-gen-vers>(...)
| TYPE = *CURRENT / *USER / *GROUP / *SYSTEM

,ATTACHED-INFO = *YES / *NO / *ONLY

,OVERWRITE-POSSIBLE = *NO / *YES / *EXTERNAL-ATTRIBUTES

OBJECT =
Type of the object whose definition is to be copied.

OBJECT = *GLOBAL-INFORMATION
Specifies that the global information of a syntax file is to be copied.

OBJECT = *DOMAIN(...)
Specifies that the definitions of domains are to be copied.

NAME = *ALL(...)
The definitions of all domains are copied.

EXCEPT = *NONE / <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the domains specified here are not copied.

NAME = <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the named domains or of the domains matching the wildcard selector are copied.

OBJECT = *COMMAND(...)
Specifies that the definitions of commands are to be copied.

NAME = *ALL
The definitions of all commands are copied.

EXCEPT = *NONE / <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the commands specified here are not copied.

NAME = <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the named commands or of the commands matching the wildcard selector are copied.

OBJECT = *PROGRAM(...)
Specifies that the definitions of programs are to be copied.

NAME = *ALL(...)
The definitions of all programs are copied.

EXCEPT = *NONE / <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the programs specified here are not copied.

NAME = <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the named programs or of the programs matching the wildcard selector are copied.

OBJECT = *STATEMENT(...)
Specifies that the definitions of statements are to be copied.

NAME = <u>*ALL(...)</u>
The definitions of all statements are copied.

EXCEPT = <u>*NONE</u> / <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the statements specified here are not copied.

NAME = <structured-name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the named statements or of the statements matching the wildcard selector are copied.

PROGRAM = <structured-name 1..30>
Name of the program to which the statements pertain. The program must have already been defined in the open syntax file.

OBJECT = *OPERAND(...)
Specifies that the definition of an operand is to be copied. If this operand is included in a structure, it is specified by the path leading to it, i.e. by specifying the operands and operand values that introduce the structure in this path. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified (it may even be omitted). An operand which is not absolutely essential to identify the operand definition to be copied, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first VALUE-Li = *NO, SDF-A takes the operand defined by OPERAND-Li as the one whose definition is to be copied. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for VALUE-Li, the value specified for OPERAND-Li + 1 must also be other than *NO.

OPERAND-L1 = <u>*CURRENT</u> / <structured-name 1..20>
Specifies the operand whose definition is to be copied (VALUE-L1 = *NO) or an operand in the path leading to it (VALUE-L1≠*NO). *CURRENT means that OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING/ *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME/ *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN/ *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30> /
*NO means that the definition of OPERAND-L1 is to be copied. Otherwise, an operand value that introduces a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be copied. If the operand value introducing the structure is of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE= <c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value introducing the structure is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that OPERAND-L2 is irrelevant for the specification of the operand whose definition is to be copied. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L1 is to be specified. This operand is either the one whose definition is to be copied (VALUE-L2=*NO) or one that is in the path leading to it (VALUE-L2≠*NO).

VALUE-L2 = analogous to VALUE-L1
*NO means that VALUE-L2 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be copied. For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand whose definition is to be copied. Otherwise the name of an operand that is unique within the structure determined by VALUE-L2 is to be specified. This operand is either the one whose definition is to be copied (VALUE-L3=*NO) or one that is in the path leading to it (VALUE-L3≠*NO).

VALUE-L3= analogous to VALUE-L1
*NO means that VALUE-L3 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be copied. For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L2.

VALUE-L4 = analogous to VALUE-L1
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L2.

ORIGIN =
Specifies the command or statement to which the operand definition to be copied belongs.

ORIGIN = *CURRENT
The operand definition belongs to the same command (or statement) into which SDF-A is to insert the copy.

ORIGIN = *COMMAND (...)
The operand definition belongs to a command.

NAME = <structured-name 1..30>
Name of the command.

ORIGIN = *STATEMENT(...)
The operand definition belongs to a statement.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *VALUE(...)
Specifies that the definition of operand value is to be copied. This operand value is specified by the path leading to it, i.e. by specifying the operands and operand values introducing the structure in this path. If the operand value pertains to an operand that is not attached to any structure, the path contains only this operand. If the operand value does pertain to an operand attached to a structure, the path also includes the higher-ranking operands as well as the associated operand values introducing the structure. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified. An operand that is not absolutely essential to identify the operand value definition to be copied, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first OPERAND-Li+1 = *NO, SDF-A takes the operand value defined by VALUE-Li as the one whose definition is to be copied. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for OPERAND-Li, the value specified for VALUE-Li must likewise be other than *NO.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Specifies the operand to which the operand value whose definition is to be copied pertains (OPERAND-L2 = *NO) or an operand in the path leading to this operand value (OPERAND-L2≠*NO).

*ABOVE-CURRENT means that a value pertaining to OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1=*CURRENT / *COMMAND-REST / *INTEGER / *X-STRING/ *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME/ *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL/*VSN/*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION/
***POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>**
Specifies the operand value whose definition is to be copied (OPERAND-L2=*NO) or an operand value, introducing a structure, in the path leading to it (OPERAND-L2≠*NO). *CURRENT means that VALUE-L1 is the current object. If it is not the current object and of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=KEYWORD,...,VALUE=<c-string>).
Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that the definition of VALUE-L1 is to be copied. Otherwise, the name of the operand to which the operand value whose definition is to be copied pertains (OPERAND-L3=*NO) or the name of an operand in the path leading to this operand value (OPERAND-L3≠*NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L1.

VALUE-L2 = *NO/*COMMAND-REST/*INTEGER/*X-STRING/*C-STRING/ *NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/*FILENAME/ *PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/ *COMPOSED-NAME/*X-TEXT/*FIXED/*DEVICE/*PRODUCT-VERSION/ *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO means that the VALUE-L2 is irrelevant for the specification of the operand value to be copied. Otherwise, an operand value is to be specified. This operand value is either the one whose definition is to be copied (OPERAND-L3=*NO) or an operand value introducing a structure in the path leading to it (OPERAND-L3≠*NO). For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the definition of the operand value to be copied. Otherwise, the name of the operand to which the operand value whose definition is to be copied pertains (OPERAND-L4=*NO) or the name of an operand in the path leading to this operand value (OPERAND-L4≠*NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L2.

VALUE-L3 = analogous to VALUE-L2
*NO means that VALUE-L3 is irrelevant for the specification of the definition of the operand value to be copied. Otherwise an operand value is to be specified. This operand value is either the one whose definition is to be copied (OPERAND-L4=*NO) or an operand value, introducing a structure, in the path leading to it (OPERAND-L4≠*NO). For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L4 = analogous to VALUE-2
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L5= analogous to VALUE-2
see VALUE-L2.

ORIGIN =
Specifies the command or statement to which the operand value definition to be copied pertains.

ORIGIN = *CURRENT
The operand value definition to be copied pertains to the same command (or statement) into which SDF-A is to insert the copy.

ORIGIN = *COMMAND(...)
The operand value definition pertains to a command.

NAME = <structured-name 1..30>
Name of the command.

ORIGIN = *STATEMENT(...)
The operand value definition pertains to a statement.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

FROM-FILE =
Syntax file containing the definitions or global information to be copied.

FROM-FILE = *CURRENT
The syntax file currently being processed contains the definitions to be copied. This is possible only when copying definitions of operands or operand values.

FROM-FILE = *TASK-HIERARCHY
The definition to be copied is taken from one of the syntax files which belong to the current syntax file hierarchy.

FROM-FILE = <filename 1..54 without-gen-vers>(...)
The syntax file named contains the definitions or global information to be copied.

TYPE = *CURRENT / *USER / *GROUP / *SYSTEM
The syntax file containing the definitions or the global information to be copied is

*CURRENT	of the same type as the syntax file being processed
*USER	a user syntax file
*GROUP	a group syntax file
*SYSTEM	a system syntax file.

ATTACHED-INFO =
Specifies which of the definitions pertaining to the specified object is to be copied. For global information, programs and domains, SDF-A interprets the value ONLY as *YES.

ATTACHED-INFO = *YES
The definition of the specified object is copied, along with the definitions of all objects associated with the specified object (in other words: domain with associated commands, program with associated statements, command or statement with associated operand values, operand value with associated structure, global information with language-dependent texts).

ATTACHED-INFO = *NO
The definition of the specified object is copied without the definitions of the objects associated with the specified object (in other words: domain without associated commands, program without associated statements, command or statement without associated operands, operand without associated operand values, operand value without associated structure, global information without language-dependent texts).

ATTACHED-INFO = *ONLY
Only the definitions of objects associated with the specified object are copied. The definition of the specified object itself is not copied (examples: operand values without the operands to which they pertain; a structure without the operand value introducing it).

OVERWRITE-POSSIBLE =
Specifies whether the definition of a domain, command, program or statement is to be copied if the object is already defined in the open syntax file.

OVERWRITE-POSSIBLE = *NO
SDF-A rejects the copying of a domain, command, program or statement (and issues an appropriate message) if the object has already been defined in the syntax file being processed. Copying of global information is possible.

OVERWRITE-POSSIBLE = *YES
SDF-A performs the copy operation regardless of whether the object is already defined in the open syntax file. In this case, SDF-A replaces the existing definition in the open syntax file by the definition to be copied. A command or statement can be overwritten only if:
– NAMEs or STANDARD-NAMEs are identical and
– INTERNAL-NAMEs are identical.

OVERWRITE-POSSIBLE = *EXTERNAL-ATTRIBUTES
SDF-A copies the objects only, without the definitions of the objects themselves. The definitions of the objects which follow the overwritten domain or the program (e.g. commands or statements) remain the same. Specification of this operand is only possible when copying domains and programs (COPY OBJ=*DOMAIN... or OBJ=*PROGRAM...). The ATTACHED-INFO operand is given the value *NO, regardless of whether or not the user has specified a different value.

DEFINE-ENVIRONMENT
Define format and version of syntax file

The DEFINE-ENVIRONMENT statement defines the SDF-A version to be used and thus the syntax file format in which the processed syntax file is stored. This statement must be explicitly executed before creating or opening a syntax file whenever the user does not wish to work with the highest SDF-A version. Note that syntax files cannot be processed with a lower version than the one with which they were created or stored.

DEFINE-**ENVIRON**MENT
SYNTAX-FILE-FORMAT = <u>***CURRENT**</u> / ***V4.1** / ***V4** / ***V3**

SYNTAX-FILE-FORMAT =
Defines the SDF-A version to be used.

SYNTAX-FILE-FORMAT = <u>*CURRENT</u> / *V4.1
SDF-A V4.1 remains loaded, so subsequently processed syntax files are stored in V4.1 format.

SYNTAX-FILE-FORMAT = *V4
SDF-A V4.0 is loaded, and the complete functionality of SDF-A V4.0 is available to the user. Subsequently processed syntax files are stored in V4 format.

SYNTAX-FILE-FORMAT = *V3
SDF-A V3.0 is loaded, and the complete functionality of SDF-A V3.0 is available to the user. Subsequently processed syntax files are stored in V3 format.

EDIT
Position to object in syntax file

The EDIT statement is used to declare a domain, a program, a command, a statement, an operand or an operand value, or the global information of the syntax file as the “current object” (see page 148). EDIT can also be used to position to an object that is not in the open syntax file but in the group or system syntax file specified with the operand GROUP-DESCRIPTION or SYSTEM-DESCRIPTION when the syntax file being processed was opened.

When the definition of an object is to be modified using a MODIFY statement, the user must first make sure that the object is the “current” object. Likewise, if the definition of an operand or operand value is to be inserted into the definition of a command or statement by means of an ADD or COPY statement, it must be ensured that the object after whose definition the insertion is to be made is the “current” object. When (in guided dialog) the global information is modified using the SET-GLOBALS statement, the current values are displayed only if the global information is the “current” object.

(part 1 of 4)

```
EDIT

OBJECT = *GLOBAL-INFORMATION / *PRIVILEGE(...) / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) /
             *STATEMENT(...) / *OPERAND(...) / *VALUE(...)

  *PRIVILEGE(...)
     |  NAME = <structured-name 1..30>
  *DOMAIN(...)
     |  NAME = <structured-name 1..30>
  *COMMAND(...)
     |  NAME = <structured-name 1..30>
  *PROGRAM(...)
     |  NAME = <structured-name 1..30>
  *STATEMENT(...)
     |  NAME = <structured-name 1..30>
     | ,PROGRAM = <structured-name 1..30>
```

continued ➠

(part 2 of 4)

```
*OPERAND(...)
 |
 |   OPERAND-L1 = *CURRENT / <structured-name 1..20>
 |
 |  ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
 |                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
 |                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
 |                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
 |                *POSIX-PATHNAME / *POSIX-FILENAME /
 |                <composed-name 1..30>
 |
 |  ,OPERAND-L2 = *NO / <structured-name 1..20>
 |
 |  ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
 |                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
 |                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
 |                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
 |                *POSIX-PATHNAME / *POSIX-FILENAME /
 |                <composed-name 1..30>
 |
 |  ,OPERAND-L3 = *NO / <structured-name 1..20>
 |
 |  ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
 |                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
 |                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
 |                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
 |                *POSIX-PATHNAME / *POSIX-FILENAME /
 |                <composed-name 1..30>
 |
 |  ,OPERAND-L4 = *NO / <structured-name 1..20>
 |
 |  ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
 |                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
 |                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
 |                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
 |                *POSIX-PATHNAME / *POSIX-FILENAME /
 |                <composed-name 1..30>
 |
 |  ,OPERAND-L5 = *NO / <structured-name 1..20>
 |
 |  ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
 |
 |     *COMMAND(...)
 |      |  NAME = <structured-name 1..30>
 |
 |     *STATEMENT(...)
 |      |  NAME = <structured-name 1..30>
 |      |
 |      | ,PROGRAM = <structured-name 1..30>
```

continued ➠

(part 3 of 4)

*VALUE(...)

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>

,VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L2 = *NO / <structured-name 1..20>

,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L3 = *NO / <structured-name 1..20>

,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L4 = *NO / <structured-name 1..20>

,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

,OPERAND-L5 = *NO / <structured-name 1..20>

,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
*ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
*PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
*COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
*POSIX-PATHNAME / *POSIX-FILENAME /
<composed-name 1..30>

continued ➠

(part 4 of 4)

```
      ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
          *COMMAND(...)
               NAME = <structured-name 1..30>
          *STATEMENT(...)
               NAME = <structured-name 1..30>
              ,PROGRAM = <structured-name 1..30>
```

OBJECT =
Type of the object being declared the current object.

OBJECT = *GLOBAL-INFORMATION
The global information of the syntax file becomes the current object.

OBJECT = *PRIVILEGE(...)
The specified privilege becomes the current object. This operand is reserved for Fujitsu Siemens Computers Software Development.

NAME = <structured-name 1..30>
Name of the privilege.

OBJECT = *DOMAIN(...)
A domain becomes the current object.

NAME = <structured-name 1..30>
Name of the domain.

OBJECT = *COMMAND(...)
A command becomes the current object.

NAME = <structured-name 1..30>
Name of the command.

OBJECT = *PROGRAM(...)
A program becomes the current object.

NAME = <structured-name 1..30>
Name of the program.

OBJECT = *STATEMENT(...)
A statement becomes the current object.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *OPERAND(...)
An operand becomes the current object. If the operand that becomes the current object is included in a structure, it is specified by the path leading to it, i.e. by specifying the operands and operand values that introduce the structure in this path. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified (and may even be omitted). An operand that is not absolutely essential to identify the operand that becomes the current object, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first VALUE-Li=*NO, SDF-A takes the operand defined by OPERAND-Li as the one that becomes the current object. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for VALUE-Li, the value specified for OPERAND-Li + 1 must also be other than *NO.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Specifies the operand that becomes the current object (VALUE-L1=*NO) or an operand in the path leading to it (VALUE-L1≠*NO). *CURRENT means that OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *NO/*COMMAND-REST/*INTEGER/*X-STRING/*C-STRING/ *NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/*FILENAME/ *PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/ *COMPOSED-NAME/*X-TEXT/*FIXED/*DEVICE/*PRODUCT-VERSION/ *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO means that OPERAND-L1 becomes the current object. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand that becomes the current object. If the operand value introducing the structure is of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=KEYWORD,...,VALUE= <c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value introducing the structure is not of the type KEYWORD(-NUMBER), the data defined for it must be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that OPERAND-L2 is irrelevant for the specification of the operand that becomes the current object. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L1 is to be specified. This operand either becomes the current object (VALUE-L2 = *NO) or is in the path leading to the operand that becomes the current object (VALUE-L2 ≠ *NO).

VALUE-L2 = analogous to VALUE-L1
*NO means that VALUE-L2 is irrelevant for the specification of the operand that becomes the current object. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand that becomes the current object. For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand that becomes the current object. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L2 is to be specified. This operand either becomes the current object (VALUE-L3 = *NO) or is in the path leading to the operand that becomes the current object (VALUE-L3 ≠ *NO).

VALUE-L3 = analogous to VALUE-L1
*NO means that VALUE-L3 is irrelevant for the specification of the operand that becomes the current object. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand that becomes the current object. For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L2.

VALUE-L4 = analogous to VALUE-L1
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L2.

ORIGIN =
Specifies the command or statement in which the specified operand becomes the current object.

ORIGIN = *CURRENT
The operand belongs to the command (or statement) which at the present time either is itself the current object or contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)
The operand belongs to a command.

> **NAME = <structured-name 1..30>**
> Name of the command.

ORIGIN = *STATEMENT(...)
The operand belongs to a statement.

> **NAME = <structured-name 1..30>**
> Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *VALUE(...)
An operand value becomes the current object. The operand value that becomes the current object is specified by the path leading to it, i.e. by specifying the operands and operand values introducing the structure in this path. If the operand value that becomes the current object pertains to an operand that is not attached to any structure, the path contains only this operand. If the operand value that becomes the current object does pertain to an operand attached to a structure, the path also includes the higher-ranking operands as well as the associated operand values introducing the structure. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified. An operand that is not absolutely essential to identify the operand value that becomes the current object, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first OPERAND-Li + 1 = *NO, SDF-A takes the operand value defined by VALUE-Li as the one that becomes the current object. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for OPERAND-Li, the value specified for VALUE-Li must likewise be other than *NO.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Specifies the operand to which the operand value that becomes the current object pertains (OPERAND-L2 = *NO) or an operand in the path leading to this operand value (OPERAND-L2 ≠ *NO). *ABOVE-CURRENT means that a value pertaining to OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

**VALUE-L1 = *CURRENT/*COMMAND-REST/*INTEGER/*X-STRING/
*C-STRING/*NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/
*FILENAME/*PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/*COMPOSED-NAME/*X-TEXT/*FIXED/*DEVICE/*PRODUCT-VERSION/
*POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>**
Specifies the operand value that becomes the current object (OPERAND-L2=*NO) or an operand value introducing a structure in the path leading to the operand value that becomes the current object (OPERAND-L2≠*NO). *CURRENT means that VALUE-L1 is the current object. If VALUE-L1 is not the current object and of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that VALUE-L1 becomes the current object. Otherwise, the name of the operand to which the operand value that becomes the current object pertains (OPERAND-L3 = *NO) or the name of an operand in the path leading to this operand value (OPERAND-L3 ≠ *NO) must be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L1.

VALUE-L2 = *NO/*COMMAND-REST/*INTEGER/*X-STRING/*C-STRING/ *NAME/*ALPHANUMERIC-NAME/*STRUCTURED-NAME/*FULL-FILENAME/ *PARTIAL-FILENAME/*TIME/*DATE/*TEXT/*CAT-ID/*LABEL/*VSN/ *COMPOSED-NAME/*X-TEXT/*FIXED/*DEVICE/*PRODUCT-VERSION/ *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO means that the VALUE-L2 is irrelevant for the specification of the operand value that becomes the current object. Otherwise, an operand value is to be specified. This operand value either becomes the current object (OPERAND-L3 = *NO) or is an operand value introducing a structure in the path leading to the operand value that becomes the current object (OPERAND-L3 ≠ *NO). For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand value that becomes the current object. Otherwise, the name of the operand to which the operand value that becomes the current object pertains (OPERAND-L4 = *NO) or the name of an operand in the path leading to this operand value (OPERAND-L4 ≠ *NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L2.

VALUE-L3 = analogous to VALUE-L2
*NO means that the VALUE-L3 is irrelevant for the specification of the operand value that becomes the current object. Otherwise, an operand value is to be specified. This operand value either becomes the current object (OPERAND-L4 = *NO) or is an operand value introducing a structure in the path leading to the operand value that becomes the current object (OPERAND-L4 ≠ *NO). For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L4 = analogous to VALUE-2
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L5 = analogous to VALUE-2
see VALUE L2.

ORIGIN =
Specifies the command or statement in which the specified operand value becomes the current object.

ORIGIN = <u>*CURRENT</u>
The operand value belongs to the command (or statement) which at the present time either is itself the current object or contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)
The operand value belongs to a command.

> **NAME=<structured-name 1..30>**
> Name of the command.

ORIGIN = *STATEMENT(...)
The operand value belongs to a statement.

> **NAME = <structured-name 1..30>**
> Name of the statement.
>
> **PROGRAM = <structured-name 1..30>**
> Name of the program to which the statement pertains.

END
Terminate program run

The END statement is used to terminate input to SDF-A and implicitly closes the most recently opened syntax file.

END

This statement has no operands.

MODIFY-CMD
Modify command definition

The MODIFY-CMD statement is used to modify the definition of a command in the open syntax file. The command must be the “current” object (see page page 148). Afterwards the first operand of the command is the “current” object.

The MODIFY-CMD statement is very similar to the ADD-CMD statement. The default value for all operands of the MODIFY-CMD statement is *UNCHANGED, i.e. only those parts of a command definition that are explicitly specified are modified. In guided dialog the operand form displays the current operand assignment instead of *UNCHANGED. When a value other than *UNCHANGED is specified for a MODIFY-CMD operand, the old specifications in the command definition are replaced by the new ones. This also applies if a list can be specified, i.e. instead of being added to the value list, the specified value replaces it. There is a special rule for STANDARD-NAME. When the command to be modified is defined in an assigned reference file (see OPEN-SYNTAX-FILE), the old entries for STANDARD-NAME are retained and the name specified with MODIFY-CMD is added to them.

With the exception of the RESULT-INTERNAL-NAME, all names given to the command must be unique with respect to the complete set of commands.

Definitions of the operands and operand values pertaining to the command are modified by means of the MODIFY-OPERAND or MODIFY-VALUE statement, rather than by means of the MODIFY-CMD statement.

(part 1 of 4)

MODIFY-**CMD**
NAME = ***UNCH**ANGED / <structured-name 1..30>
,**RES**ULT-**INT**ERN**AL**-**NAME** = ***UNCH**ANGED / ***SAME** / <alphanum-name 1..8>
,**STAND**ARD-**NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): ***NAME** / <structured-name 1..30>
,**ALIAS-NAME** = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..30>
,**MIN**IMAL-**ABBREV**IATION = ***UNCH**ANGED / ***NO** / <structured-name 1..30>
,**HELP** = ***UNCH**ANGED / ***NO** / list-poss(2000): <name 1..1>(...)
<name 1..1>(...)
| **TEXT** = ***UNCH**ANGED / <c-string 1..500 with-low>
,**DOM**AIN = ***UNCH**ANGED / ***NO** / list-poss(2000): <structured-name 1..30>

continued ➠

(part 2 of 4)

```
,IMPLEMENTOR = *UNCHANGED / *PROCEDURE(...) / *TPR(...) / *APPLICATION(...) / *BY-TPR(...)

   *PROCEDURE(...)
   |     NAME = *UNCHANGED / <c-string 1..280> / *BY-IMON(...)
   |
   |        *BY-IMON(...)
   |        |     LOGICAL-ID = *UNCHANGED / <filename 1..30 without-cat-user-gen-vers>
   |        |    ,INSTALLATION-UNIT = *UNCHANGED / <text 1..30 without-sep>
   |        |    ,VERSION = *UNCHANGED / *STD / <product-version>
   |        |    ,DEFAULT-PATH-NAME = *UNCHANGED / <filename 1..54>
   |        |    ,ELEMENT = *UNCHANGED / *NONE / <composed-name 1..64>
   |
   |    ,CALL-TYPE = *UNCHANGED / *CALL-PROCEDURE / *INCLUDE-PROCEDURE /
   |                 *ENTER-PROCEDURE
   |
   |    ,CALL-OPTIONS = *UNCHANGED / *NONE / <c-string 1..1800 with-low>
   |
   |    ,UNLOAD-PROGRAM = *UNCHANGED / *NO / *YES

   *TPR(...)
   |     ENTRY = *UNCHANGED / <name 1..8>
   |
   |    ,INTERFACE = *UNCHANGED / *SPL / *ASS / *ISL(...)
   |
   |        *ISL(...)
   |        |     VERSION = *UNCHANGED / <integer 1..2>
   |
   |    ,CMD-INTERFACE = *UNCHANGED / *STRING(...) / *TRANSFER-AREA(...) / *NEW(...)
   |
   |        *STRING(...)
   |        |     OUT-CMD-NAME = *UNCHANGED / *SAME / <structured-name 1..30>
   |
   |        *TRANSFER-AREA(...)
   |        |     MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
   |        |
   |        |    ,CMD-VERSION = *UNCHANGED / *NONE / <integer 1..999>
   |
   |        *NEW(...)
   |        |     MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
   |
   |    ,LOGGING = *UNCHANGED / *BY-SDF / *BY-IMPLEMENTOR
   |
   |    ,INPUT-FORM = *UNCHANGED / *INVARIANT / *STANDARD / *NONE
   |
   |    ,SCI = *UNCHANGED / *YES / *NO

   *APPLICATION(...)
   |     LOGGING = *UNCHANGED / *BY-SDF / *BY-IMPLEMENTOR

   *BY-TPR(...)
   |     TPR-CMD = *UNCHANGED / <structured-name 1..30>
```

continued ➠

(part 3 of 4)

```
,REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO

,DIALOG-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)

   *YES(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)

   *YES(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

,GUIDED-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)

   *YES(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

,BATCH-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)

   *YES(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

,BATCH-PROC-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)

   *YES(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

,CMD-ALLOWED = *UNCHANGED / *NO(...) / *YES(...)

   *NO(...)
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>

   *YES(...)
     |   UNLOAD = *UNCHANGED / *YES / *NO
     |
     |   PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
```

continued ➠

(part 4 of 4)

```
,NEXT-INPUT = *UNCHANGED / *CMD / *STMT / *DATA / *ANY
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
                 list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
     |   EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
   *ADD(...)
     |   ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
   *REMOVE(...)
     |   REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..30>
(External) command name to be specified when the command is entered. In contrast to STANDARD-NAME and ALIAS-NAME, the user may abbreviate this name when entering the command.

RESULT-INTERNAL-NAME = *UNCHANGED / *SAME / <alphanum-name 1..8>
This operand is of significance only for some of the commands implemented via system modules. Implementation via system modules is reserved for Fujitsu Siemens Computers System Software Development. For this reason, the RESULT-INTERNAL-NAME operand is not described here.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30>
Additional external command name, which may be alternatively used when entering the command. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted so long as the command with this name exists in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE). If the original external name given in the documentation for a command is declared to be its standard name, it is thereby ensured that the command can be entered using the original name, regardless of any name changes. Specifying *NAME causes SDF-A to take as STANDARD-NAME the external command name entered for the NAME operand.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Additional external command name, which can be alternatively used when the command is entered. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
Additional external command name which defines the shortest permissible abbreviation for the command. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other commands.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct command but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the command name (NAME).
3. The ALIAS-NAMEs and STANDARD-NAMEs of the command must not be shorter than the minimum abbreviation if they are an abbreviation of the command name.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the command, and if so, what those texts are.

HELP = *UNCHANGED
No change with regard to help texts.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Help text. UNCHANGED is permissible only if there is already a help text defined for the language key.
The help text can contain the special character string “\n” for line breaks.

DOMAIN = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Specifies whether the command is assigned to a domain, and if so, to which one.

IMPLEMENTOR =
Specifies how the command is implemented.

IMPLEMENTOR = *UNCHANGED
No change with regard to the implementation of the command.

IMPLEMENTOR = *PROCEDURE(...)
The command is implemented via a procedure. Entering the command causes the procedure to be called.

NAME = *UNCHANGED / <c-string 1..280> / *BY-IMON(...)
Name of the file containing the procedure.

NAME = <c-string 1..280>
Name of the file containing the procedure. If SDF-P is loaded, the name of a list variable containing the procedure may also be specified. A variable can be specified in the form '*VARIABLE(VARIABLE-NAME=varname)'.
Library elements can be specified with
'*LIBRARY-ELEMENT(LIBRARY=library,ELEMENT=element)'
If 'library(element)' is specified, the value of CALL-OPTIONS is ignored. This notation should therefore no longer be used.

It is the user's responsibility to ensure that the string to specify the container of the procedure is correctly constructed. Errors made at this position can only be detected when the newly defined command is first called.

NAME = *BY-IMON(...)
The name of the procedure or of the library that contains this procedure as a library element is provided by calling IMON-GPN, the installation path manager (see the "IMON" manual [13]).

LOGICAL-ID = *UNCHANGED / <filename 1..30 without-cat-user-gen-vers>
Logical name of the procedure or of the library that contains this procedure as a library element for implementing the commands, e.g. SYSSPR.

INSTALLATION-UNIT = *UNCHANGED /<text 1..30 without-sep>
Name of the installation unit, e.g. SDF-A.

VERSION = *UNCHANGED / *STD / <product-version>
Version of the installation unit (see also "product-version" on page 15 and page 181).
If *STD is specified, the version selected using the SELECT-PRODUCT-VERSION command is used. If this command has not yet been carried out for the relevant installation unit, the highest version is used.

DEFAULT-PATH-NAME = *UNCHANGED / <filename 1..54>
Full file name of a procedure which is called if IMON-GPN is not available or if LOGICAL-ID, INSTALLATION-UNIT or VERSION is not known in the system. If the procedure is stored in a library element, then the file name specified here designates the library from which the procedure specified in the ELEMENT operand is called.
In the case of other errors, the command is rejected with an error message, i.e. the specified procedure is not called.

ELEMENT = *UNCHANGED / *NONE / <composed-name 1..64>
Specifies if the procedure is stored in a library element.

ELEMENT = <composed-name 1..64>
Name of the library element that contains the procedure. The element name is passed to IMON-GPN.

CALL-TYPE = *UNCHANGED / *CALL-PROCEDURE / *INCLUDE-PROCEDURE / *ENTER-PROCEDURE
Defines whether the command procedure is called with CALL-PROCEDURE, INCLUDE-PROCEDURE or ENTER-PROCEDURE. CALL-PROCEDURE and ENTER-PROCEDURE can be used to call S procedures as well as non-S procedures; INCLUDE-PROCEDURE can only be used to call S procedures (see also the manuals "SDF-P" [12] and "Commands" [4]).

Example:
To call the command procedure with
`CALL-PROCEDURE NAME=*LIB-ELEM(LIBRARY=xxx,ELEMENT=yyy)`
the command must be defined as follows:
```
ADD-CMD ...,IMPLEMENTOR=*PROC(NAME='*LIB-ELEM(LIBRARY=xxx,
ELEMENT=yyy)',CALL-TYPE=*CALL-PROCEDURE...
```

CALL-OPTIONS = *UNCHANGED / *NONE / <c-string 1..1800 with-low>
Specifies a string containing additional operands (e.g. LOGGING) for the procedure call (using CALL-/INCLUDE-/ENTER-PROCEDURE) in the following format:
`CALL-OPTIONS='operandx=valuex,operandy=valuey, ...'` . This character string must not contain the PROCEDURE-PARAMETERS operand of the CALL-/INCLUDE-/ENTER-PROCEDURE command.

UNLOAD-PROGRAM = *UNCHANGED / *YES / *NO
Specifies if a program is to be unloaded when the command defined in the NAME operand is executed in the program via the CMD macro.
The procedure called may not contain any commands defined with CMD-ALLOWED=*YES(UNLOAD=*YES), and in particular no TU program may be called in the procedure.

IMPLEMENTOR = *TPR(...)
The command is implemented via system modules. This alternative for implementing commands is reserved for Fujitsu Siemens Computers System Software Development. For this reason, the structure *TPR(...) is not described here.

IMPLEMENTOR = *APPLICATION(...)
The command is generated by a $CONSOLE application. This option is reserved for system administration. The prerequisites for its use are described in the manual “Introductory Guide to Systems Support” [6]. The required commands CONNECT-CMD-SERVER and DISCONNECT-CMD-SERVER are detailed in the “Commands” [4] manual.

LOGGING = *UNCHANGED / *BY-SDF / *BY-IMPLEMENTOR
The operand is reserved for Fujitsu Siemens Computers System Software Development. For this reason, it is not described here.

IMPLEMENTOR = *BY-TPR(...)
An existing command serves as the basis for the new command.

TPR-CMD = *UNCHANGED / <structured-name 1..30>
Name of a command defined with IMPLEMENTOR=*TPR(...) which is known in the syntax hierarchy (see page 140 for an example).

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Specifies whether the command may be deleted (see REMOVE). If the command has been defined with REMOVE-POSSIBLE=*NO in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE), SDF-A rejects a change to *YES.

DIALOG-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Specifies whether the command is allowed in interactive mode.

DIALOG-ALLOWED = *YES(...)
The command is allowed in interactive mode for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is allowed only for user tasks with exactly the same privileges as those defined in this list.

DIALOG-ALLOWED = *NO(...)
The command is not allowed in interactive mode for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES(...) / *NO(...)
Specifies whether the command is allowed in interactive mode within a procedure.

DIALOG-PROC-ALLOWED = *YES(...)
The command is allowed in interactive mode within a procedure for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is allowed only for user tasks with exactly the same privileges as those defined in this list.

DIALOG-PROC-ALLOWED = *NO(...)
The command is not allowed in interactive mode within a procedure for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE = *UNCHANGED / *ALL / list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

GUIDED-ALLOWED = <u>*UNCHANGED</u> / *YES(...) / *NO(...)
Specifies whether the command is allowed in guided dialog.

GUIDED-ALLOWED = *YES(...)
The command is allowed in guided dialog for all user tasks which have at least one of the privileges specified for PRIVILEGE.

PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>
The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is allowed only for user tasks with exactly the same privileges as those defined in this list.

GUIDED-ALLOWED = *NO(...)
The command is not allowed in guided dialog for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

BATCH-ALLOWED = <u>*UNCHANGED</u> / *YES(...) / *NO(...)
Specifies whether the command is allowed in batch mode.

BATCH-ALLOWED = *YES(...)
The command is allowed in batch mode for all user tasks which have at least one of the privileges specified for PRIVILEGE.

> **PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>**
> The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).
>
> **PRIVILEGE = *SAME**
> The command is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).
>
> **PRIVILEGE = list-poss(64): <structured-name 1..30>**
> The command is allowed only for user tasks with exactly the same privileges as those defined in this list.

BATCH-ALLOWED = *NO(...)
The command is not allowed in batch mode for user tasks which have only the privileges specified for PRIVILEGE.

> **PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>**
> The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).
>
> **PRIVILEGE = *SAME**
> The command is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).
>
> **PRIVILEGE = list-poss(64): <structured-name 1..30>**
> The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

BATCH-PROC-ALLOWED = <u>*UNCHANGED</u> / *YES(...) / *NO(...)
Specifies whether the command is allowed in batch mode within a procedure.

BATCH-PROC-ALLOWED = *YES(...)
The command is allowed in batch mode within a procedure for all user tasks which have at least one of the privileges specified for PRIVILEGE.

> **PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>**
> The command is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).
>
> **PRIVILEGE = *SAME**
> The command is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is allowed only for user tasks with exactly the same privileges as those defined in this list.

BATCH-PROC-ALLOWED = *NO(...)
The command is not allowed in batch mode within a procedure for user tasks which have only the privileges specified for PRIVILEGE.

PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

CMD-ALLOWED =
Specifies whether the command can be called with the CMD macro.

CMD-ALLOWED = <u>*UNCHANGED</u>
No change with regard to calling the command with the CMD macro.

CMD-ALLOWED = *YES(...)
The command can be called with the CMD macro. Calling with the CMD macro is allowed for all user tasks which have at least one of the privileges specified for PRIVILEGE. Possible restrictions as to the permissible modes of operation (DIALOG-ALLOWED, DIALOG-PROC-ALLOWED, BATCH-ALLOWED, BATCH-PROC-ALLOWED) do not apply when the command is called with the CMD macro.

UNLOAD = <u>*NO</u> / *YES
Specifies whether the calling program is to be unloaded. For commands implemented via a command procedure the calling program is always unloaded.

PRIVILEGE = <u>*UNCHANGED</u> / *SAME / list-poss(64): <structured-name 1..30>
The command call is allowed for all user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command call is allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command call is allowed only for user tasks with exactly the same privileges as those defined in this list.

CMD-ALLOWED = *NO(...)
The command cannot be called with the CMD macro by any user task which has only the privileges specified for PRIVILEGE.

PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
The command call is not allowed for user tasks with the privileges specified here (possible privileges are listed in the “SECOS” manual [10]).

PRIVILEGE = *SAME
The command call is not allowed for all user tasks with exactly the same privileges as those defined for the command itself (see the EXCEPT-PRIVILEGE operand on page 243).

PRIVILEGE = list-poss(64): <structured-name 1..30>
The command is not allowed for user tasks with the privileges defined in this list. If a user task has at least one privilege that is not included in this list, it may execute the command.

NEXT-INPUT =
Specifies what type of input is expected to follow the command. SDF needs this specification in order to conduct the guided dialog.

NEXT-INPUT = *UNCHANGED
No change with regard to the subsequent input.

NEXT-INPUT = *CMD
A command is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a command.

NEXT-INPUT = *STMT
A statement is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a statement. Example: a command implemented by means of a procedure starts a program whose first step is to read in a statement.

NEXT-INPUT = *DATA
Data is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as data. Example: a command implemented by means of a procedure starts a program whose first step is to read in data.

NEXT-INPUT = *ANY
It is not possible to predict what kind of input will follow.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) / list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the command.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the command.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the command.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the command.

PRIVILEGE = *ADD(...)
The privileges specified with *ADD(...) are assigned to the command in addition to the privileges already assigned to it.

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies which additional privileges are to be assigned to the command.

PRIVILEGE = *REMOVE(...)
The privileges specified with *REMOVE(...) are removed from the command.

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies which privileges are to be removed from the command.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the command.

MODIFY-CMD-ATTRIBUTES
Modify command attributes

The statement MODIFY-CMD-ATTRIBUTES is used to modify security attributes (input modes and privileges) for several commands simultaneously.

(part 1 of 2)

MODIFY-**CMD-ATT**RIBUTES

NAME = ***ALL** / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>

,**DIAL**OG-**ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

 ***YES**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

 ***NO**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**DIAL**OG-**PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

 ***YES**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

 ***NO**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**GUID**ED-**ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

 ***YES**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

 ***NO**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

 ***YES**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

 ***NO**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

,**BATCH-PROC-ALLOW**ED = ***UNCH**ANGED / ***YES**(...) / ***NO**(...)

 ***YES**(...)

  | **PRIV**ILEGE = ***UNCH**ANGED / ***SAME** / list-poss(64): <structured-name 1..30>

continued ➠

(part 2 of 2)

```
    *NO(...)
      |  PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
 ,CMD-ALLOWED = *UNCHANGED / *NO(...) / *YES(...)
    *NO(...)
      |  PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
    *YES(...)
      |  UNLOAD = *UNCHANGED / *YES / *NO
      | ,PRIVILEGE = *UNCHANGED / *SAME / list-poss(64): <structured-name 1..30>
 ,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
              list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *ADD(...)
      |  ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *REMOVE(...)
      |  REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Names of the commands whose attributes are to be modified.

The description of all other operands of the MODIFY-CMD-ATTRIBUTES statement are to be found under MODIFY-CMD as of page 238.

MODIFY-DOMAIN
Modify domain definition

The MODIFY-DOMAIN statement is used to modify the definition of a domain in the open syntax file. This domain must be the current object (see page 221).

The MODIFY-DOMAIN statement is very similar to the ADD-DOMAIN statement. The default value for all operands of the MODIFY-DOMAIN statement is *UNCHANGED, i.e. only those parts of a domain definition that are explicitly specified are modified. In guided dialog the operand form displays the current operand assignment rather than *UNCHANGED. When a value other than *UNCHANGED is specified for a MODIFY-DOMAIN operand, the old specifications in the domain definition are replaced by the new ones. All names given to a domain must be unique with respect to all other domains.

```
MODIFY-DOMAIN

 NAME = *UNCHANGED / <structured-name 1..30>
,HELP = *UNCHANGED / *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |  TEXT = *UNCHANGED / <c-string 1..500 with-low>
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..30>
Name of the domain which is to be used in guided dialog.

HELP =
Specifies whether there are help texts for the domain, and if so, what those texts are.

HELP = *UNCHANGED
No changes with regard to help texts.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Help text. *UNCHANGED is permissible only if a help text has already been defined for the language key.
The help text can contain the special character string “\n” for line breaks.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) /
list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the domain.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the domain.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the domain.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the domain.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the domain.

MODIFY-OPERAND
Modify operand definition

The MODIFY-OPERAND statement is used to modify the definition of an operand in the open syntax file. The operand must be the “current” object (see page 148). Afterwards the first operand value of the operand is the “current” object.

The MODIFY-OPERAND statement is very similar to the ADD-OPERAND statement. The default value for all operands of the MODIFY-OPERAND statement is *UNCHANGED, i.e. only those parts of an operand definition that are explicitly specified are modified. In guided dialog the operand form displays the current operand assignment rather than *UNCHANGED. When a value other than *UNCHANGED is specified for a MODIFY-OPERAND operand, the old specifications in the operand definition to be modified are replaced by the new ones. This also applies if a list is possible, i.e. instead of being added to the value list, the specified value replaces the list.

There is a special rule for STANDARD-NAME. When the operand to be modified is defined in an assigned reference file (see OPEN-SYNTAX-FILE), the old entries for STANDARD-NAME are retained and the name specified with MODIFY-OPERAND is added to them. All names given to the operand must be unique with respect to all other operands at the same level (or in the same structure). Definitions of the operand values pertaining to the operand are modified by means of the MODIFY-VALUE statement rather than the MODIFY-OPERAND statement.

(part 1 of 2)

```
MODIFY-OPERAND

 NAME = *UNCHANGED / <structured-name 1..20>
,INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
,STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..20>
,ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..20>
,MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
,HELP = *UNCHANGED / *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |   TEXT = *UNCHANGED / <c-string 1..500 with-low>
,DEFAULT = *UNCHANGED / *NONE / *JV(...) / *VARIABLE(...) / <c-string 1..1800 with-low>(...)
    *JV(...)
      |   JV-NAME = *UNCHANGED / <filename 1..54 without-gen-vers>
      |  ,ALTERNATE-DEFAULT = *UNCHANGED / *NONE / <c-string 1..1800>(...)
      |     <c-string 1..1800>(...)
      |       |   ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
    *VARIABLE(...)
      |   VARIABLE-NAME = *UNCHANGED / <composed-name 1..255>
      |  ,ALTERNATE-DEFAULT = *UNCHANGED / *NONE / <c-string 1..1800>(...)
      |     <c-string 1..1800>(...)
      |       |   ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
    <c-string 1..1800 with-low>(...)
      |   ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
,SECRET-PROMPT = *UNCHANGED / *YES / *NO
,STRUCTURE-IMPLICIT = *UNCHANGED / *YES / *NO
,REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
,DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
```

continued ➠

(part 2 of 2)

```
,LIST-POSSIBLE = *UNCHANGED / *NO / *YES(...)

   *YES(...)
     |  LIMIT = *UNCHANGED / *STD / <integer 1..3000>
     |
     | ,FORM = *UNCHANGED / *NORMAL / *OR

,LINES-IN-FORM = *UNCHANGED / <integer 1..15>

,PRESENCE = *UNCHANGED / *NORMAL / *EXTERNAL-ONLY / *INTERNAL-ONLY

,RESULT-OPERAND-LEVEL = *UNCHANGED / <integer 1..5>

,RESULT-OPERAND-NAME = *UNCHANGED / *SAME / <structured-name 1..20> / *POSITION(...) /
                                 *LABEL / *COMMAND-NAME

   *POSITION(...)
     |  POSITION = *UNCHANGED / <integer 1..3000>

,CONCATENATION-POS = *UNCHANGED / *NO / <integer 1..20>

,VALUE-OVERLAPPING = *UNCHANGED / *YES / *NO

,OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES

,PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>

   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *UNCHANGED / <structured-name 1..20>
(External) operand name, to be specified when the command or statement is entered (but see the operand PRESENCE=*INTERNAL-ONLY). The user can abbreviate this name on input, unlike STANDARD-NAME and ALIAS-NAME.

INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
Internal operand name. With the help of the internal operand name, SDF identifies an operand defined in several syntax files under different external names as being the same operand. When *STD is specified, SDF-A takes as the internal operand name the first eight characters (omitting hyphens) of the external name entered for the NAME operand.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..20>
Additional external operand name, which can be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted so long as the operand with this name exists in one of the assigned syntax files (see OPEN-SYNTAX-FILE). If the original external name given in the command or program documentation is declared to be the standard

name, it is thereby ensured that the operand can be entered using the original name, regardless of any name changes. Specifying *NAME causes SDF-A to take as STANDARD-NAME the external operand name entered for the NAME operand.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..20>
Additional external operand name, which can be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
Additional external operand name which defines the shortest permissible abbreviation for the operand. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other commands.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct operand but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the operand name (NAME).
3. The ALIAS-NAMEs and STANDARD-NAMEs of the operand must not be shorter than the minimum abbreviation if they are an abbreviation of the operand name.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the operand, and if so, what those texts are.

HELP = *UNCHANGED
No changes with regard to help texts.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Help text. *UNCHANGED is permissible only if a help text has already been defined for the language key.
The help text can contain the special character string “\n” for line breaks.

DEFAULT =
Defines whether a default value exists for the operand.

DEFAULT = <u>*UNCHANGED</u>
No change with regard to the default value.

DEFAULT = *NONE
There is no default value. The operand is a mandatory operand.

DEFAULT = *JV(...)
The operand is optional and its default value is stored in the job variable whose name is specified here. If a job variable is used as a default value, this default value is always analyzed by SDF at execution time. If it is not possible to access the job variable, the alternate default value defined with ALTERNATE-DEFAULT is used. If no alternate default value exists, the operand must be regarded as mandatory (corresponding to DEFAULT=*NONE). Consequently, DEFAULT=*JV(...) must not be used together with PRESENCE=*INTERNAL-ONLY.

JV-NAME = <u>*UNCHANGED</u>
The name of the job variable is not changed.

JV-NAME = <filename 1..54 without-gen-vers>
Name of the job variable.

ALTERNATE-DEFAULT = <u>*UNCHANGED</u> / *NONE / <c-string 1..1800 with-low>(...)
Alternate default value to be used if errors occur when accessing the job variable.

ALTERNATE-DEFAULT = *NONE
There is no alternate default value.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternate default value, to be specified in accordance with the rules governing the input of operands. The alternate default may thus be given in the form of a list enclosed in parentheses and must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If this default value is contained in the keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

DEFAULT = *VARIABLE(...)
The operand is optional and its default value is stored in the S variable whose name is specified here (see the “SDF-P” [12] User Guide).
If an S variable is used as a default value, this default value is always analyzed by SDF at execution time. If it is not possible to access the S variable, the alternate default value defined with ALTERNATE-DEFAULT is used. If no alternate default value exists, this operand must be regarded as mandatory (corresponding to DEFAULT = *NONE). Consequently, DEFAULT=*VARIABLE(...) must not be used together with PRESENCE=*INTERNAL-ONLY.

VARIABLE-NAME = *UNCHANGED
The name of the S variable is not changed.

VARIABLE-NAME = <composed-name 1..255>
Name of the S variable.

ALTERNATE-DEFAULT = *UNCHANGED / *NONE / <c-string 1..1800 with-low>(...)
Alternate default value to be used if errors occur when accessing the S variable.

ALTERNATE-DEFAULT = *NONE
There is no alternate default value.

ALTERNATE-DEFAULT = <c-string 1..1800 with-low>(...)
Alternate default value, to be specified in accordance with the rules governing the input of operands. The alternate default may thus also be given in the form of a list enclosed in parentheses and must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If this default value is contained in the keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

DEFAULT = <c-string 1..1800 with-low>(...)
This operand is optional and has the specified default value. It must be specified in accordance with the rules governing the input of operands. The default value may thus be specified as a list enclosed in parentheses and must be contained in a definition of the operand value associated with the operand (see ADD-VALUE). If the default value is contained in the keyword defined with STAR=*MANDATORY, it must also be entered with an asterisk.

ANALYSE-DEFAULT = *UNCHANGED / *YES / *NO
Specifies whether the given value will be analyzed syntactically by SDF-A as soon as the command or statement definition has been completed. This expedites analysis of the command or statement at runtime, but presupposes that the default value does not consist of a list or introduce a structure.

SECRET-PROMPT = *UNCHANGED / *NO / *YES
Specifies whether the operand is treated as a secret operand. The input fields for values of secret operands are kept blank, and logging is suppressed (see also ADD-VALUE..., OUTPUT=*SECRET-PROMPT and ADD-VALUE...,SECRET-PROMPT=*SAME/*NO).

STRUCTURE-IMPLICIT = *UNCHANGED / *NO / *YES
Relevant only for an operand that stands in a structure, and specifies whether the structure containing the operand is implicitly selected via global specification of the operand name when the command or statement is entered. (For details, see ADD-OPERAND, page 148).

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Specifies whether the operand may be deleted (see REMOVE). If the operand has been defined with REMOVE-POSSIBLE=*NO in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE), SDF-A rejects a change to *YES.

DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the operand is allowed in interactive mode. Specifying *YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in interactive mode.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the operand is allowed in interactive mode within a procedure. Specifying *YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in interactive mode within a procedure.

GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the operand is allowed in guided dialog. Specifying YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in guided dialog.
GUIDED-ALLOWED=*NO is not suitable for security-related aspects, since operands defined with this setting are shown in the procedure error dialog as well as for /SHOW-CMD and //SHOW-STMT with FORM=*UNGUIDED.

BATCH-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the operand is allowed in batch mode. Specifying *YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in batch mode.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the operand is allowed in batch mode within a procedure. Specifying *YES presupposes that the command or statement and, where applicable, the operand value introducing the structure are allowed in batch mode within a procedure.

LIST-POSSIBLE =
Specifies whether a list is allowed at the operand position. The ADD-VALUE statement is used to define for which of the operand values a list is allowed.

LIST-POSSIBLE = *UNCHANGED
No change with regard to whether a list is allowed.

LIST-POSSIBLE = *NO
No list is allowed.

LIST-POSSIBLE = *YES(...)
A list is allowed.

LIMIT = *UNCHANGED / *STD / <integer 1..3000>
Specifies the maximum number of list elements. Unless otherwise specified, SDF-A sets the value at 2000 (see also page 377).

FORM = *UNCHANGED / *NORMAL / *OR
Specifies whether the list elements are to be addressed individually (*NORMAL) or passed to the implementation converted into a single value using logical OR (see section “Format of the standardized transfer area” on page 365”). The latter is appropriate only for list elements of the data type KEYWORD, for which hexadecimal transfer values have been defined (see ADD-VALUE...,VALUE=<c-string>(...,OUTPUT=<x-string>...). A specification here is relevant only when the defined operand pertains to a statement or to a command defined with IMPLEMENTOR=*TPR(..,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). The specification made here must be consistent with the transfer area defined in the implementation.

LINES-IN-FORM = *UNCHANGED / <integer 1..15>
Specifies the number of input lines in the guided dialog form.

PRESENCE =
Specifies whether the operand is to be suppressed.

PRESENCE = *UNCHANGED
No change as far as suppressing the operand is concerned.

PRESENCE = *NORMAL
The operand is not suppressed.

PRESENCE = *EXTERNAL-ONLY
Transfer of the operand to the implementation is suppressed (e.g. an operand that is no longer needed but must be retained at the user interface for compatibility reasons, or an operand that serves merely to group further operands in a structure).

PRESENCE = *INTERNAL-ONLY
The operand is suppressed at the user interface. Together with the then mandatory definition of a default value (see the DEFAULT operand), a fixed value may be assigned to a parameter implemented in this way without an operand appearing to the user in the command or statement format. If a structure is attached to the operand, all of the sub-operands contained in the structure will be integrated into the higher level.
PRESENCE=*INTERNAL-ONLY must not be used together with DEFAULT=*JV(...) or DEFAULT=*VARIABLE(...).

RESULT-OPERAND-LEVEL = *UNCHANGED / <integer 1..5>
Specifies the structure level at which the operand is to be passed to the implementation. For an operand not attached to a structure, this value must be 1. The following applies to an operand attached to a structure: the RESULT-OPERAND-LEVEL is equal to or less than the structure level at which the operand stands in the input format of the command or statement. It is lower than, equal to or one higher than the RESULT-OPERAND-LEVEL of the operand to which the operand value introducing the structure belongs. For statements and for commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...), see also ADD-VALUE...STRUCTURE=*YES(...,FORM=...).

RESULT-OPERAND-NAME =
Specifies how the implementation identifies the operand in the transfer area or string that SDF passes to it. Note: SDF uses a transfer area (see section “Format of the standardized transfer area” on page 365”) for statements and for commands defined with IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). SDF passes a string in the case of commands defined with IMPLEMENTOR=*PROCEDURE (...) or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=STRING*,...) (see ADD-CMD).

RESULT-OPERAND-NAME = *UNCHANGED
No change with regard to the RESULT-OPERAND-NAME.

RESULT-OPERAND-NAME = *SAME
Permissible only when operands are transferred by means of a string. In the string to be passed, the operand has the same name as the one given to it with NAME=.

RESULT-OPERAND-NAME = <structured-name 1..20>
Permissible only when operands are transferred by means of a string. In the string to be passed, the operand has the name specified.

RESULT-OPERAND-NAME = *POSITION(...)
In the transfer area (see section “Format of the standardized transfer area” on page 365) or in the string to be passed, the operand has a specified position. A name is not assigned to it.

POSITION = *UNCHANGED / <integer 1..3000>
Specifies the position. For operands attached to a structure, the position is specified relative to the structure, i.e. the first operand in the structure is assigned position 1 if the RESULT-OPERAND-LEVEL that was defined for the current operand is higher than that of the operand at the level above it.

RESULT-OPERAND-NAME = *LABEL
Permissible only for operands in commands defined with IMPLEMENTOR=*TPR(..., CMD-INTERFACE=*STRING,...) (see ADD-CMD). This operand value is reserved for Fujitsu Siemens Computers Development and is therefore not described here.

RESULT-OPERAND-NAME = *COMMAND-NAME
Permissible only for operands in commands defined with IMPLEMENTOR=*TPR(..., CMD-INTERFACE=*STRING,...) (see ADD-CMD). This operand value is reserved for Fujitsu Siemens Computers Development and is therefore not described here.

CONCATENATION-POS = *UNCHANGED / *NO / <integer 1..20>
Specifies whether and, if so, how the operand is to be combined with other input operands to form a single operand in the string to be passed to the implementation. The input operands are concatenated. The position they take when concatenated must be specified here. It is presupposed that the transfer to the implementation is in the form of a string (commands that are defined with IMPLEMENTOR=*PROCEDURE or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...); see ADD-CMD). All input operands to be concatenated must have the same RESULT-OPERAND-NAME.

VALUE-OVERLAPPING = *UNCHANGED / *YES
Specifies whether overlapping of data types is to be permitted in the definition of the operand values. When the command or statement is input, SDF checks the value which has been entered, with the aid of the data type definitions and in the order they were specified for the operand. If the data type KEYWORD-(NUMBER) has been specified, SDF checks whether the entered value is unique with respect to further definitions of this type. In addition, SDF checks the attributes which have been defined (e.g. length, value range) for the entered value. If these attributes do not match, checking continues with the next data type defined. Overlapping of data types is supported starting with SDF Version 1.3. If a syntax file with the overlapping of data types is used in an older version of SDF, this leads to errors. A list of mutually exclusive data types can be found on page 623). If the overlapping of data types has been permitted for an operand, this overlapping cannot be canceled by the MODIFY-OPERAND statement (*NO cannot be specified). Modification is possible only by deleting the operand and then redefining it, along with its values.

OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES
This is only relevant for statements, and for commands defined with IMPLE-MENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...); see the ADD-CMD statement. OVERWRITE-POSSIBLE determines whether the operand default value can be replaced by a value created dynamically by the implementation (see the DEFAULT operand

in the CMDCST, CMDRST and CMDTST macros). The program-generated value must be a valid operand value. In guided dialog, SDF shows the implementation-specific value in the form display.

PRIVILEGE = <u>*UNCHANGED</u> / *SAME / *EXCEPT(.....) /
list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the operand.

PRIVILEGE = *SAME
The operand is assigned the same privileges as those defined for the associated command or statement. If the operand is part of a structure, it is assigned the same privileges as the operand value which introduces the structure.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the operand.

> **EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>**
> Specifies the privileges that are not assigned to the operand.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the operand.

MODIFY-PROGRAM
Modify program definition

The MODIFY-PROGRAM statement is used to modify the definition of a program in the open syntax file. This program must be the “current” object (see page 148). The name given for the program must be unique with regard to all programs defined in the syntax file.

The MODIFY-PROGRAM statement is very similar to the ADD-PROGRAM statement.

```
MODIFY-PROGRAM

 NAME = *UNCHANGED / <structured-name 1..30>
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
,COMMENT-LINE = *UNCHANGED / *NONE / *STD / <c-string 1..50 with-low>
```

NAME = *UNCHANGED / <structured-name 1..30>
(External) program name, which is displayed in guided dialog. This name is freely selectable (it does not need to be the same as the module or phase name). In guided dialog, the operand form displays the current program name rather than *UNCHANGED.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the program.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the program.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the program.

> **EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>**
> Specifies the privileges that are not assigned to the program.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the program.

COMMENT-LINE =
Specifies which program comment line is to be displayed in the guided dialog. The program comment line appears at the top of guided dialog forms.

COMMENT-LINE = *UNCHANGED
No changes are made with respect to the program comment line.

COMMENT-LINE = *NONE
No program comment line is displayed.

COMMENT-LINE = *STD
The version and creation date of the program are displayed in the program comment line. Object modules (elements of type R) have no internal version, so the execution date is shown instead of the creation date.

COMMENT-LINE = <c-string 1..50 with-low>
String to be output as the program comment line.

MODIFY-STMT
Modify statement definition

The MODIFY-STMT statement is used to modify the definition of a statement in the open syntax file. The statement must be the current object (see page 148). Afterwards, the first operand for this statement is the current object. The MODIFY-STMT statement is very similar to the ADD-STMT statement.

The default value for all operands for MODIFY-STMT is *UNCHANGED, i.e. only those parts of a statement definition that are explicitly specified are modified. In guided dialog the operand form displays the current operand assignment rather than *UNCHANGED. When a value other than *UNCHANGED is specified for a MODIFY-STMT operand, the old specifications in the statement definition to be modified are replaced by the new ones. This also applies if a list is possible, i.e. the specified value is not added to the value list but replaces it.

There is a special rule for STANDARD-NAME. When the statement to be modified is defined in an assigned reference file (see OPEN-SYNTAX-FILE), the old entries for STANDARD-NAME are retained, and the name specified with MODIFY-STMT is added to them. All names given to the statement must be unique with respect to the set of statements pertaining to the program.

Definitions of the operands and operand values pertaining to the statement are modified by means of the MODIFY-OPERAND and MODIFY-VALUE statements, rather than by means of MODIFY-STMT.

```
MODIFY-STMT

 NAME = *UNCHANGED / <structured-name 1..30>
,PROGRAM = *UNCHANGED / <structured-name 1..30>
,STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30>
,ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
,MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30>
,HELP = *UNCHANGED / *NO / list-poss(2000): <name 1..1>(...)
    <name 1..1>(...)
      |   TEXT = *UNCHANGED / <c-string 1..500 with-low>
,MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
,REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
,DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,NEXT-INPUT = *UNCHANGED / *STMT / *DATA / *ANY
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
                  list-poss(64): <structured-name 1..30>
    *EXCEPT(...)
      |   EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *ADD(...)
      |   ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
    *REMOVE(...)
      |   REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
,STMT-VERSION = *UNCHANGED / *NONE / <integer 1..999>
```

NAME = *UNCHANGED / <structured-name 1..30>
(External) statement name, to be specified when the statement is entered. The user can abbreviate this name on input, in contrast to the STANDARD-NAME and ALIAS-NAME.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30>
Additional external statement name, which can be alternatively used when entering the statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted. It is reserved for Fujitsu Siemens Computers Software Development and is therefore not described here.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Additional external statement name, which can be alternatively used when entering the statement. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *NO / <structured-name 1..30>
Additional external statement name which defines the shortest permissible abbreviation for the statement. Any shorter abbreviation will not be accepted, even if it is unambiguous with respect to other statements.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct statement but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.
2. The minimum abbreviation must be derived from the statement name (NAME).
3. The ALIAS-NAMEs and STANDARD-NAMEs of the statement must not be shorter than the minimum abbreviation if they are an abbreviation of the statement name.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

HELP =
Specifies whether there are help texts for the statement, and if so, what those texts are.

HELP = *UNCHANGED
No change with regard to help texts.

HELP = *NO
There are no help texts.

HELP = list-poss(2000): <name 1..1>(...)
There are help texts in the specified languages (E = English, D = German). SDF uses the language specified for message output.

TEXT = *UNCHANGED / <c-string 1..500 with-low>
Help text. UNCHANGED is permissible only if a help text has already been defined for the language key.
The help text can contain the special character string “\n” for line breaks.

MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
Number of operand positions to be reserved at the highest level in structured transfer. When STD is specified, the size of the operand array corresponds to requirements. However, the array may also be made larger to accommodate planned expansions.

REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
Specifies whether the statement may be deleted (see REMOVE). This operand cannot be changed for the standard statements.

DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the statement is allowed in interactive mode. This operand cannot be changed for the standard statements.

DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the statement is allowed in interactive mode within a procedure. This operand cannot be changed for the standard statements.

GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the statement is allowed in guided dialog. This operand cannot be changed for the standard statements.

BATCH-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the statement is allowed in batch mode. This operand cannot be changed for the standard statements.

BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether the statement is allowed in batch mode within a procedure. This operand cannot be changed for the standard statements.

NEXT-INPUT =
Specifies what type of input is expected following the statement. SDF needs this specification in order to conduct the guided dialog.

NEXT-INPUT = *UNCHANGED
No change with regard to the subsequent input expected.

NEXT-INPUT = *STMT
A statement is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as a statement.

NEXT-INPUT = *DATA
Data is expected as the next entry. SDF interprets input in the NEXT field of the guided dialog as data.

NEXT-INPUT = *ANY
It is not possible to predict what kind of input will follow.

PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) / list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the statement.

PRIVILEGE = *ALL
All privileges currently defined and all subsequently defined privileges are assigned to the statement.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the statement.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the statement.

PRIVILEGE = *ADD(...)
The privileges specified with *ADD(...) are assigned to the command in addition to the privileges already assigned to it.

ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies which additional privileges are to be assigned to the command.

PRIVILEGE = *REMOVE(...)
The privileges specified with *REMOVE(...) are removed from the command.

REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies which privileges are to be removed from the command.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the statement.

STMT-VERSION = *UNCHANGED / *NONE / <integer 1..999>
Version of the statement. The value is transferred in the standardized transfer area. *NONE means that a zero value can be entered in the standardized transfer area. STMT-VERSION is ignored if the program which the statement belongs to is not working with the new CMDRST and CMDTST macros or is using the old format of the standardized transfer area. More information on the format of the standardized transfer area and the SDF macros can be found in section “Format of the standardized transfer area” on page 365ff and section “SDF macros” on page 379ff.

MODIFY-STMT-ATTRIBUTES
Modify statement attributes

The MODIFY-STMT-ATTRIBUTES statement is used to modify the security attributes (input modes and privileges) of several statements of a program simultaneously.

```
MODIFY-STMT-ATTRIBUTES

NAME = *ALL / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
,PROGRAM = <structured-name 1..30>
,DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,PRIVILEGE = *UNCHANGED / *ALL / *EXCEPT(...) / *ADD(...) / *REMOVE(...) /
                     list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
     |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
   *ADD(...)
     |  ADD-PRIVILEGE = list-poss(64): <structured-name 1..30>
   *REMOVE(...)
     |  REMOVE-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

NAME = *ALL / <structured-name 1..30 with-wild> /
list-poss(2000): <structured-name 1..30>
Names of the statements whose attributes are to be modified.

PROGRAM = <structured-name 1..30>
External name of the program to which the statements belong.
This name cannot be modified using MODIFY-STMT-ATTRIBUTES.

The description of all other operands of the MODIFY-STMT-ATTRIBUTES statement can be found under MODIFY-STMT as of page 262.

MODIFY-VALUE
Modify operand value definition

The MODIFY-VALUE statement is used to modify the definition of an operand value in the syntax file being processed. The operand value must be the current object (see page 148). If the value introduces a structure, the first operand of the structure then becomes the current object. Otherwise, it is a question of whether there are still more values defined for the operand to which the modified value belongs. If there are, then the next value belonging to the operand becomes the current object. If there are no further values, then the next operand in the command or statement structure becomes the current object. If the last value of a structure is changed, the value following the one introducing the structure becomes the current object. If this value is missing, the subsequent operand becomes the current object.

The MODIFY-VALUE statement is very similar to the ADD-VALUE statement. The default value for all operands for MODIFY-VALUE is *UNCHANGED, i.e. only those parts of an operand value definition that are explicitly specified are modified. In guided dialog the operand form displays the current operand assignment rather than *UNCHANGED. When a value other than *UNCHANGED is specified for a MODIFY-VALUE operand, the old specifications in the operand value definition to be modified are replaced by the new ones. This also applies if a list is possible, i.e. instead of being added to the value list, the specified value replaces it. There is a special rule for STANDARD-NAME. When the operand value to be modified is defined in an assigned reference file (see OPEN-SYNTAX-FILE), the old entries for STANDARD-NAME are retained and the name specified with MODIFY-VALUE is added to them.

All names given to the operand value must be unique with regard to its environment.

(part 1 of 6)

```
MODIFY-VALUE

TYPE = *UNCHANGED / *ALPHANUMERIC-NAME(...) / *CAT-ID / *COMMAND-REST(...) /
          *COMPOSED-NAME(...) / *C-STRING(...) / *DATE(...) / *DEVICE(...) / *FIXED(...) /
          *FILENAME(...) / *INTEGER(...) / *KEYWORD(...) / *KEYWORD-NUMBER(...) / *LABEL(...) /
          *NAME(...) / *PARTIAL-FILENAME(...) / *POSIX-PATHNAME(...) / *POSIX-FILENAME(...) /
          *PRODUCT-VERSION(...) / *STRUCTURED-NAME(...) / *TEXT(...) / *TIME(...) / *VSN(...) /
          *X-STRING(...) / *X-TEXT(...)

   *ALPHANUMERIC-NAME(...)
      |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
      | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
      | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
      |
      |     *YES(...)
      |       |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
      |       | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>

   *COMMAND-REST(...)
      |  LOWER-CASE = *UNCHANGED / YES / *NO

   *COMPOSED-NAME(...)
      |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
      | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
      | ,UNDERSCORE = *UNCHANGED / *NO / *YES
      | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
      |
      |     *YES(...)
      |       |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
      |       | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..1800>

   *C-STRING(...)
      |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
      | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
      | ,LOWER-CASE = *UNCHANGED / YES / *NO

   *DATE(...)
      |  COMPLETION = *UNCHANGED / *NO / *YES
```

continued ➠

(part 2 of 6)

```
*DEVICE(...)
  |  CLASS-TYPE = *UNCHANGED / list-poss(2000): *DISK(...) / *TAPE(...)
  |    *DISK(...)
  |      |  EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
  |      | ,SCOPE = *UNCHANGED / *ALL / *STD-DISK
  |    *TAPE(...)
  |      |  EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
  | ,ALIAS-ALLOWED = *UNCHANGED / *YES / *NO
  | ,VOLUME-TYPE-ONLY = *UNCHANGED / *YES / *NO
  | ,RESULT-VALUE = *UNCHANGED / *BY-NAME / *BY-CODE
*FIXED(...)
  |  LOWEST = *UNCHANGED / *ANY / <fixed -2147483648..2147483647>
  | ,HIGHEST = *UNCHANGED / *ANY / <fixed -2147483648..2147483647>
*FILENAME(...)
  |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
  | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
  | ,CATALOG-ID = *UNCHANGED / *YES / *NO
  | ,USER-ID = *UNCHANGED / *YES / *NO
  | ,GENERATION = *UNCHANGED / *YES / *NO
  | ,VERSION = *UNCHANGED / *YES / *NO
  | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
  |    *YES(...)
  |      |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
  |      | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..80>
  | ,PATH-COMPLETION = *UNCHANGED / *YES / *NO
  | ,TEMPORARY-FILE = *UNCHANGED / *YES / *NO
*INTEGER(...)
  |  LOWEST = *UNCHANGED / *ANY / <integer -2147483648..2147483647>
  | ,HIGHEST = *UNCHANGED / *ANY / <integer -2147483648..2147483647>
  | ,OUT-FORM = *UNCHANGED / *BINARY / *PACKED / *UNPACKED / *CHAR / *STD
```

continued ➠

(part 3 of 6)

```
*KEYWORD(...)
   |  STAR = *UNCHANGED / *OPTIONAL / *MANDATORY
*KEYWORD-NUMBER(...)
   |  STAR = *UNCHANGED / *OPTIONAL / *MANDATORY
*LABEL(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..8>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..8>
*NAME(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,UNDERSCORE = *UNCHANGED / *YES / *NO
   | ,LOWER-CASE = *UNCHANGED / YES / *NO
   | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
   |     *YES(...)
   |        |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
   |        | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
*PARTIAL-FILENAME(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
   | ,CATALOG-ID = *UNCHANGED / *YES / *NO
   | ,USER-ID = *UNCHANGED / *YES / *NO
   | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
   |     *YES(...)
   |        |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
   |        | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 2..79>
*POSIX-PATHNAME(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1023>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1023>
   | ,WILDCARD = *UNCHANGED / *YES / *NO
   | ,QUOTES = *UNCHANGED / *OPTIONAL / *MANDATORY
```

continued ➠

(part 4 of 6)

```
*POSIX-FILENAME(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,WILDCARD = *UNCHANGED / *YES / *NO
   | ,QUOTES = *UNCHANGED / *OPTIONAL / *MANDATORY
*PRODUCT-VERSION(...)
   |  USER-INTERFACE = *UNCHANGED / *NO / *YES(...) *ANY(...)
   |     *YES(...)
   |       |  CORRECTION-STATE = *UNCHANGED /*YES / *NO / *ANY
   |     *ANY(...)
   |       |  CORRECTION-STATE = *UNCHANGED / *ANY / *NO / *YES
*STRUCTURED-NAME(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
   | ,WILDCARD = *UNCHANGED / *NO / *YES(...)
   |     *YES(...)
   |       |  TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
   |       | ,LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
*TEXT(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
   | ,LOWER-CASE = *UNCHANGED / *YES / *NO
   | ,SEPARATORS = *UNCHANGED / *YES / *NO
*TIME(...)
   |  OUT-FORM = *UNCHANGED / *STD / *BINARY / *CHAR
*VSN(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
*X-STRING(...)
   |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
   | ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
```

continued ➠

(part 5 of 6)

```
    *X-TEXT(...)
      |  SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
      |  ,LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
      |  ,ODD-POSSIBLE = *UNCHANGED / *YES / *NO
,INTERNAL-NAME = *UNCHANGED / *STD / <alphanum-name 1..8>
,REMOVE-POSSIBLE = *UNCHANGED / *YES / *NO
,SECRET-PROMPT = *UNCHANGED / *SAME / *NO
,DIALOG-ALLOWED = *UNCHANGED / *YES / *NO
,DIALOG-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,GUIDED-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-ALLOWED = *UNCHANGED / *YES / *NO
,BATCH-PROC-ALLOWED = *UNCHANGED / *YES / *NO
,STRUCTURE = *UNCHANGED / *NO / *YES(...)
    *YES(...)
      |  SIZE = *UNCHANGED / *SMALL / *LARGE
      |  ,FORM = *UNCHANGED / *NORMAL / *FLAT
      |  ,MAX-STRUC-OPERAND = *UNCHANGED / *STD / <integer 1..3000>
,LIST-ALLOWED = *UNCHANGED / *YES / *NO
,VALUE = *UNCHANGED / *NO / list-poss(2000): <c-string 1..1800 with-low>(...)
    <c-string 1..1800 with-low>(...)
      |  STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME /
      |                                <structured-name 1..30> / <c-string 1..30>
      |  ,ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
      |  ,GUIDED-ABBREVIATION = *UNCHANGED / *NAME / <structured-name 1..30> / <c-string 1..30>
      |  ,MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30> / <c-string 1..30>
      |  ,NULL-ABBREVIATION = *UNCHANGED / *YES / *NO
      |  ,OUTPUT = *UNCHANGED / SAME / *EMPTY-STRING / *DROP-OPERAND /
      |                  <c-string 1..1800> / <x-string 1..3600>
```

continued ➠

(part 6 of 6)

```
      | ,OUT-TYPE = *UNCHANGED / *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST /
      |                *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE / *FIXED / *FILENAME /
      |                *INTEGER / *KEYWORD / *KEYWORD-NUMBER / *LABEL / *NAME /
      |                *PARTIAL-FILENAME / *PRODUCT-VERSION / *POSIX-PATHNAME /
      |                *POSIX-FILENAME / *STRUCTURED-NAME / *TEXT / *TIME / *VSN /
      |                *X-STRING / *X-TEXT
      |
      | ,OVERWRITE-POSSIBLE = *UNCHANGED / *YES / *NO
,OUTPUT = *UNCHANGED / *SECRET-PROMPT / *NORMAL(...)
   *NORMAL(...)
      |  AREA-LENGTH = *UNCHANGED / *VARIABLE / <integer 1..3000>
      | ,LEFT-JUSTIFIED = *UNCHANGED / *STD / *YES / *NO
      | ,FILLER = *UNCHANGED / *STD / <c-string 1..1> / <x-string 1..2>
      | ,STRING-LITERALS = *UNCHANGED / *HEX / *CHAR / *NO
      | ,HASH = *UNCHANGED / *NO / *YES(...)
      |    *YES(...)
      |       |  OUTPUT-LENGTH = *UNCHANGED / <integer 2..32>
,PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) / list-poss(64): <structured-name 1..30>
   *EXCEPT(...)
      |  EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
```

TYPE =
Specifies the data type of the operand value. The various values defined for an operand must not be mutually exclusive with respect to the data type (see page 623). (If this is not possible in exceptional cases, the value YES must be specified for VALUE-OVERLAPPING in the ADD-OPERAND or MODIFY-OPERAND statement.) Otherwise it is not possible, for example, to define a value of the type NAME for an operand together with an alternative value of the type STRUCTURED-NAME. Only the data type KEYWORD may be specified in more than one alternative operand value definition. When a command or statement is entered, SDF checks whether an operand value entered is of the type defined for it. In the following descriptions of the data types, the term “alphanumeric character” is used repeatedly. This may be a letter (A, B, C, ..., Z), a digit (0, 1, 2, ..., 9) or a special character (@, #, $).

TYPE = *UNCHANGED
No change with respect to the data type.

TYPE = *ALPHANUMERIC-NAME(...)
Specifies that the operand value is of the data type ALPHANUMERIC-NAME. This is defined as a sequence of alphanumeric characters.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string must have a minimum length and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Specifies whether the string can be a wildcard selector or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard (selector or constructor) is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *CAT-ID
Specifies that the operand value is of the data type CAT-ID, defined as a sequence of up to 4 characters (without “:” delimiters) from the ranges A-Z and 0-9; the special characters $, @ and # are not permitted. A four-character catalog ID must not begin with the string ’PUB’.

TYPE = *COMMAND-REST(...)
Specifies that the operand value is of the data type COMMAND-REST. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *COMPOSED-NAME(...)
Specifies that the operand value is of the data type COMPOSED-NAME. This data type is very similar to the data type FILENAME, with the following differences:
- catid, userid, version, generation and wildcard are not permitted
- the length is not restricted to 54 characters
- there does not need to be a letter in the name.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

UNDERSCORE = *UNCHANGED / *NO / *YES
Specifies whether an underscore character (_) is accepted.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..1800>
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..1800>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *C-STRING(...)
Specifies that the operand value is of the data type C-STRING. This is defined as a sequence of EBCDIC characters, enclosed in single quotes. It may be prefixed with the letter C. A single quote as a value within the limiting apostrophes must be specified twice.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

LOWER-CASE = *UNCHANGED / *NO / *YES
Specifies whether lowercase letters within the apostrophes are retained.

TYPE = *DATE(...)
Specifies that the operand value is of the data type DATE. This is defined as a sequence of one four-digit number and two two-digit numbers joined together by hyphens (<year>-<month>-<day>). The year may also be specified with two digits instead of four.

COMPLETION = *UNCHANGED / *NO / *YES
Specifies whether a two-digit year specification is to be extended. If *YES is specified, SDF extends two-digit year specifications (in the form yy-mm-dd) as follows:
- 20yy-mm-dd if yy < 60
- 19yy-mm-dd if yy ≥ 60.

TYPE = *DEVICE(...)
Specifies that the operand value is of the data type DEVICE. In guided dialog, the user is offered a list of the disk or tape devices available in the system for operands whose value is defined with the data type DEVICE (see the manual “System Installation” [9]).

CLASS-TYPE = *UNCHANGED / list-poss(2000): *DISK(...) / *TAPE(...)
Specifies the class type of the device.

CLASS-TYPE = *DISK(...)
The class type of the device is disk.

EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
Specifies the disks that will not appear in the list of available devices.

SCOPE = *UNCHANGED / *ALL / *STD-DISK
Specifies whether all the disks appear in the list of available devices or only the standard disks specified by DMS (see the “Introductory Guide to DMS“ [7]).
In BS2000/OSD-BC < V3.0, SCOPE=*ALL is always valid.

SCOPE = *ALL
All disks appear in the list.

SCOPE = *STD-DISK
Only disks which are specified in DMS as standard appear in the list.

CLASS-TYPE = *TAPE(...)
The class type of the device is tape.

EXCEPT = *UNCHANGED / *NO / list-poss(50): <text 1..8 without-sep>
Specifies the devices that will not appear in the list of available devices.

ALIAS-ALLOWED = *UNCHANGED / *YES / *NO
Specifies whether an alias is allowed for the device.

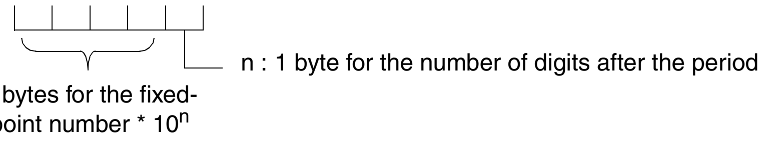
VOLUME-TYPE-ONLY = *UNCHANGED / *NO / *YES
Specifies whether the volume type is accepted.

RESULT-VALUE = *UNCHANGED / *BY-NAME / *BY-CODE
Specifies the format in which SDF passes information to the implementation.

RESULT-VALUE = *BY-NAME
SDF outputs the external device name. The external device name is 8 characters in length.

RESULT-VALUE = *BY-CODE
SDF outputs the internal device code. The internal device code is 2 bytes long.

TYPE = *FIXED(...)
Specifies that the operand value is of the data type FIXED. This is defined as follows:
[sign][digits].[digits]
[sign] is + or –
[digits] are 0..9
FIXED must consist of at least one digit; it must not exceed 10 characters (digits and a '.').
The value is stored in the standardized transfer area in the following format:


n : 1 byte for the number of digits after the period
4 bytes for the fixed-point number * 10^n

LOWEST = *UNCHANGED / *ANY / <fixed -2147483648 .. 2147483647>.
Specifies whether there is a lower limit for the fixed value, and if so, what this limit is.

HIGHEST = *UNCHANGED / *ANY / <fixed -2147483648 .. 2147483647>
Specifies whether there is an upper limit for the fixed value, and if so, what this limit is.

TYPE = *FILENAME(...)
Specifies that the operand value is of the data type FILENAME. The definition of the string to be entered is the one given in the “Introductory Guide to DMS” [7] for fully qualified file names.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..80>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

CATALOG-ID = *UNCHANGED / *YES / *NO
Specifies whether the catalog ID may be specified as part of the file name.

USER-ID = *UNCHANGED / *YES / *NO
Specifies whether the user ID may be specified as part of the file name.

GENERATION = *UNCHANGED / *YES / *NO
Specifies whether a generation number may be specified as part of the file name.

VERSION = *UNCHANGED / *YES / *NO
Specifies whether a version identifier may be specified as part of the file name.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.
The data type <filename x..y with-wild> also contains partially qualified file names, i.e. it is not necessary to define an additional value of the data type <partial-filename> for the operand.

TYPE = _*UNCHANGED_ / *SELECTOR / *CONSTRUCTOR
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = _*UNCHANGED_ / *LONGEST-LENGTH / <integer 1..80>
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..80>
Specifies the maximum length of the name matched by the wildcard.

PATH-COMPLETION = _*UNCHANGED_ / *NO / *YES
Specifies whether the file name is extended to a full path name upon transfer to the implementation. This includes the substitution of aliases by means of the ACS function. PATH-COMPLETION=*YES may only be specified if CATALOG-ID and USER-ID are permitted and if wildcards are not permitted in the file name.

TEMPORARY-FILE = _*UNCHANGED_ / *YES / *NO
Specifies whether temporary file names are permitted.

TYPE = *INTEGER(...)
Specifies that the operand value is of the data type INTEGER. This is defined as a sequence of digits which may be preceded by a sign.

LOWEST = _*UNCHANGED_ / *ANY / <integer -2 147 483648..2 147 483647>
Specifies whether there is a lower limit for the integer, and if so, what that lower limit is.

HIGHEST = *UNCHANGED / *ANY / <integer -2 147 483648..2 147 483647>
Specifies whether there is an upper limit for the integer, and if so, what that upper limit is.

OUT-FORM = <u>*UNCHANGED</u> / *STD / *BINARY / *PACKED / *UNPACKED / *CHAR
Specifies the form in which the integer is to be passed by SDF to the implementation. If passed in a transfer area (see section “Format of the standardized transfer area” on page 365), SDF-A converts *STD to *BINARY. If a string is passed (commands defined with IMPLEMENTOR=*PROCEDURE or IMPLEMENTOR=*TPR (...,CMD-INTERFACE=*STRING,...); see ADD-CMD), SDF-A converts OUT-FORM=*STD to *CHAR.

TYPE = *KEYWORD(...)
Specifies that the operand value is of the data type *KEYWORD. This is defined as a sequence of alphanumeric characters. This character string may be subdivided into several substrings, joined together by hyphens. Substrings may contains periods, but the periods must not be at the beginning or the end of the substring. The entire string, or, as the case may be, the first substring, must begin with a letter or special character. The value range for an operand value of the type *KEYWORD is limited to one or a finite number of precisely defined individual values (see the VALUE operand of this statement).
From SDF-A/SDF Version 2.0 onwards, the value ’*...’ is also accepted. This value can be used where a list of keywords has to be defined for an operand (e.g. the definition for all external devices). This permits new keywords (e.g. new device names) to be inserted without having to modify the syntax file, since ’*...’ causes SDF to accept additional values and to analyze them as keywords. The data type *KEYWORD may be up to 30 characters long.

STAR =
Specifies whether the string must be preceded by an asterisk when entered.

STAR = <u>*UNCHANGED</u>
No change with respect to the asterisk.

STAR = *OPTIONAL
An asterisk may be prefixed, but need not be. If an asterisk is prefixed, it is suppressed when the operand value is passed to the implementation.

STAR = *MANDATORY
An asterisk must be prefixed (necessary in order to distinguish between alternative data types).

TYPE = *KEYWORD-NUMBER(...)
Specifies that the operand value is of the data type KEYWORD-NUMBER. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

STAR = <u>*UNCHANGED</u>
No change with respect to the asterisk.

STAR = *OPTIONAL
An asterisk may be prefixed, but need not be.

STAR = *MANDATORY
An asterisk must be prefixed (necessary in order to distinguish between alternative data types).

TYPE = *LABEL(...)
Specifies that the operand value is of the data type LABEL. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *NAME(...)
Specifies that the operand value is of the data type NAME. This is defined as a sequence of alphanumeric characters, beginning with a letter or special character.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

UNDERSCORE = *UNCHANGED / *NO / *YES
Specifies whether an underscore character (_) is accepted.

LOWER-CASE = *UNCHANGED / *NO / *YES
Specifies whether lowercase characters are retained.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Specifies whether the string can be a wildcard selector or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *PARTIAL-FILENAME(...)
Specifies that the operand value is of the data type PARTIAL-FILENAME. The definition of the string to be entered is the one given in the “Introductory Guide to DMS” [7] for partially qualified file names.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 2..79>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

CATALOG-ID = *UNCHANGED / *YES / *NO
Specifies whether the catalog ID may be specified as part of the file name.

USER-ID = *UNCHANGED / *YES / *NO
Specifies whether the user ID may be specified as part of the file name.

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Specifies whether the string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 2..79>
Specifies the maximum length of the name matched by the wildcard (selector and constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 2..79>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *POSIX-PATHNAME(...)
Defines the data type of the operand value as POSIX-PATHNAME. The string entered here must comply with the conventions below:
– The following characters are allowed: letters, digits, and the characters ‘_’, ‘-’, ‘.’, and ‘/’ (the ‘/’ always serves as a delimiter between directories and file names). Control characters are not allowed.
– A POSIX-PATHNAME consists of POSIX-FILENAMEs, separated by ’/’. The total length of a POSIX-PATHNAME must not exceed 1023 characters.
– A ´\´(backslash) is used to escape metacharacters in all POSIX-specific names; the backslash character itself is not included when counting the length. The metacharacter * is likewise excluded from the count.

Metacharacters are characters used in wildcard patterns. The following metacharacters may be used:

*	matches zero, one, or any number of characters
?	matches any single character
[S]	matches any single character from the defined character set S
[!S]	matches any single character that is not an element of the defined character set S
where S is	a set of fixed characters (e.g. abcd) or a range of characters (e.g. a-d) or a combination of the above (set and range).

POSIX-PATHNAMEs which contain characters other than those listed above or which begin with a ’?’ or ’!’ must be enclosed within single quotes on input (as in the case of C-STRINGs). Since the single quotes are not a part of the file name, they are removed by SDF in the standardized transfer area or the transferred string. Single quotes that are part of a file name must be duplicated.
To avoid compatibility problems, the C-string syntax should always be used in procedures.

SHORTEST-LENGTH = *ANY / <integer 1..1023>
Specifies whether the character string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..1023>
Specifies whether the character string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *YES / *NO
Defines whether metacharacters (see above) may be specified in the name (instead of characters/strings) when entering a command or statement.

TYPE = *POSIX-FILENAME(...)
Defines the data type of the operand value as POSIX-FILENAME. The string to be entered here must comply with the conventions listed for POSIX-PATHNAMEs (see page 179f), but with the following restrictions:
– the slash (/) is not allowed
– the maximum length is limited to 255 characters.

SHORTEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *YES / *NO
Defines whether metacharacters (see POSIX-PATHNAME) may be specified in the name (instead of characters/strings) when entering a command or statement.

TYPE = *PRODUCT-VERSION(...)
Specifies that the operand value is of the data type PRODUCT-VERSION. The product version has the following format:

```
[[C]’][V][m]m.naso[’]     m, n: digit (0..9)
              |  |        a: letter
              |  |        s,o: digit
              |  |
              |  +------- Correction state
              +---------- Release status
```

“C”, “V” and “ ’ ” are optional characters and are suppressed in the SDF transfer area.

USER-INTERFACE = *UNCHANGED / *NO / *YES(...) / *ANY(...)
Specifies whether the release status of the user interface can or must be specified.

USER-INTERFACE = *NO
The release status of the user interface and the correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format: [[C]’][V][m]m.n[’].

USER-INTERFACE = *YES(...)
The release status of the user interface must be specified.

CORRECTION-STATE = *UNCHANGED / *YES / *NO / *ANY
Indicates whether the correction state can or must be specified.

CORRECTION-STATE = *YES
The correction state must be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.naso[’].

CORRECTION-STATE = *NO
The correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.na[’].

CORRECTION-STATE = *ANY
The correction state can be specified.

USER-INTERFACE = *ANY
The release status of the user interface can be specified.

CORRECTION-STATE = *UNCHANGED / *ANY / *NO / *ANY
Indicates whether the correction state can or must be specified.

CORRECTION-STATE = *ANY
The correction state can be specified.

CORRECTION-STATE = *NO
The correction state must not be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.na[’].

CORRECTION-STATE = *YES
The correction state must be specified. Specifications for the data type PRODUCT-VERSION then have the following format:
[[C]’][V][m]m.naso[’].

TYPE = *STRUCTURED-NAME(...)
Specifies that the operand value is of the data type STRUCTURED-NAME. This is defined as a sequence of alphanumeric characters. This string may be subdivided into several substrings, joined together by hyphens. The entire string, or, as the case may be, the first substring, must begin with a letter or special character.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..255>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

WILDCARD = *UNCHANGED / *NO / *YES(...)
Specifies whether wildcards (see the description of the SDF metasyntax in section 1.5) may be used in place of characters/strings within the name when the command or statement is entered.

WILDCARD = *NO
No wildcards are allowed when entering the operand value.

WILDCARD = *YES(...)
Wildcards may be used.

TYPE = *UNCHANGED / *SELECTOR / *CONSTRUCTOR
Specifies whether the character string can be a wildcard selector (search pattern) or wildcard constructor. Wildcard constructors are used to build names from strings generated with the aid of a wildcard selector.

TYPE = *SELECTOR
The string can be a wildcard selector; the data type receives the suffix with-wild (see the description of the SDF metasyntax in section 1.5).

TYPE = *CONSTRUCTOR
The string can be a wildcard constructor; the data type receives the suffix with-wild-constr (see the description of the SDF metasyntax in section 1.5).

LONGEST-LOGICAL-LEN = *UNCHANGED / *LONGEST-LENGTH / <integer 1..255>
Specifies the maximum length of the name matched by the wildcard (selector or constructor).

LONGEST-LOGICAL-LEN = *LONGEST-LENGTH
The maximum length of the name matched by the wildcard is the same as the length specified by the LONGEST-LENGTH operand (for reasons of compatibility).

LONGEST-LOGICAL-LEN = <integer 1..255>
Specifies the maximum length of the name matched by the wildcard.

TYPE = *TEXT(...)
Specifies that the operand value is of the data type TEXT. This data type is provided only for special purposes for Fujitsu Siemens Computers System Software Development and is therefore not described here.

TYPE = *TIME(...)
Specifies that the operand value is of the data type TIME. This is defined as a sequence of one, two or three unsigned numbers, joined together by colons (<hours>[:<minutes>[:<seconds>]]). The specifications for hours, minutes and seconds must not contain more than two digits. Numbers with less digits may be preceded by leading zeros. The value range for minutes and seconds lies between 0 and 59.

OUT-FORM = *UNCHANGED / *STD / *BINARY / *CHAR
Specifies the format that SDF uses to represent numbers in the time specification passed to the implementation.

OUT-FORM = *STD
If the transfer is made in a transfer area (see the section "Format of the standardized transfer area" on page 365), then the time specification is passed in the binary format. When passing in a string (for commands that are defined with IMPLEMENTOR= *PROCEDURE(...) or IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING,...), see ADD-CMD), the time specification is passed in the character format.

OUT-FORM = *BINARY
The time specification is passed in the binary format.

OUT-FORM = *CHAR
The time specification is passed in the character format.

TYPE = *VSN(...)
Specifies that the operand value is of data type VSN. As of V1.4A, SDF is able to distinguish between two types:

a) VSN with a period:
This VSN must consist of 6 characters. Apart from a single period, only the letters A-Z and the digits 0-9 are accepted. Such a VSN has the format `pvsid.sequence-number`, where: `pvsid` consists of 2 to 4 characters and `sequence-number` of 1 to 3

characters.
This subordinate type of VSN must not be preceded by PUB: for example, PUBA.0 or PUB.02 would be invalid. The period may be the third, fourth or fifth character of the VSN.

b) VSN without a period:
This VSN consists of a string of up to 6 characters. Only the letters A-Z, the digits 0-9 and the special characters $, @ and # are allowed. Such a VSN may start with “PUB”. In this case, the subsequent characters must not be special characters (e.g. PUB@1 or PUB$## will be rejected). SDF makes no further distinctions between public or private VSNs or PUBRES.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..6>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

TYPE = *X-STRING(...)
Specifies that the operand value is of the data type X-STRING. This is defined as a sequence of hexadecimal characters (digits 0 through 9, capital letters A through F), enclosed in apostrophes. It is prefixed by the letter X.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..1800>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

TYPE = *X-TEXT(...)
Specifies that the operand value is of the data type X-TEXT. This data type is very similar to the data type X-STRING, but it is not enclosed in single quotes and is not preceded by the letter ’X’.

SHORTEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
Specifies whether the string must have a minimum length, and if so, what that minimum length is (specified in bytes).

LONGEST-LENGTH = *UNCHANGED / *ANY / <integer 1..3600>
Specifies whether the string is not to exceed a maximum length, and if so, what that maximum length is (specified in bytes).

ODD-POSSIBLE = *UNCHANGED / *YES / *NO
Specifies whether an odd number of characters is accepted.

INTERNAL-NAME = <u>*UNCHANGED</u> / *STD / <alphanum-name 1..8>
Internal operand value name. SDF identifies an operand value by means of this name. When *STD is specified, SDF-A takes the first eight characters (omitting hyphens) of the data type specified for the TYPE operand. For operand values of the data type KEYWORD, when *STD is specified SDF-A takes the first eight characters (omitting hyphens) of the first value specified for the VALUE operand.

REMOVE-POSSIBLE = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value may be deleted (see REMOVE). If the operand value has been defined with REMOVE-POSSIBLE=*NO in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE), SDF-A rejects a change to *YES.

SECRET-PROMPT = **<u>*UNCHANGED</u> / *SAME** / ***NO**
Specifies whether the operand value is to be treated as secret. SECRET-PROMPT= *SAME assumes the setting of the operand to which the defined operand value belongs (see ADD-OPERAND ...,SECRET-PROMPT= , page 153).The input fields for values of secret operands are kept blank, and logging is suppressed.
If SECRET-PROMPT=*NO is specified, the operand value is not treated as secret.
If a secret operand value is preset to a value that is not secret, the input field is not kept blank. The input field can be kept blank by entering the character ^ or a value defined by OUTPUT=*SECRET-PROMPT.

DIALOG-ALLOWED = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value is valid in interactive mode. Specifying YES presupposes that the operand to which the value pertains is allowed in dialog mode.

DIALOG-PROC-ALLOWED = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value is allowed in interactive mode within a procedure. Specifying *YES presupposes that the operand to which the value pertains is allowed in interactive mode within a procedure.

GUIDED-ALLOWED = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value is allowed in guided dialog. Specifying *YES presupposes that the operand to which the value pertains is allowed in guided dialog.

BATCH-ALLOWED = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value is allowed in batch mode. Specifying *YES presupposes that the operand to which the value pertains is allowed in batch mode.

BATCH-PROC-ALLOWED = <u>*UNCHANGED</u> / *YES / *NO
Specifies whether the operand value is allowed in batch mode within a procedure. Specifying *YES presupposes that the operand to which the value pertains is allowed in batch mode within a procedure.

STRUCTURE =
Specifies whether the operand value introduces a structure.

STRUCTURE = <u>*UNCHANGED</u>
No change as far as introducing a structure is concerned.

STRUCTURE = *NO
The operand value does not introduce a structure.

STRUCTURE = *YES(...)
The operand value introduces a structure.

> **SIZE = <u>*UNCHANGED</u> / *SMALL / *LARGE**
> Specifies whether, at the minimum or medium level of guided dialog, the structure is to be integrated into the parent form (*SMALL), or whether a separate form is to be created for it (*LARGE).
>
> **FORM = <u>*UNCHANGED</u> / *FLAT / *NORMAL**
> Relevant only for statements and for commands defined with IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). When *FLAT is specified, the structure is linearized for the implementation in the transfer area, and the operands for the structure are integrated into a higher-ranking operand array. In the *NORMAL case, the structure receives its own operand array. In this array, those operands are passed for which a higher structure level is defined for RESULT-OPERAND-LEVEL than for the operand to which the operand value that introduces a structure and which is defined here pertains (see ADD-OPERAND ..., RESULT-OPERAND-LEVEL=,... and section “Format of the standardized transfer area” on page 365”).
>
> **MAX-STRUC-OPERAND = <u>*UNCHANGED</u> / *STD / <integer 1..3000>**
> Number of operand positions to be reserved in the structured transfer. If *STD is specified, the operand array will be made as large as necessary for the structure. However, the array may also be made larger to accommodate planned expansions. This operand refers to the structure introduced by the operand value, and is only relevant when *NORMAL was specified for the preceding operand.

LIST-ALLOWED = <u>*UNCHANGED</u> / *NO / *YES
Specifies whether a list may be specified for the operand value when the command or statement is entered. This presupposes that the operand to which the value pertains was defined with LIST-POSSIBLE=*YES (see ADD-OPERAND).

VALUE =
Specifies which values are permitted as input.

VALUE = <u>*UNCHANGED</u>
No change with regard to the permissible input values.

VALUE = *NO
All values corresponding to the operand type are permitted. Limitations exist only insofar as these have been specified in the definition of the operand type (e.g. length restrictions). For operands of the type KEYWORD, *NO is not permitted.

VALUE = list-poss(2000): <c-string 1..1800 with-low>(...)
The operand value must have one of the specified values (mandatory for values of the type KEYWORD). In contrast to the STANDARD-NAME and the ALIAS-NAME, the user may abbreviate this value on input. If the operand value is of the type KEYWORD, specification of a list is not permissible.

STANDARD-NAME = *UNCHANGED / *NO / list-poss(2000): *NAME / <structured-name 1..30> / <c-string 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the standard name of the operand value, and may be alternatively used when entering the command or statement. It must not be abbreviated when entered. In contrast to an ALIAS-NAME, a STANDARD-NAME must not be deleted so long as the operand value with this name exists in one of the assigned reference syntax files (see OPEN-SYNTAX-FILE). If the original individual value, designated a keyword in the command or program documentation, is declared to be the standard name, it is thereby ensured that the operand value can be entered using the original keyword, regardless of any changes. If *NAME is specified, SDF-A takes as STANDARD-NAME the particular value specified for the VALUE operand.

GUIDED-ABBREVIATION = *UNCHANGED / *NAME / <structured-name 1..30> / <c-string 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the name by which SDF identifies the operand value at the medium help level of guided dialog. When *NAME is specified, SDF-A takes as GUIDED-ABBREVIATION the particular value entered for the VALUE operand.

ALIAS-NAME = *UNCHANGED / *NO / list-poss(2000): <structured-name 1..30>
Relevant only for operand values of the type KEYWORD. This specifies the alias for the operand value, which may be alternatively used when the command is entered. It must not be abbreviated when entered. In contrast to a STANDARD-NAME, an ALIAS-NAME may be deleted.

MINIMAL-ABBREVIATION = *UNCHANGED / *NO / <structured-name 1..30> / <c-string 1..30>
Applies only to operand values of the type KEYWORD and defines the shortest permissible abbreviation for the operand value. Any shorter abbreviation will not be accepted by SDF.
The following should be noted:

1. Checking against the minimum abbreviation is carried out only *after* SDF has checked the input for ambiguity. It may thus happen that SDF selects the correct operand value but then rejects it because the abbreviation entered is shorter than the specified minimum abbreviation.

2. The minimum abbreviation must be derived from the KEYWORD.

3. The ALIAS-NAMEs and STANDARD-NAMEs of the operand value must not be shorter than the minimum abbreviation if they are abbreviations of the operand value.
4. The minimum abbreviation may only be shortened - not lengthened - within a syntax file hierarchy.

NULL-ABBREVIATION = <u>*UNCHANGED</u> / *NO / *YES
Relevant only for operand values of the type KEYWORD defined with STRUCTURE=*YES. This specifies whether the operand value may be omitted preceding the opening structure parentheses when the command or statement is entered, e.g. an operand value introducing a structure when there are no alternative values for the operand.

OUTPUT =
Specifies which value is passed to the implementation when OUTPUT=*NORMAL applies (see below).

OUTPUT = <u>*UNCHANGED</u>
No change with regard to the value to be passed.

OUTPUT = *SAME
The value specified for the VALUE operand is passed.

OUTPUT = *EMPTY-STRING
An empty string is passed.

OUTPUT = *DROP-OPERAND
Transfer of the operand is suppressed.

OUTPUT = <c-string 1..1800>
The value specified here is passed.

OUTPUT = <x-string 1..3600>
The value specified here is passed.

OUT-TYPE = <u>*UNCHANGED</u> / *SAME / *ALPHANUMERIC-NAME / *CAT-ID / *COMMAND-REST / *COMPOSED-NAME / *C-STRING / *DATE / *DEVICE / *FIXED / *FILENAME / *INTEGER / *KEYWORD-NUMBER / *NAME / *PARTIAL-FILENAME /*POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / *STRUCTURED-NAME / *TEXT / *TIME / *VSN / *X-STRING / *X-TEXT
Relevant only for statements and for commands defined with IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD). OUT-TYPE specifies whether SDF changes the data type of the operand value and, if so, how, when the value is stored in the transfer area for the implementation (see section “Format of the standardized transfer area” on page 365ff). If SAME is specified, SDF does not change the data type.

OVERWRITE-POSSIBLE = *UNCHANGED / *NO / *YES
Relevant only for statements and for commands defined with IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA,...) (see ADD-CMD).
OVERWRITE-POSSIBLE specifies whether the operand value entered is overwritten by a value dynamically generated by the implementation (see the DEFAULT operand in the CMDRST and CMDTST macros). One of the existing operand value definitions for the operand must cover the value generated by the implementation. In guided dialog, SDF shows the value generated by the implementation in the form.
Example: The value UNCHANGED in MODIFY statements for SDF-A is overwritten by SDF-A with the current value.

OUTPUT =
Specifies whether, and if so how, SDF is to pass the operand value entered to the implementation.

OUTPUT = *UNCHANGED
No change regarding the transfer of the operand value.

OUTPUT = *NORMAL(...)
SDF passes the operand value to the implementation. From SDF-A Version 2.0 onwards, the specifications AREA-LENGTH=, LEFT-JUSTIFIED= and FILLER= are no longer restricted to specific operand values.

AREA-LENGTH = *UNCHANGED / *VARIABLE / <integer 1..3000>
Specifies the length of the field in which SDF stores the operand value for the implementation. The field must be large enough to hold the maximum value which can be entered during execution. If the value specified for AREA-LENGTH is less than the value defined for LONGEST-LENGTH, SDF issues a warning and accepts the value specified for AREA-LENGTH.

There are two possible cases when a value that is greater than AREA-LENGTH and less than LONGEST-LENGTH is analyzed:

1. Values that are of type C-STRING with LONGEST-LENGTH ≤ 32 and which are part of a secret operand are compressed by SDF and stored in a hexadecimal string with the length defined for AREA-LENGTH. This behavior is typically used for passwords. The passwords are compressed with the aid of a hash algorithm and are protected against unauthorized access by their hexadecimal storage format.

   Notes:
   - The same hash algorithm is used as in the HASH-STRING function provided in SDF-P.
   - The command server or the program that processes the value analyzed by SDF may need to be adapted if the password definition was changed in order to support hash passwords. The hash value returned by SDF may otherwise be rejected by the semantic analysis module of the program or command server.

2. In all other cases, i.e. those which deviate from case 1, the value is truncated to the length specified for AREA-LENGTH.

LEFT-JUSTIFIED = *UNCHANGED / *STD / *YES / *NO
Relevant only when a fixed length has been defined for the field in which the operand value is stored. LEFT-JUSTIFIED specifies whether SDF stores the operand value in the field left-justified or right-justified. SDF-A changes *STD to *NO for numeric values and to *YES for all other values.

FILLER = *UNCHANGED / *STD / <c-string 1..1> / <x-string 1..2>
Relevant only when a fixed length has been defined for the field in which the operand value is stored. FILLER specifies the character with which SDF pads the field when necessary. SDF-A changes *STD to X’00’ for values of the types X-STRING or INTEGER, and to C’␣’ (blank) for all other values.

STRING-LITERALS = *UNCHANGED / *NO / *HEX / *CHAR
Specifies whether SDF converts the operand value into the data type X-STRING or C-STRING for the transfer to the implementation. When *NO is specified, SDF does not change the data type. It must then be borne in mind that, for operand values of the type C-STRING, SDF transfers only the contents of the string (without the apostrophes and the prefixed C). This operand is valid only if VALUE=*NO is specified.

HASH = *UNCHANGED / * NO / * YES(...)
Specifies whether the input value can be converted to a value with a defined length using a hash algorithm.

HASH = *YES(...)
Is only permitted for operand values of the data type C-STRING which are defined with LONGEST-LENGTH ≤ 32.
The other operands in the structure OUTPUT=*NORMAL(..) do not format the value until after the hash function has been carried out. The value then has the data type X-STRING and may then contain non-printable characters.

OUTPUT-LENGTH = <integer 2..32>
Length of the value to which the input value is changed.

OUTPUT = *SECRET-PROMPT
The operand value is not passed to the implementation; instead it causes SDF to request the user to enter one of the alternative values for the operand. The input that follows is then not displayed and is not logged. Prerequisites for this are:
– The operand is defined as secret (see ADD-OPERAND...,SECRET-PROMPT=*YES)
– Input is made in unguided dialog or in a foreground procedure
– A single value is specified as permissible input for the operand value defined with SECRET-PROMPT (see the operand VALUE=<c-string>; normally the value is of the type KEYWORD).

The following case occurs in guided dialog:
The input field for a secret operand which is set to a value which is not secret will not be kept blank. The input field can be kept blank by entering a value defined with OUTPUT=*SECRET-PROMPT.

PRIVILEGE = *UNCHANGED / *SAME / *EXCEPT(...) /
list-poss(64): <structured-name 1..30>
Specifies the privileges assigned to the operand value.

PRIVILEGE = *SAME
The operand value is assigned the same privileges as the operand to which it belongs.

PRIVILEGE = *EXCEPT(...)
With the exception of those defined with *EXCEPT(...), all privileges currently defined and all subsequently defined privileges are assigned to the operand value.

EXCEPT-PRIVILEGE = list-poss(64): <structured-name 1..30>
Specifies the privileges that are not assigned to the operand value.

PRIVILEGE = list-poss(64): <structured-name 1..30>
Only the privileges specified in this list are assigned to the operand value.

OPEN-SYNTAX-FILE
Open syntax file

The OPEN-SYNTAX-FILE statement is used to open a syntax file for processing with SDF-A. The format of this syntax file can be defined with the DEFINE-ENVIRONMENT statement before opening the file. Each subsequent OPEN-SYNTAX-FILE statement implicitly causes SDF-A to close the previously opened syntax file.

A syntax file can be opened either with (default) or without a hierarchy. Any modification of the hierarchy between two operations may lead to errors. It is therefore advisable to edit a syntax file always in the same hierarchy.

(part 1 of 2)

```
OPEN-SYNTAX-FILE

 FILE = <filename 1..54>
,TYPE = *USER (...) / *GROUP(...) / *SYSTEM(...)
    *USER(...)
      |   GROUP-DESCRIPTIONS = *CURRENT / *NO / <filename 1..54>
      |  ,SYSTEM-DESCRIPTIONS = *CURRENT / *NO / <filename 1..54>
      |  ,USER-CONTROL = *NO / <filename 1..54>
    *GROUP(...)
      |   SYSTEM-DESCRIPTIONS = *CURRENT / *NO / <filename 1..54>
      |  ,SYSTEM-CONTROL = *NO / <filename 1..54>
    *SYSTEM(...)
      |   SYSTEM-CONTROL = *NO / <filename 1..54>
,MODE = *UPDATE (...) / *CREATE / *READ / *INIT(...)
    *UPDATE(...)
      |   COMPONENT-VERSION = *UNCHANGED / <integer 0..999>
      |  ,SOFTWARE-UNIT-NAME = *NOCHECK (...) / <structured-name 1..15>(...)
      |     *NOCHECK(...)
      |       |   VERSION = *UNCHANGED / <product-version>
      |     <structured-name 1..15>(...)
      |       |   VERSION = *UNCHANGED / <product-version>
```

continued ➟

(part 2 of 2)

```
*INIT(...)
   |  KERNEL = <filename 1..54>
   | ,MONSYS-DOMAIN = <structured-name 1..13>
   | ,COMMAND-CLASS = <name 3..3>(...)
   |     <name 3..3>(...)
   |        |  SOFTWARE-UNIT-NAME = <structured-name 1..15>(...)
   |        |     <structured-name 1..15>(...)
   |        |        |  VERSION = <product-version>
   | ,COMPONENT-VERSION = <integer 0..999>
```

FILE = <filename 1..54>
Name of the syntax file to be opened.

TYPE =
Type of the syntax file to be opened.

TYPE = <u>*USER</u>(...)
A user syntax file is to be opened.

GROUP-DESCRIPTIONS =
Specifies whether SDF-A is to access the contents of a group syntax file when processing the user syntax file to be opened.

GROUP-DESCRIPTIONS = <u>*CURRENT</u>
SDF-A is to access the group syntax file activated for the current task.

GROUP-DESCRIPTIONS = *NO
SDF-A is not to access a group syntax file.

GROUP-DESCRIPTIONS = <filename 1..54>
Name of the group syntax file to be accessed by SDF-A.

SYSTEM-DESCRIPTIONS =
Specifies whether SDF-A is to access the contents of a system syntax file when processing the user syntax file to be opened.

SYSTEM-DESCRIPTIONS = <u>*CURRENT</u>
SDF-A is to access the currently activated system syntax file.

SYSTEM-DESCRIPTIONS = *NO
SDF-A is not to access a system syntax file.

SYSTEM-DESCRIPTIONS = <filename 1..54>
Name of the system syntax file to be accessed by SDF-A.

USER-CONTROL =
Specifies whether SDF-A is to check that definitions of user-defined commands or statements are not modified in ways that are not permissible. For this monitoring, SDF-A requires a user syntax file in which the definitions of the user-defined commands/statements are stored in their original versions.

USER-CONTROL = <u>*NO</u>
No monitoring is to take place.

USER-CONTROL = <filename 1..54>
Name of the user syntax file containing the command/statement definitions required for monitoring.

TYPE = *GROUP(...)
A group syntax file is to be opened.

SYSTEM-DESCRIPTIONS =
Specifies whether SDF-A is to access the contents of a system syntax file when processing the group syntax file to be opened.

SYSTEM-DESCRIPTIONS = <u>*CURRENT</u>
SDF-A is to access the currently activated system syntax file.

SYSTEM-DESCRIPTIONS = *NO
SDF-A is not to access a system syntax file.

SYSTEM-DESCRIPTIONS = <filename 1..54>
Name of the system syntax file to be accessed by SDF-A.

SYSTEM-CONTROL =
Specifies whether SDF-A is to check that definitions of commands or statements available to specific user IDs are not modified in ways that are not permissible. For this monitoring, SDF-A requires a system syntax file in which the definitions of these commands/statements are stored in their original versions.

SYSTEM-CONTROL = <u>*NO</u>
No monitoring is to take place.

SYSTEM-CONTROL = <filename 1..54>
Name of the system syntax file containing the command/statement definitions required for monitoring.

TYPE = *SYSTEM(...)
A system syntax file is to be opened.

SYSTEM-CONTROL =
Specifies whether SDF-A is to check that definitions of commands or statements available throughout the system are not modified in ways that are not permissible. For this check, SDF-A requires as a reference syntax file a system syntax file in which the definitions of these commands/statements are stored in their original versions.

SYSTEM-CONTROL = <u>*NO</u>
No monitoring is to take place.

SYSTEM-CONTROL = <filename 1..54>
Name of the system syntax file containing the command/statement definitions required for monitoring.

MODE =
Specifies whether the opened syntax file may be processed or whether it is to be created.

MODE = <u>*UPDATE</u>(...)
The contents of the syntax file may be both displayed and modified. The syntax file already exists. It must not have been activated.

COMPONENT-VERSION = *UNCHANGED / <integer 0..999>
This operand is reserved for Fujitsu Siemens Computers Software Development.

SOFTWARE-VERSION = *NOCHECK(...) / <structured-name 1..15>(...)
This operand is reserved for Fujitsu Siemens Computers Software Development.

MODE = *CREATE
The contents of the syntax file may be both displayed and modified. The file is to be newly created by SDF-A. This also means that SDF-A generates the global information. If a file with the same name already exists, SDF-A refuses to open the file and issues an error message.

MODE = *READ
The contents of the syntax file can be displayed, but not modified (read access only). The syntax file already exists. It may already have been activated.

MODE = *INIT(...)
This operand value is reserved for Fujitsu Siemens Computers Software Development.

REMOVE
Delete objects from syntax file

The REMOVE statement is used to delete objects from the open syntax file. Objects in this sense are domains, programs, commands, statements, operands or operand values. The term “delete” as used below implies two operations:

1. SDF-A removes the definition of the object to be deleted from the open syntax file. This applies to all types of syntax files.
2. SDF-A writes into the processed user or group syntax file the information that the object to be deleted is disabled. This presupposes that, when the syntax file was opened, a group or system syntax file in which the object to be deleted is defined was specified for the GROUP-DESCRIPTIONS or SYSTEM-DESCRIPTIONS operand. In this case the command name cannot be used for another command.

The removal of mandatory operands from a system command is not allowed. This prevents a permanent syntax error during execution.

The standard SDF statements cannot be deleted.

REMOVE OBJECT=*DOMAIN removes also the commands assigned exclusively to this domain, unless the command has been defined with REMOVE-POSSIBLE=*NO.

An object defined with REMOVE-POSSIBLE=*NO (in ADD or MODIFY statements) cannot be deleted.

(part 1 of 4)

```
REMOVE

OBJECT = *PRIVILEGE(...) / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) / *STATEMENT(...) /
         *OPERAND(...) / *VALUE(...) / *CORRECTION-INFORMATION(...)

  *PRIVILEGE(...)
   |  NAME = <structured-name 1..30>

  *DOMAIN(...)
   |  NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
   |     *ALL(...)
   |      |  EXCEPT = *NONE / <structured-name 1..30 with-wild> /
   |      |           list-poss(2000): <structured-name 1..30>
   | ,REMOVE-ATT-COMMANDS = *YES / *NO
```

continued ➠

(part 2 of 4)

```
*COMMAND(...)
   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
     *ALL(...)
        EXCEPT = *NONE / <structured-name 1..30 with-wild> /
                    list-poss(2000): <structured-name 1..30>
*PROGRAM(...)
   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
     *ALL(...)
        EXCEPT = *NONE / <structured-name 1..30 with-wild> /
                    list-poss(2000): <structured-name 1..30>
*STATEMENT(...)
   NAME = *ALL(...) / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
     *ALL(...)
        EXCEPT = *NONE / <structured-name 1..30 with-wild> /
                    list-poss(2000): <structured-name 1..30>
  ,PROGRAM = <structured-name 1..30>
*OPERAND(...)
   OPERAND-L1 = *CURRENT / <structured-name 1..20>
  ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
               *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
               *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
               *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
               *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
  ,OPERAND-L2 = *NO / <structured-name 1..20>
  ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
               *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
               *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
               *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
               *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
```

continued ➠

(part 3 of 4)

```
      ,OPERAND-L3 = *NO / <structured-name 1..20>
      ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
      ,OPERAND-L4 = *NO / <structured-name 1..20>
      ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
      ,OPERAND-L5 = *NO / <structured-name 1..20>
      ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
        *COMMAND(...)
           NAME = <structured-name 1..30>
        *STATEMENT(...)
           NAME = <structured-name 1..30>
          ,PROGRAM = <structured-name 1..30>
   *VALUE(...)
       OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
      ,VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
      ,OPERAND-L2 = *NO / <structured-name 1..20>
      ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
```

continued ➠

(part 4 of 4)

```
      ,OPERAND-L3 = *NO / <structured-name 1..20>

      ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>

      ,OPERAND-L4 = *NO / <structured-name 1..20>

      ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>

      ,OPERAND-L5 = *NO / <structured-name 1..20>

      ,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                  *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                  *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                  *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                  *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>

      ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)

         *COMMAND(...)

            NAME = <structured-name 1..30>

         *STATEMENT(...)

            NAME = <structured-name 1..30>

           ,PROGRAM = <structured-name 1..30>

   *CORRECTION-INFORMATION(...)

      PM-NUMBER = *ALL / list-poss(20): <alphanum-name 8..8>
```

OBJECT =
Type of the object to be deleted.

OBJECT = *PRIVILEGE(...)
Specifies that a privilege is to be deleted. This operand value is reserved for Fujitsu Siemens Computers Software Development.

OBJECT = *DOMAIN(...)
Specifies that domains are to be deleted. The commands assigned to these domains are *not* deleted if:
– they are assigned to at least one other domain,
– they are defined with REMOVE-POSSIBLE=*NO or
– the user specifies REMOVE-ATT-COMMANDS=*NO.

Domains do not contain statements, which means that no statements can be deleted when a domain is deleted.

NAME = *ALL(...)
All domains are deleted.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The domains specified here are not deleted.

NAME = <structured name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The domains named or the domains whose name matches the name of the wildcard selector are deleted.

REMOVE-ATT-COMMANDS = <u>*YES</u> / *NO
Specifies whether the commands assigned to the domain are deleted.

OBJECT = *COMMAND(...)
Specifies that commands are to be deleted. This includes the deletion of the associated operands.

NAME = *ALL(...)
All commands are deleted.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The commands specified here are not deleted.

NAME = <structured name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The commands named or the commands whose name matches the name of the wildcard selector are deleted.

OBJECT = *PROGRAM(...)
Specifies that programs are to be deleted. This includes deleting the associated statements.

NAME = *ALL(...)
All programs are deleted.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The programs specified here are not deleted.

NAME = <structured name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The programs named or the programs whose name matches the name of the wildcard selector are deleted.

OBJECT = *STATEMENT(...)
Specifies that statements are to be deleted. This includes deleting the associated operands.

NAME = *ALL(...)
All statements are deleted.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The statements specified here are not deleted.

NAME = <structured name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The statements / the statements whose name matches the name of the wildcard selector are deleted.

PROGRAM = <structured-name 1..30>
Name of the program to which the statements belong.

OBJECT = *OPERAND(...)
Specifies that an operand is to be deleted. This includes deleting the associated operand values. If this operand is included in a structure, it is specified by the path leading to it, i.e. by specifying the operands and operand values that introduce the structure in this path. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely (or not at all) specified. An operand that is not absolutely essential to identify the operand to be deleted, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first VALUE-Li=*NO, SDF-A takes the operand defined by OPERAND-Li as the one that is to be deleted. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for VALUE-Li, the value specified for OPERAND-Li + 1 must also be other than *NO.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Specifies the operand to be deleted (VALUE-L1 = *NO) or an operand in the path leading to it (VALUE-L1 ≠ *NO). *CURRENT means that OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
*NO means that OPERAND-L1 is to be deleted. Otherwise, an operand value that introduces a structure is to be specified. This structure must directly or indirectly contain the operand that is to be deleted. If the operand value introducing the structure is of the data type KEYWORD-NUMBER, then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value introducing the structure is not of the type KEYWORD-NUMBER, then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that OPERAND-L2 is irrelevant for the specification of the operand to be deleted. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L1 is to be specified. This operand is either the one to be deleted (VALUE-L2 = *NO) or an operand that is in the path leading to the operand to be deleted (VALUE-L2 ≠ *NO).

VALUE-L2 = analogous to VALUE-L1
*NO means that VALUE-L2 is irrelevant for the specification of the operand to be deleted. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand to be deleted. For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand that is to be deleted. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L2 is to be specified. This operand is either the one to be deleted (VALUE-L3 = *NO) or an operand that is in the path leading to the operand to be deleted (VALUE-L3 ≠ *NO).

VALUE-L3 = analogous to VALUE-L1
*NO means that VALUE-L3 is irrelevant for the specification of the operand to be deleted. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand that is to be deleted. For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L2.

VALUE-L4= analogous to VALUE-L1
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L2.

ORIGIN =
Specifies the command or statement to which the operand to be deleted belongs.

ORIGIN = *CURRENT
The operand pertains to a command or statement that currently either is itself the current object or contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)
The operand pertains to a command.

> **NAME = <structured-name 1..30>**
> Name of the command.

ORIGIN = *STATEMENT(...)
The operand pertains to a statement.

> **NAME = <structured-name 1..30>**
> Name of the statement.
>
> **PROGRAM = <structured-name 1..30>**
> Name of the program to which the statement pertains.

OBJECT = *VALUE(...)
Specifies that an operand value is to be deleted. This includes deleting the associated structures. The operand value to be deleted is specified by the path leading to it, i.e. by specifying the operands and operand values introducing a structure in this path. If the operand value to be deleted pertains to an operand that is not attached to any structure, the path contains only this operand. If the operand value to be deleted does pertain to an operand attached to a structure, the path also includes the higher-ranking operands as well as the associated operand values introducing the structure. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified. An operand that is not absolutely essential to identify the operand value to be deleted, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first OPERAND-Li + 1 = *NO, SDF-A takes the operand value defined by VALUE-Li as the one whose definition is to be deleted. Subsequently, SDF-A does not interpret the specifications as to any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for OPERAND-Li, the value specified for VALUE-Li must likewise be other than *NO.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Specifies the operand to which the operand value to be deleted pertains (OPERAND-L2 = *NO) or an operand in the path leading to this operand value (OPERAND-L2 ≠ *NO). *ABOVE-CURRENT means that a value pertaining to OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *POSIX-PATHNAME / *POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
Specifies the operand value to be deleted (OPERAND-L2=*NO) or an operand value introducing a structure in the path leading to the operand to be deleted (OPERAND-L2≠*NO). *CURRENT means that VALUE-L1 is the current object. If it is not the current object and of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Here it must be remembered that this particular value is to be specified in each case without the prefixed asterisk. If the operand value is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that the definition of VALUE-L1 is to be deleted. Otherwise, the name of the operand to which the operand value that is to be deleted pertains (OPERAND-L3 = *NO) or the name of an operand in the path leading to the operand value to be deleted (OPERAND-L3 ≠ *NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L1.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *POSIX-PATHNAME / * POSIX-FILENAME / *PRODUCT-VERSION / <composed-name 1..30>
*NO means that the VALUE-L2 is irrelevant for the specification of the operand value to be deleted. Otherwise, an operand value is to be specified. This operand value is either the one that is to be deleted (OPERAND-L3 = *NO) or an operand value introducing a structure in the path leading to the operand value to be deleted (OPERAND-L3 ≠ *NO). For further information see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand value to be deleted. Otherwise, the name of the operand to which the operand value that is to be deleted pertains (OPERAND-L4 = *NO) or the name of an operand in the path leading to the operand value to be deleted (OPERAND-L4 ≠ *NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L2.

VALUE-L3 = analogous to VALUE-L2
*NO means that VALUE-L3 is irrelevant for the specification of the operand value to be deleted. Otherwise, an operand value is to be specified. This operand value is either the one that is to be deleted (OPERAND-L4 = *NO) or an operand value introducing a structure in the path leading to the operand value to be deleted (OPERAND-L4 ≠ *NO). For further information see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L4 = analogous to VALUE-2
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L5 = analogous to VALUE-2
see VALUE-L2.

ORIGIN =

Specifies the command or statement to which the operand value to be deleted pertains.

ORIGIN = <u>*CURRENT</u>

The operand value to be deleted belongs to a command (or statement) that currently either is itself the current object or else contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)

The operand value pertains to a command.

NAME = <structured-name 1..30>

Name of the command.

ORIGIN = *STATEMENT(...)

The operand value pertains to a statement.

NAME = <structured-name 1..30>

Name of the statement.

PROGRAM = <structured-name 1..30>

Name of the program to which the statement pertains.

OBJECT = *CORRECTION-INFORMATION(...)

Reserved for Fujitsu Siemens Computers Software Development.

RESTORE
Restore objects of syntax file

The RESTORE statement is used to lift the lock on objects in the open syntax file. The lock was set with the aid of the REMOVE statement; it can be lifted for commands, statements and programs. Their definitions must be included in the syntax file to be edited or in a syntax file which is higher up in the file hierarchy. Physically deleted objects cannot be restored. The RESTORE statement cannot be used if the syntax file being processed is the system syntax file.

```
RESTORE

OBJECT = *COMMAND(...) / *PROGRAM(...) / *STATEMENT(...)
   *COMMAND(...)
     |  NAME = *ALL / list-poss(2000): <structured-name 1..30>
   *PROGRAM(...)
     |  NAME = *ALL / list-poss(2000): <structured-name 1..30>
   *STATEMENT(...)
     |  NAME = *ALL / list-poss(2000): <structured-name 1..30>
     | ,PROGRAM = <structured-name 1..30>
```

OBJECT =
Type of object for which the lock is to be lifted.

OBJECT = *COMMAND(...)
Specifies that the lock is to be lifted for commands.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
The lock is lifted for all commands or for those specified by name.

OBJECT = *STATEMENT(...)
Specifies that the lock is to be lifted for statements.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
The lock is lifted for all statements or for those specified by name.

PROGRAM = <structured-name 1..30>
Name of the program to which the statements belong.

OBJECT = *PROGRAM(...)
Specifies that the lock is to be lifted for programs. In this case, the lock is not lifted for the program itself, but for all statements belonging to the program.

NAME = *ALL / list-poss(2000): <structured-name 1..30>
The lock is lifted for all programs or for those specified by name.

SET-GLOBALS
Modify global information

The global information includes general specifications relating to command/statement input and processing. These specifications take effect as soon as the syntax file is activated. The prevailing specifications can be changed for a specific task by means of the MODIFY-SDF-OPTIONS command or statement.

Normally, SDF-A generates the global information when a syntax file is first opened. The resulting specifications can be changed using the SET-GLOBALS statement. When working in guided dialog with the SET-GLOBALS statement, the operands with the current values are preset in the form, provided global information is the current object (see EDIT). Otherwise, presetting is performed with the default value *UNCHANGED.

(part 1 of 11)

```
SET-GLOBALS

 VERSION = *UNCHANGED / <alphanum-name 1..12> / <c-string 1..12>
,GUIDANCE = *UNCHANGED / *STD / *MAXIMUM / *MEDIUM / *MINIMUM / *NO / *EXPERT
,LOGGING = *UNCHANGED / *STD / *INPUT-FORM / *ACCEPTED-FORM / *INVARIANT-FORM
,PROCEDURE-DIALOG = *UNCHANGED / *STD / *YES / *NO
,UTILITY-INTERFACE = *UNCHANGED / *STD / *OLD / *NEW
,CONTINUATION = *UNCHANGED / *STD / *OLD / *NEW
,FUNCTION-KEYS = *UNCHANGED / *STD / *OLD-MODE / *STYLE-GUIDE-MODE / *BY-TERMINAL-TYPE
,INPUT-HISTORY = *UNCHANGED / *STD / *ON / *OFF
,NUMBER-OF-INPUTS = *UNCHANGED / *STD / <integer 1..100>
,GENERAL-INFO-VERSION = *UNCHANGED / <integer 1..255>
,MODIFY-LANGUAGE-TEXT = *UNCHANGED / list-poss(2000): <name 1..1>(...)

   <name 1..1>(...)
   |
   |    COMMAND-REST = *NAMES (...)
   |
   |      *NAMES(...)
   |      |
   |      |    LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
   |      |
   |      |   ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
   |      |
   |      |   ,HELP = *UNCHANGED / <c-string 1..100 with-low>
```

continued ➠

(part 2 of 11)

```
   ,INTEGER = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..100 with-low>

   ,X-STRING = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..150 with-low>

   ,C-STRING = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..250 with-low>

   ,NAME = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..100 with-low>

   ,ALPHANUM-NAME = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..100 with-low>

   ,STRUCTURED-NAME = *NAMES (...)

     *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..250 with-low>
```

continued ➠

(part 3 of 11)

,**LABEL** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..150 with-low>

,**VSN** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**FILENAME** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..700 with-low>

,**PAR**TIAL**-FILENAME** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..400 with-low>

,**TIME** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

,**DATE** = ***NAMES** (...)

***NAMES**(...)

LONG-TEXT = ***UNCH**ANGED / <c-string 1..20 with-low>

,**ABBREV**IATION = ***UNCH**ANGED / <c-string 1..10 with-low>

,**HELP** = ***UNCH**ANGED / <c-string 1..250 with-low>

continued ➟

(part 4 of 11)

```
,COMPOSED-NAME = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..350 with-low>

,TEXT = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..350 with-low>

,CAT-ID = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..100 with-low>

,PRODUCT-VERSION = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

       ,HELP = *UNCHANGED / <c-string 1..250 with-low>

,POSIX-PATHNAME = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..20 with-low>

       ,HELP = *UNCHANGED / <c-string 1..250 with-low>

,POSIX-FILENAME = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..20 with-low>

       ,HELP = *UNCHANGED / <c-string 1..250 with-low>
```

continued ➠

(part 5 of 11)

```
,AMBIGUOUS-OPERATIONS = *UNCHANGED / <c-string 1..50>
,X-TEXT = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
       ,HELP = *UNCHANGED / <c-string 1..150 with-low>
,FIXED = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
       ,HELP = *UNCHANGED / <c-string 1..250 with-low>
,WILDCARD = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,LOWER-CASE = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,USER-ID = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,GENERATION = *NAMES (...)
   *NAMES(...)
        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
```

continued ➠

(part 6 of 11)

```
,VERSION = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,UNDERSCORE = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,SEPARATORS = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,ODD-POSSIBLE = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,COMPLETION = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,EXIT = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,TEMPORARY-FILE = *NAMES (...)

   *NAMES(...)

        LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

       ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
```

continued ➠

(part 7 of 11)

```
 ,QUOTES-MANDATORY = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..11 with-low>
 ,LONGEST-LOGICAL-LEN = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
 ,USER-INTERFACE = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
 ,CORRECTION-STATE = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
 ,PATH-COMPLETION = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
 ,WILDCARD-CONSTRUCT = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..20 with-low>
 ,REFRESH = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
```

continued ➠

(part 8 of 11)

```
 ,REST-SDF-IN = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..11 with-low>
 ,EXIT-ALL = *NAMES (...)
    *NAMES(...)
       LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
      ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
 ,DOMAIN = *UNCHANGED / <c-string 1..11>
 ,COMMAND = *UNCHANGED / <c-string 1..11>
 ,PROGRAM = *UNCHANGED / <c-string 1..11>
 ,STATEMENT = *UNCHANGED / <c-string 1..13>
 ,KEYS = *UNCHANGED / <c-string 1..13>
 ,STRUCTURE = *UNCHANGED / <c-string 1..11>
 ,SITUATION = *UNCHANGED / <c-string 1..11>
 ,COMMENT-LINE = *UNCHANGED / <c-string 1..50>
 ,PROCEDURE-ERROR = *UNCHANGED / <c-string 1..60>
 ,INTERNAL-ERROR = *UNCHANGED / <c-string 1..60>
 ,PROC-HELP = *UNCHANGED / <c-string 1..80>
 ,INTERNAL-HELP = *UNCHANGED / <c-string 1..80>
 ,CORRECT-COMMAND = *UNCHANGED / <c-string 1..80>
 ,CORRECT-STATEMENT = *UNCHANGED / <c-string 1..80>
 ,OPERANDS = *UNCHANGED / <c-string 1..11>
 ,DOMAIN-TITLE = *UNCHANGED / <c-string 1..80>
 ,COMMAND-TITLE = *UNCHANGED / <c-string 1..80>
 ,STATEMENT-TITLE = *UNCHANGED / <c-string 1..80>
 ,DIRECT-EXECUTION = *UNCHANGED / <c-string 1..27>
 ,OR = *UNCHANGED / <c-string 1..10 with-low>
 ,DEFAULT = *UNCHANGED / <c-string 1..10 with-low>
 ,DEFAULT-BY-JV = *UNCHANGED / <c-string 1..20 with-low>
```

continued ➠

(part 9 of 11)

```
,DEFAULT-BY-VAR = *UNCHANGED / <c-string 1..20 with-low>
,ALTERNATE-DEFAULT = *UNCHANGED / <c-string 1..20 with-low>
,MANDATORY = *UNCHANGED / <c-string 1..20 with-low>
,WITH = *UNCHANGED / <c-string 1..10 with-low>
,WITHOUT = *UNCHANGED / <c-string 1..10 with-low>
,LIST-POSSIBLE = *UNCHANGED / <c-string 1..20 with-low>
,STRUCTURE-INCLUDED = *UNCHANGED / <c-string 1..20>
,NEXT = *UNCHANGED / <c-string 1..10>
,MESSAGE = *UNCHANGED / <c-string 1..10>
,ERROR = *UNCHANGED / <c-string 1..10>
,NUMBER = *NAMES (...)
   *NAMES(...)
      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,NEXT-CMD = *NAMES (...)
   *NAMES(...)
      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,NEXT-STMT = *NAMES (...)
   *NAMES(...)
      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,NEXT-DATA = *NAMES (...)
   *NAMES(...)
      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
,NEXT-INPUT = *NAMES (...)
   *NAMES(...)
      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
```

continued ➠

(part 10 of 11)

```
,NEXT-DOMAIN = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,DOWN-OPERAND = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,DOMAIN-MENU = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,UP = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,DOWN = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,TEST = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>

,EXECUTE = *NAMES (...)

   *NAMES(...)

      LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>

     ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
```

continued ➠

(part 11 of 11)

```
    ,CONTINUE = *NAMES (...)
       *NAMES(...)
          LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
         ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
    ,CANCEL = *NAMES (...)
       *NAMES(...)
          LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
         ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
    ,RESTORE = *NAMES (...)
       *NAMES(...)
          LONG-TEXT = *UNCHANGED / <c-string 1..20 with-low>
         ,ABBREVIATION = *UNCHANGED / <c-string 1..10 with-low>
    ,ENTER-COMMAND = *UNCHANGED / <c-string 1..80>
    ,ENTER-STATEMENT = *UNCHANGED / <c-string 1..80>
    ,ENTER-CONTINUATION = *UNCHANGED / <c-string 1..80>
    ,ENTER-OPERANDS = *UNCHANGED / <c-string 1..80>
    ,CORRECT-OPERATION = *UNCHANGED / <c-string 1..80>
    ,SECRET-OPERAND = *UNCHANGED / <c-string 1..80>
    ,PROC-INTERRUPT = *UNCHANGED / <c-string 1..80>
    ,PROC-INT-YES = *UNCHANGED / <c-string 1..10>
    ,PROC-INT-NO = *UNCHANGED / <c-string 1..10>
,REMOVE-LANGUAGE-TEXT = *NO / list-poss(2000): <name 1..1>
```

VERSION =
Defines the version number/name of the syntax file; useful only for documentation purposes.

VERSION = *UNCHANGED
The version number remains unchanged.

VERSION = <alphanum-name 1..12> / <c-string 1..12>
The syntax file receives the specified version number or the specified name as the version name. Blanks at the end of a c-string are ignored.

GUIDANCE =
Specifies the level of user guidance. (This definition does not apply to jobs started via OMNIS; for these, EXPERT is always assumed.)

GUIDANCE = <u>*UNCHANGED</u>
The specification in the global information regarding user guidance remains unchanged.

GUIDANCE = *STD
User guidance remains unchanged when the processed user or group syntax file is activated. For a system syntax file, *STD has the same effect as *NO.

GUIDANCE = *MAXIMUM
Menus with explanatory texts are displayed for the selection of domains, commands and statements. Forms are displayed for the assignment of values to operands. There is a separate form for each structure. The forms contain help texts for the operands, the default values, all permissible operand values, and additional information regarding these values.

GUIDANCE = *MEDIUM
Menus with explanatory texts are displayed for the selection of domains, commands and statements. Forms are displayed for the assignment of values to operands. There is a separate form for each structure whose introductory value is defined with SIZE=*LARGE (see ADD-VALUE). The forms contain the default values and all permissible operand values.

GUIDANCE = *MINIMUM
Menus are displayed for the selection of domains, commands and statements. Forms are displayed for the assignment of values to operands. There is a separate form for each structure whose introductory value is defined with SIZE=*LARGE (see ADD-VALUE). The forms contain only the default values.

GUIDANCE = *NO
Input is requested with the text %CMD: or %STMT:. Several commands may be entered in a block, one after the other, each separated from the next by a “logical end-of-line”. It is possible to correct incorrect input.

GUIDANCE = *EXPERT
Input is requested with / or //. Several commands may be entered in a block, one after the other, each one separated from the next by a “logical end-of-line”. Correction of incorrect input is not possible.

LOGGING =
Specifies how input is to be logged.

LOGGING = <u>*UNCHANGED</u>
The specification in the global information regarding logging remains unchanged.

LOGGING = *STD
Logging remains unchanged if the processed user or group syntax file is activated. For a system syntax file, *STD has the same effect as *INPUT-FORM.

LOGGING = *INPUT-FORM
In unguided dialog input strings are logged exactly as entered. Passwords are masked out. In guided dialog or in error dialog, logging is carried out in the same way as with *ACCEPTED-FORM.

LOGGING = *ACCEPTED-FORM
The following are logged:
– all names in their unabbreviated form
– each operand appearing in the input, with its name and the value specified
– the final input, reflecting any corrections.

Passwords are masked out. Entries in guided dialog are concatenated to form a string.

LOGGING = *INVARIANT-FORM
The following are logged:
– all names in the versions defined in the syntax file as STANDARD-NAME (i.e. the names specified in the manuals)
– each operand appearing in the input, with its name and the value specified
– all optional operands implicit in the input, with their default values
– the final input, reflecting any corrections.

Passwords are masked out. Input entered in guided dialog is concatenated to form a string.

PROCEDURE-DIALOG =
Specifies whether the user is to be requested to make interactive corrections when syntax or semantic errors occur with a SYSSTMT file or a procedure executing in interactive mode.

PROCEDURE-DIALOG = <u>*UNCHANGED</u>
The specification in the global information regarding interactive corrections remains unchanged.

PROCEDURE-DIALOG = *STD
The rule governing interactive corrections remains unchanged if the processed user or group syntax file is activated. For a system syntax file, *STD has the same effect as *NO.

PROCEDURE-DIALOG = *YES
The user is requested to make interactive corrections.

PROCEDURE-DIALOG = *NO
The user is not requested to make interactive corrections.

UTILITY-INTERFACE =
Sets a switch that can be checked by a program with the CMDSTA macro. By means of this switch, the type of statement input can be controlled for programs that can read their statements both with RDATA as well as via SDF with CMDRST.

UTILITY-INTERFACE = *UNCHANGED
The specification in the global information regarding the switch remains unchanged.

UTILITY-INTERFACE = *STD
The switch remains unchanged if the processed user or group syntax file is activated. For a system syntax file, *STD has the same effect as *NEW.

UTILITY-INTERFACE = *OLD
Programs are to read their statements using RDATA.

UTILITY-INTERFACE = *NEW
Programs are to read their statements via SDF using CMDRST.

CONTINUATION =
Specifies in which column the continuation character “-” is to be entered in the case of command input (SYSCMD), if required. (For statement input (SYSSTMT), the continuation character may be entered in any column.)

CONTINUATION = *UNCHANGED
The specification in the global information regarding the continuation character remains unchanged.

CONTINUATION = *STD
The rule governing continuation characters remains unchanged if the processed user or group syntax file is activated. For a system syntax file, specifying *STD means that the specification made at system generation applies.

CONTINUATION = *OLD
The continuation character must be entered in column 72.

CONTINUATION = *NEW
The continuation character may be entered anywhere in columns 2 through 72.

FUNCTION-KEYS =
Defines function key assignments. A detailed description of the different modes can be found in the “Introductory Guide to the SDF Dialog Interface” [1]. Unsupported function keys do nothing; they do not have the same effect as the DUE or SEND key.

FUNCTION-KEYS = *UNCHANGED
The function key assignments defined in the global information are not changed. Unsupported function keys do nothing; they do not have the same effect as the DUE or SEND key.

FUNCTION-KEYS = *STD
The existing setting for the option is retained when the processed user or syntax file is activated. In the case of a system syntax file, *STD has the same effect as *BY-TERMINAL-TYPE.

FUNCTION-KEYS = *OLD-MODE
Function keys assignments correspond to the old mode, which is supported by all terminal types. The following key assignments apply:

K1	Exit function
K2	Interrupt function
K3	Refresh function (only in guided dialog)
F1	Exit-all function
F2	Test function (only in guided dialog)
F3	Execute function (only in guided dialog)

FUNCTION-KEYS = *STYLE-GUIDE-MODE
Function keys are assigned in accordance with the Fujitsu Siemens style guide. The following key assignments apply:

K2	Interrupt function
F1	Help function
F3	Exit function
F5	Refresh function (only in guided dialog)
F6	Exit-all function
F7	Page backward (only in guided dialog)
F8	Page forward (only in guided dialog)
F9	Execute RESTORE-SDF-INPUT INPUT=*LAST
F11	Execute function (only in guided dialog)
F12	Cancel function

FUNCTION-KEYS = *BY-TERMINAL-TYPE
The assignment of function keys depends on the type of terminal. If the terminal type supports the more comprehensive functionality of the Fujitsu Siemens style guide, SDF selects the *STYLE-GUIDE-MODE setting; otherwise, the *OLD-MODE setting.

i The terminal type with which the terminal or terminal emulation was generated in the system is evaluated for this setting. If the generated terminal type differs from the actual terminal type, there is no guarantee that the setting will reflect the actually supported functionality. In the case of an emulation, the recognized terminal type will depend on the generation as well as the environment variables. See the description of the emulation program for more details.

INPUT-HISTORY =
Specifies whether the input buffer is to be turned on, turned off, or reset.

INPUT-HISTORY = *UNCHANGED
The input history setting defined in the global information is not changed.

INPUT-HISTORY = *STD
The existing setting for the option is retained when the processed user or syntax file is activated. In the case of a system syntax file, *STD has the same effect as *ON.

INPUT-HISTORY = *ON
The input buffer is turned on, and SDF saves all syntactically correct inputs in it. SET-LOGON-PARAMETERS, RESTORE-SDF-INPUT, SHOW-INPUT-HISTORY, and ISP commands are not saved.
The saved inputs can be output by the user by means of the SHOW-INPUT-HISTORY statement. The RESTORE-SDF-INPUT statement can be used to retrieve a particular input and then repeat it with or without modifications.

i Values which are specified for “secret” operands and which correspond to neither the default value nor a value defined with SECRET=*NO are saved in the input buffer as a “^”. Values specified for operands not defined as secret, by contrast, are saved as plain text. In some cases, such information (e.g. procedure parameters) may also be worth protecting from a user´s viewpoint. To prevent such inputs from being re-displayed on the screen by SHOW-INPUT-HISTORY or RESTORE-SDF-INPUT, the user can turn off the input buffer (i.e. the history feature) before making entries for which security is required and then turn it on again. Alternatively, if the inputs have already been saved, the input buffer can be purged with *RESET, in which case all saved inputs will be deleted.

INPUT-HISTORY = *OFF
The input buffer is turned off. Subsequent inputs are not stored; however, but inputs saved earlier remain accessible.

NUMBER-OF-INPUTS = *UNCHANGED / *STD / <integer 1..100>
part of the INPUT-HISTORY operand; defines how many inputs can be saved in the input buffer. The maximum possible number is 100.

GENERAL-INFO-VERSION = *UNCHANGED / <integer 1..255>
This operand is reserved for the administrator of privileges and is therefore not described here.

MODIFY-LANGUAGE-TEXT =
The global information includes the texts used by SDF when conducting the dialog. These texts may be defined for various languages. Which language is given priority by SDF is determined at system generation, along with the language for message output.

MODIFY-LANGUAGE-TEXT = *UNCHANGED
The texts remain unchanged.

MODIFY-LANGUAGE-TEXT = list-poss(2000): <name 1..1>(...)
The texts for the specified language key <name> (E = English, D = German) are changed. If no texts exist for a specified language key, a new set of texts is created for it by copying from the first available language. These texts are then subsequently changed. For certain texts there is an unabbreviated form (LONG-TEXT) and a short form (ABBREVIATION). SDF uses the long form for the maximum level of guided dialog, the short form for the medium and minimum levels.
From SDF/SDF-A Version 2.0 onwards, a help text can also be created for the predefined SDF data types. This help text is displayed if the user enters '??' during guided dialog. In guided dialog, using the EDIT statement to position on GLOBAL-INFORMATION before calling the SET-GLOBALS statement calls up the operand forms for SET-GLOBALS with the default texts. An example (see pages 331 to 341) illustrates the default texts and the functionality of the SET-GLOBALS statement for modifying texts.

REMOVE-LANGUAGE-TEXT =
Specifies whether the texts for the specified languages are deleted.

REMOVE-LANGUAGE-TEXT = *NO
No texts are deleted.

REMOVE-LANGUAGE-TEXT = list-poss(2000): <name 1..1>
The texts for the specified language key are deleted.

Default language-dependent texts

As each new syntax file is created, it is assigned predefined global information which can be modified with the SET-GLOBALS statement. Default texts are defined for the language-dependent texts specified with the MODIFY-LANGUAGE-TEXTS operand. The language-dependent texts can be classified as follows:

- Data types: used in the help texts and error messages for operand values. The default values are as follows:

Data type	**MODIFY-LANGUAGE-TEXT=E(...**	
	NAMES(LONG-TEXT=...	**,ABBREVIATION=...)**
COMMAND-REST	'command-rest'	'cmdrest'
INTEGER	'integer'	'integer'
X-STRING	'x-string'	'x-string'
C-STRING	'c-string'	'c-string'
NAME	'name'	'name'
ALPHANUM-NAME	'alphanum-name'	'aln-name'
STRUCTURED-NAME	'structured-name'	'struc-name'
LABEL	'label'	'label'
VSN	'vsn'	'vsn'
FILENAME	'filename'	'filename'
PARTIAL-FILENAME	'partial-name'	'p-filename'
TIME	'time'	'time'
DATE	'date'	'date'
COMPOSED-NAME	'composed-name'	'comp-name'
TEXT	'text'	'text'
CAT-ID	'cat-id'	'cat-id'
RODUCT-VERSION	'product-version'	'prod-ver'
POSIX-PATHNAME	'posix-pathame'	'posix-pathname'
POSIX-FILENAME	'posix-filename'	'posix-filename'
X-TEXT	'x-text'	'x-text'
FIXED	'fixed'	'fixed'

- Data type attributes: used in the help texts and error messages for operand values. The default values are as follows:

Data type attribute	**MODIFY-LANGUAGE-TEXT=E(...**	
	NAMES(LONG-TEXT=...	**,ABBREVIATION=...)**
WILDCARDS	'wildcards'	'wildcards'
LOWER-CASE	'lower-case'	'lower-case'
USER-ID	'user-id'	'user-id'
GENERATION	'generation'	'gen'
VERSION	'version'	'version'
UNDERSCORE	'underscore'	'under'
SEPARATORS	'separators'	'separ'
ODD-POSSIBLE	'odd-possible'	'odd-poss'
COMPLETION	'completion'	'completion'
LONGEST-LOGICAL-LEN	'longest-logical-len'	'long-log'
TEMPORARY-FILE	'temporary-file'	'temp-file'
QUOTES-MANDATORY	'quotes'	'quotes'
USER-INTERFACE	'manual-release'	'man-rel'
CORRECTION-STATE	'correction-state'	'corr-state'
PATH-COMPLETION	'path-completion'	'path-compl'
WILDCARD-CONSTRUCT	'wildcard-constr'	'w-constr'

- Identifiers in the headers of command and statement menus and in the operand forms. The default values are as follows:

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
DOMAIN	'DOMAIN‘
COMMAND	'COMMAND'
PROGRAM	'PROGRAM'
STATEMENT	'STATEMENT'
KEYS	'KEYS'
STRUCTURE	'STRUCTURE'
SITUATION	'SITUATION'
COMMENT-LINE	'COMMENT'
AMBIGUOUS-OPERATIONS	'OPERATIONS CORRESPONDING TO'

- Identifiers for available objects in the body of the menu. The default values are as follows:

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
DOMAIN-TITLE	'AVAILABLE APPLICATION DOMAINS'
COMMAND-TITLE	'AVAILABLE COMMANDS'
STATEMENT-TITLE	'AVAILABLE STATEMENTS'

- Prompt for immediate execution: this prompt appears only in the guided dialog (MAXIMUM guidance) and is displayed only for commands and statements having no operands.

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
DIRECT-EXECUTION	'EXECUTED IMMEDIATLY'

- Operand attributes: output in help texts and error messages in the operand forms. The default values are as follows:

Operand attribute	**MODIFY-LANGUAGE-TEXT=E(...**
DEFAULT	'default'
DEFAULT-BY-JV	'default-by-JV'
DEFAULT-BY-VAR	'default-by-variable'
ALTERNATE-DEFAULT	'alternate-default'
MANDATORY	'mandatory'
OR	'or'
WITH	'with'
WITHOUT	'without'
LIST-POSSIBLE	'list-possible'

- Structure identifiers: text output for structure identifier indicates whether there is a subform. The default identifier is as follows:

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
STRUCTURE-INCLUDED	'INPUT TRANSFERRED'

- Identifiers for menu control and errors: these appear in the bottom section of menus. The default values are as follows:

  User action/Messages/Errors

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
NEXT	'NEXT'
MESSAGE	'MESSAGE'
ERROR	'ERROR'

Explanation of syntax for NEXT field

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**	
	NAMES(LONG-TEXT=...	**,ABBREVIATION=...)**
NUMBER	'Number'	'Number'
NEXT-CMD	'Next-command'	'Next-cmd'
NEXT-STMT	'Next-statement'	'Next-stmt'
NEXT-DATA	'Next-data'	'Next-data'
NEXT-INPUT	'Next-input'	'Next-input'
NEXT-DOMAIN	'Next-domain'	'Next-dom'
DOWN-OPERAND	'DOWN'	'DOWN'
DOMAIN-MENU	'DOMAIN-MENU'	'DOM-MENU'
UP	'UP'	'UP'
DOWN	'DOWN'	'DOWN'
TEST	'TEST'	'TEST'
EXECUTE	'EXECUTE'	'EXECUTE'
CONTINUE	'CONTINUE'	'CONTINUE'
CANCEL	'CANCEL'	'CANCEL'
RESTORE	'RESTORE'	'RESTORE'
EXIT	'EXIT'	'EXIT'
REFRESH	'REFRESH'	'REFRESH'
REST-SDF-IN	'REST-SDF-IN'	'REST-SDF-IN'
EXIT-ALL	'EXIT-ALL'	'EXIT-ALL'

- Default texts for unguided dialog are as follows:

Identifier	**MODIFY-LANGUAGE-TEXT=E(...**
SECRET-OPERAND	' ENTER SECRET OPERAND'

The input line for secret operand is masked out.

A number of prompts are used if the dialog mode is GUIDANCE=*NO. The default values for these prompts are as follows:

Identifier	MODIFY-LANGUAGE-TEXT=E(...
ENTER-COMMAND	'CMD'
ENTER-STATEMENT	'STMT'
ENTER-CONTINUATION	'ENTER CONTINUATION'
ENTER-OPERANDS	'ENTER OPERANDS'
CORRECT-OPERATION	'CORRECT OPERATIONNAME'
CORRECT-COMMAND	'CORRECT CMD'
CORRECT-STATEMENT	'CORRECT STMT'
OPERANDS	'OPERANDS'

- Procedure error messages in guided dialog. These error messages appear in the menu header and indicate the reason for the dialog:

Identifier	MODIFY-LANGUAGE-TEXT=E(...
PROCEDURE-ERROR	' ERROR IN PROCEDURE-CMD/STMT'
PROC-HELP	' DIALOG IN PROCEDURE'

- Program error messages in guided dialog. These error messages appear in the menu header and indicate the reason for the correction dialog:

Identifier	MODIFY-LANGUAGE-TEXT=E(...
INTERNAL-ERROR	'ERROR IN PROC/S-PROC'
INTERNAL-HELP	'DIALOG IN PROG/S-PROC'

- Prompts concerning procedure interruption. The default values are as follows:

Identifier	MODIFY-LANGUAGE-TEXT=E(...
PROC-INTERRUPT	' NEXT-DATA'
PROC-INT-YES	' YES'
PROC-INT-NO	' NO'

Example

The example below shows how a user syntax file is created and a number of language-dependent texts forming part of global information are modified. In the screens shown on the following pages, the original texts to be modified and the new wording are shown in bold print.

```
/start-sdf-a
%  BLS0517 MODULE 'SDAMAIN' LOADED
%  SDA0001 'SDF-A' VERSION '04.1E10' STARTED
//open-syntax-file file=syssdf.user.globals,type=*user,mode=*create  ——  (1)
//set-globals mod-lang-text=e(generation=names('gen','g'),-          ⎫
                              domain-title='list of domains',-       ⎪
                              command-title='list of commands',-     ⎬  ——  (2)
                              without='no',-                         ⎪
                              next='select',-                        ⎭
                              number=names('#','#'))
//end
/modify-sdf-options guid=*max,funct-keys=*style-guide-mode  ————————  (3)
```

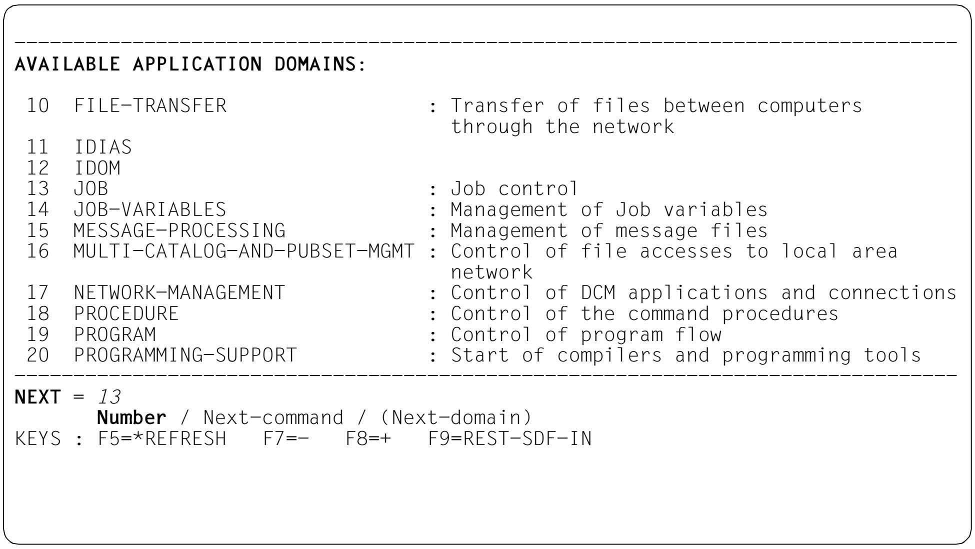
```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                      : Output of informations about the user
                                       identification and introduction of data
                                       into the accounting record
  2  ALL-COMMANDS                    : Output of all command names in alphabetic
                                       order
  3  CONSOLE-MANAGEMENT              : Control of operator console/terminal
  4  DATABASE                        : Management and administration of databases
  5  DCAM                            : Control of transaction-driven system (DCAM)
  6  DCE                             : Management of DCE (Distributed Computing
                                       Environment)
  7  DEVICE                          : Information about devices and volumes
  8  FILE                            : Management of files
  9  FILE-GENERATION-GROUP           : Management of file generation groups
--------------------------------------------------------------------------------
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```

———————————————————————————————— (4)

1. When SDF-A is called, a user syntax file with the name SYSSDF.USER.GLOBALS is created and opened.
2. A number of English texts that appear in guided dialog are then modified.

3. SDF-A is terminated, which means that the user syntax file is saved. The user then switches to guided dialog. The SYSSDF.USER.GLOBALS syntax file is not yet activated, so the default SDF texts appear in the menus.

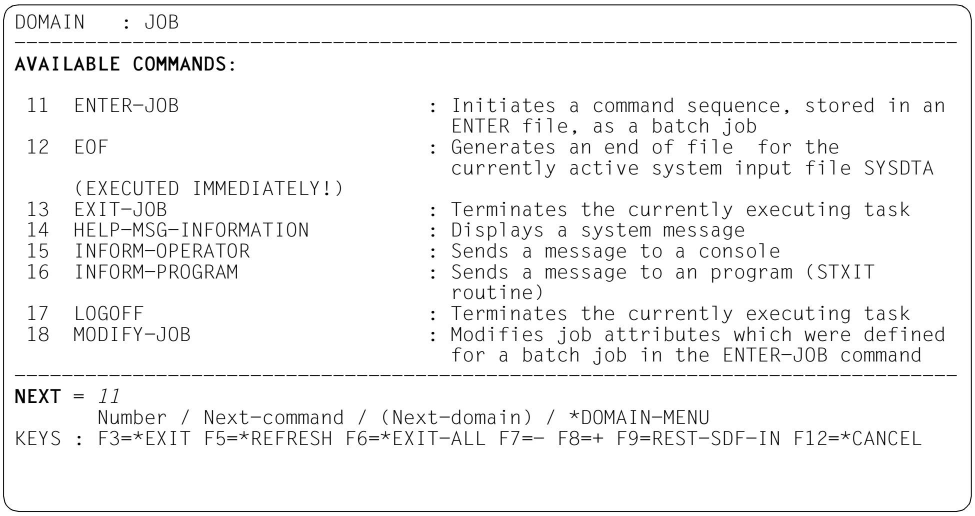
4. The domains menu appears. In this screen, the texts to be modified are shown in bold print. By selecting option 9, the user switches to the JOB domain.



```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 10  FILE-TRANSFER                  : Transfer of files between computers
                                      through the network
 11  IDIAS
 12  IDOM
 13  JOB                            : Job control
 14  JOB-VARIABLES                  : Management of Job variables
 15  MESSAGE-PROCESSING             : Management of message files
 16  MULTI-CATALOG-AND-PUBSET-MGMT  : Control of file accesses to local area
                                      network
 17  NETWORK-MANAGEMENT             : Control of DCM applications and connections
 18  PROCEDURE                      : Control of the command procedures
 19  PROGRAM                        : Control of program flow
 20  PROGRAMMING-SUPPORT            : Start of compilers and programming tools
--------------------------------------------------------------------------------
NEXT = 13
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F8=+   F9=REST-SDF-IN
```

(5)

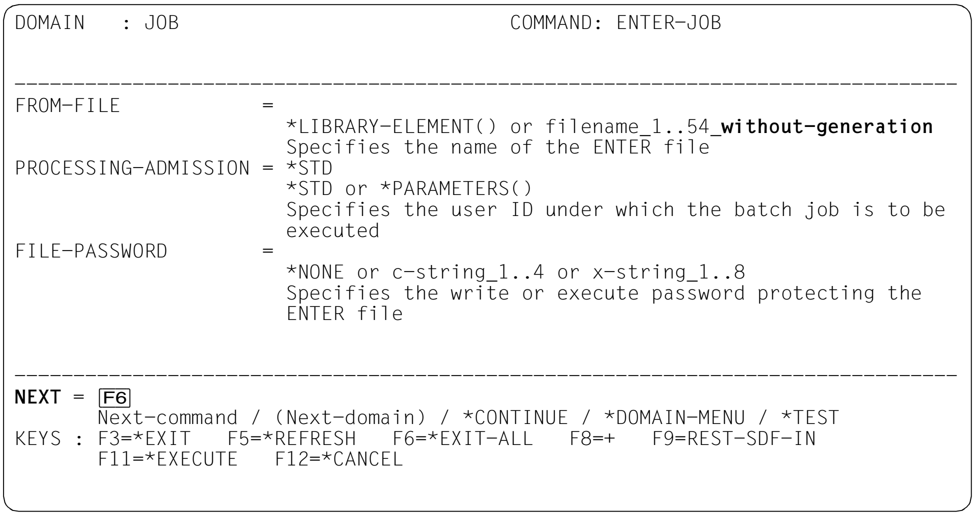
5. Entering the number 13 and pressing DUE switches the user to the JOB domain.



```
DOMAIN   : JOB
--------------------------------------------------------------------------------
AVAILABLE COMMANDS:

 11  ENTER-JOB                        : Initiates a command sequence, stored in an
                                        ENTER file, as a batch job
 12  EOF                              : Generates an end of file  for the
                                        currently active system input file SYSDTA
     (EXECUTED IMMEDIATELY!)
 13  EXIT-JOB                         : Terminates the currently executing task
 14  HELP-MSG-INFORMATION             : Displays a system message
 15  INFORM-OPERATOR                  : Sends a message to a console
 16  INFORM-PROGRAM                   : Sends a message to an program (STXIT
                                        routine)
 17  LOGOFF                           : Terminates the currently executing task
 18  MODIFY-JOB                       : Modifies job attributes which were defined
                                        for a batch job in the ENTER-JOB command
--------------------------------------------------------------------------------
NEXT = 11
       Number / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT F5=*REFRESH F6=*EXIT-ALL F7=- F8=+ F9=REST-SDF-IN F12=*CANCEL
```

(6)

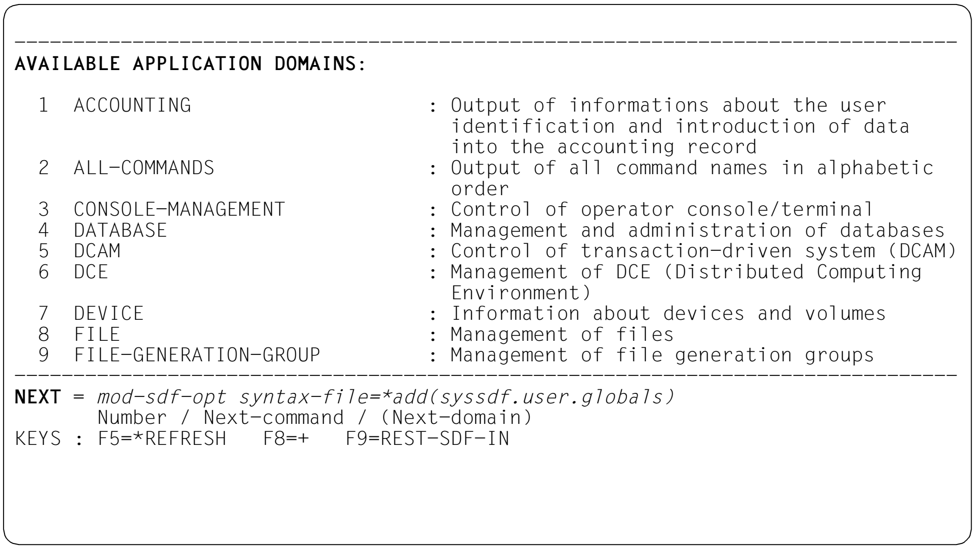
6. The command menu for the JOB domain appears. The DUE key (preset value '+' in the NEXT line) or F8 was already pressed once to page down (to see the ENTER-JOB command). Text that is highlighted in the command menu needs to be changed. Entering the number 11 switches to the operand form of the ENTER-JOB command.



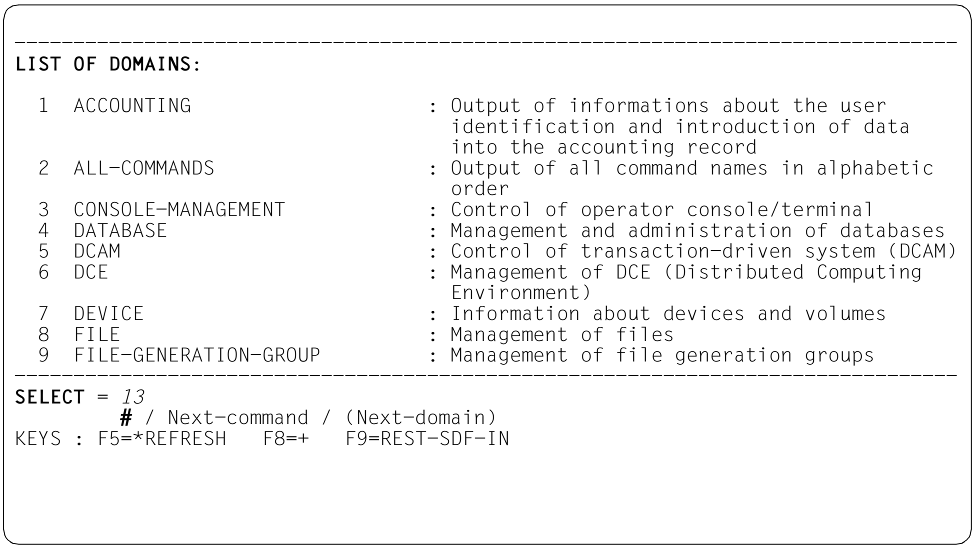
```
DOMAIN   : JOB                          COMMAND: ENTER-JOB


--------------------------------------------------------------------------------
FROM-FILE             =
                        *LIBRARY-ELEMENT() or filename_1..54_without-generation
                        Specifies the name of the ENTER file
PROCESSING-ADMISSION = *STD
                        *STD or *PARAMETERS()
                        Specifies the user ID under which the batch job is to be
                        executed
FILE-PASSWORD         =
                        *NONE or c-string_1..4 or x-string_1..8
                        Specifies the write or execute password protecting the
                        ENTER file


--------------------------------------------------------------------------------
NEXT = F6
       Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

(7)

7. The operand form of the ENTER-JOB command appears. Under the FROM-FILE operand, the **`without-generation`** data type attribute is highlighted - this text, too, will be modified. To perform a comparison, the user syntax file SYSSDF.USER.GLOBALS should now be activated by pressing F6 (alternative: enter *EXIT-ALL in the NEXT line and press DUE) to return to the global domains menu.


```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                      : Output of informations about the user
                                       identification and introduction of data
                                       into the accounting record
  2  ALL-COMMANDS                    : Output of all command names in alphabetic
                                       order
  3  CONSOLE-MANAGEMENT              : Control of operator console/terminal
  4  DATABASE                        : Management and administration of databases
  5  DCAM                            : Control of transaction-driven system (DCAM)
  6  DCE                             : Management of DCE (Distributed Computing
                                       Environment)
  7  DEVICE                          : Information about devices and volumes
  8  FILE                            : Management of files
  9  FILE-GENERATION-GROUP           : Management of file generation groups
--------------------------------------------------------------------------------
NEXT = mod-sdf-opt syntax-file=*add(syssdf.user.globals)
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```
(8)

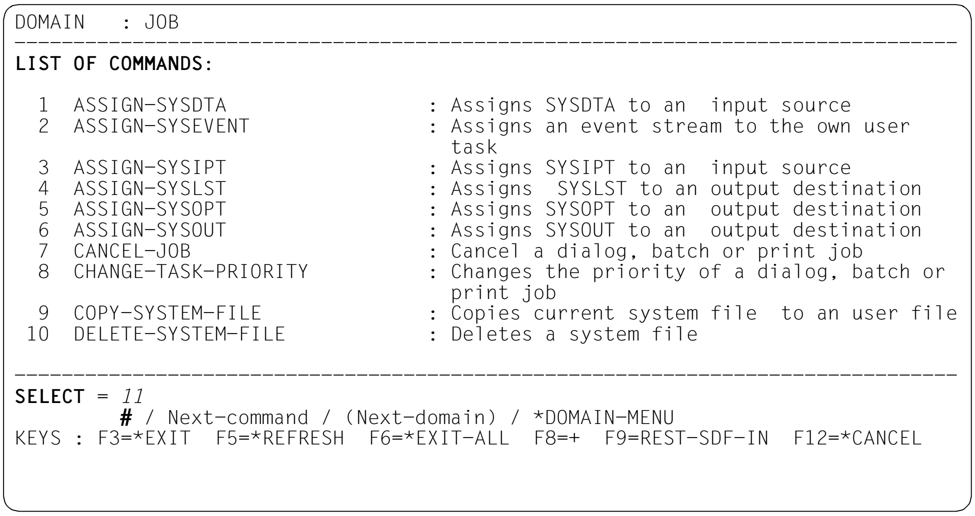
8. The global domains menu appears. The highlighted texts have not yet been modified. The user now calls the MODIFY-SDF-OPTIONS to activate the user syntax file SYSSDF.USER.GLOBALS, in response to which the texts are immediately modified.



```
--------------------------------------------------------------------------------
LIST OF DOMAINS:

  1  ACCOUNTING                     : Output of informations about the user
                                      identification and introduction of data
                                      into the accounting record
  2  ALL-COMMANDS                   : Output of all command names in alphabetic
                                      order
  3  CONSOLE-MANAGEMENT             : Control of operator console/terminal
  4  DATABASE                       : Management and administration of databases
  5  DCAM                           : Control of transaction-driven system (DCAM)
  6  DCE                            : Management of DCE (Distributed Computing
                                      Environment)
  7  DEVICE                         : Information about devices and volumes
  8  FILE                           : Management of files
  9  FILE-GENERATION-GROUP          : Management of file generation groups
--------------------------------------------------------------------------------
SELECT = 13
         # / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN
```

(9)

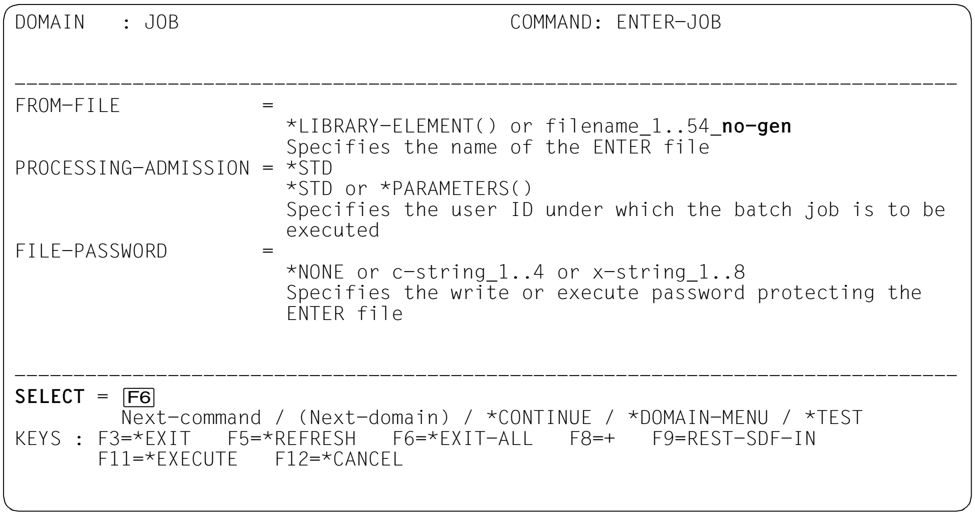
9. The global domains menu remains on the screen, but the new texts (highlighted in bold in the screens shown here) have been incorporated. The user selects option 13 to switch to the JOB domain.



```
DOMAIN   : JOB
--------------------------------------------------------------------------------
LIST OF COMMANDS:

  1  ASSIGN-SYSDTA                  : Assigns SYSDTA to an  input source
  2  ASSIGN-SYSEVENT                : Assigns an event stream to the own user
                                      task
  3  ASSIGN-SYSIPT                  : Assigns SYSIPT to an  input source
  4  ASSIGN-SYSLST                  : Assigns  SYSLST to an output destination
  5  ASSIGN-SYSOPT                  : Assigns SYSOPT to an  output destination
  6  ASSIGN-SYSOUT                  : Assigns SYSOUT to an  output destination
  7  CANCEL-JOB                     : Cancel a dialog, batch or print job
  8  CHANGE-TASK-PRIORITY           : Changes the priority of a dialog, batch or
                                      print job
  9  COPY-SYSTEM-FILE               : Copies current system file  to an user file
 10  DELETE-SYSTEM-FILE             : Deletes a system file

--------------------------------------------------------------------------------
SELECT = 11
         # / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT  F5=*REFRESH  F6=*EXIT-ALL  F8=+  F9=REST-SDF-IN  F12=*CANCEL
```

(10)

10. The menu for the JOB domain appears. Here, too, bold print is used to highlight the modified texts. The user selects option 11 to switch to the operand form for the ENTER-JOB command.



```
DOMAIN   : JOB                                  COMMAND: ENTER-JOB


--------------------------------------------------------------------------------
FROM-FILE              =
                         *LIBRARY-ELEMENT() or filename_1..54_no-gen
                         Specifies the name of the ENTER file
PROCESSING-ADMISSION = *STD
                         *STD or *PARAMETERS()
                         Specifies the user ID under which the batch job is to be
                         executed
FILE-PASSWORD          =
                         *NONE or c-string_1..4 or x-string_1..8
                         Specifies the write or execute password protecting the
                         ENTER file


--------------------------------------------------------------------------------
SELECT = F6
         Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL
```

11. The operand form for the ENTER-JOB command appears. The text for the data type attribute under the FROM-FILE operand has been changed from **without-generation** to **no-gen**. By pressing F6 (alternative: enter *EXIT-ALL in the NEXT line and press DUE), the user returns to the global domains menu.

SHOW
Display objects of syntax file

The SHOW statement is used to output the contents of a syntax file to SYSOUT or SYSLST. The output can be interrupted and restarted or aborted with the K2 key.

The output from this statement can be used as input in order to reconstruct a syntax file or a syntax file object (see OBJECT=*ALL, IMPLEMENTATION=*YES, LINES-PER-PAGE=*UNLIMITED(OUTPUT=*FOR-INPUT)).

(part 1 of 4)

```
SHOW

OBJECT = *ALL / *GLOBAL-INFORMATION / *DOMAIN(...) / *COMMAND(...) / *PROGRAM(...) /
              *STATEMENT(...) / *PRIVILEGE(...) / *OPERAND(...) / *VALUE(...) /
              *CORRECTION-INFORMATION(...)

   *DOMAIN(...)
     |   NAME = *ALL(...) / *NONE / <structured-name 1..30 with-wild> /
     |             list-poss(2000): <structured-name 1..30>
     |
     |     *ALL(...)
     |       |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
     |       |               list-poss(2000): <structured-name 1..30>

   *COMMAND(...)
     |   NAME = *ALL(...) / *REMOVED / *CURRENT / <structured-name 1..30 with-wild> /
     |               list-poss(2000): <structured-name 1..30>
     |
     |     *ALL(...)
     |       |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
     |       |               list-poss(2000): <structured-name 1..30>

   *PROGRAM(...)
     |   NAME = *ALL(...) / *REMOVED / *CURRENT / <structured-name 1..30 with-wild> /
     |               list-poss(2000): <structured-name 1..30>
     |
     |     *ALL(...)
     |       |   EXCEPT = *NONE / <structured-name 1..30 with-wild> /
     |       |               list-poss(2000): <structured-name 1..30>
```

continued ➠

(part 2 of 4)

```
*STATEMENT(...)
   NAME = *ALL(...) / *REMOVED / *CURRENT / <structured-name 1..30 with-wild> /
                list-poss(2000): <structured-name 1..30>
     *ALL(...)
        EXCEPT = *NONE / <structured-name 1..30 with-wild> /
                     list-poss(2000): <structured-name 1..30>
  ,PROGRAM = <structured-name 1..30>
*PRIVILEGE(...)
   NAME = *ALL / <structured-name 1..30>
*OPERAND(...)
   OPERAND-L1 = *CURRENT / <structured-name 1..20>
  ,VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
              *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
              *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
              *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
              *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
  ,OPERAND-L2 = *NO / <structured-name 1..20>
  ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
              *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
              *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
              *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
              *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
  ,OPERAND-L3 = *NO / <structured-name 1..20>
  ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
              *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
              *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
              *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
              *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
  ,OPERAND-L4 = *NO / <structured-name 1..20>
  ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
              *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
              *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
              *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
              *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
```

continued ➠

(part 3 of 4)

```
    ,OPERAND-L5 = *NO / <structured-name 1..20>
    ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
       *COMMAND(...)
          NAME = <structured-name 1..30>
       *STATEMENT(...)
          NAME = <structured-name 1..30>
          ,PROGRAM = <structured-name 1..30>
*VALUE(...)
    OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
    ,VALUE-L1 = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
    ,OPERAND-L2 = *NO / <structured-name 1..20>
    ,VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
    ,OPERAND-L3 = *NO / <structured-name 1..20>
    ,VALUE-L3 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
    ,OPERAND-L4 = *NO / <structured-name 1..20>
    ,VALUE-L4 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
                *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
                *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
                *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
                *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
    ,OPERAND-L5 = *NO / <structured-name 1..20>
```

continued ➠

(part 4 of 4)

```
        | ,VALUE-L5 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME /
        |             *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME /
        |             *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN /
        |             *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION /
        |             *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
        |
        | ,ORIGIN = *CURRENT / *COMMAND(...) / *STATEMENT(...)
        |
        |    *COMMAND(...)
        |       | NAME = <structured-name 1..30>
        |
        |    *STATEMENT(...)
        |       | NAME = <structured-name 1..30>
        |       | ,PROGRAM = <structured-name 1..30>
     *CORRECTION-INFORMATION(...)
        | PM-NUMBER = *ALL / list-poss(20): <alphanum-name 8..8>
,ATTACHED-INFORMATION = *YES / *NO / *IMMEDIATE
,SIZE = *MINIMUM / *MAXIMUM / *MEDIUM
,IMPLEMENTATION-INFO = *NO(...) / *YES
     *NO(...)
        | FORM = *UNGUIDED / *GUIDED
        | ,LANGUAGE = E / <name 1..1>
,LINE-LENGTH = *STD / <integer 72..132>
,LINES-PER-PAGE = *STD / *UNLIMITED(...) / <integer 1..200>
     *UNLIMITED(...)
        | OUTPUT-FORM = *STD / *FOR-INPUT
,OUTPUT = *SYSOUT / *SYSLST(...)
     *SYSLST(...)
        | SYSLST-NUMBER = *STD / <integer 1..99>
,PRIVILEGE = *ANY / list-poss(64): <structured-name 1..30>
```

OBJECT =
Type of the object whose definition is to be output.

OBJECT = *ALL
Specifies that the entire contents of a syntax file are to be output.

OBJECT = *GLOBAL-INFORMATION
Specifies that the global information of a syntax file is to be output.

OBJECT = *DOMAIN(...)
Specifies that the definitions of domains are output.

NAME = *ALL(...)
The definitions of all domains are output.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the named domains are not output.

NAME =*NONE
The definitions of all the commands which are not assigned to a domain are output.

NAME = <structured name 1..30 with-wild / list-poss(2000): <structured-name 1..30>
The definitions of the named domains or of the domains whose names match the wildcard selector are output.

OBJECT = *COMMAND(...)
Specifies that the definitions of commands are to be output.

NAME = *ALL(...)
The definitions of all commands are to be output.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the named commands are not output.

NAME = *REMOVED
Outputs all commands which have been removed (and which can be restored because their description exists on a higher hierarchical level).

NAME = *CURRENT
Outputs the current commands (if any).

NAME = <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the named commands or the definitions of the commands which match the wilcard selector are output.

OBJECT = *PROGRAM(...)
Specifies that definitions of programs are to be output.

NAME = *ALL(...)
The definitions of all programs are output.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the programs named here are not output.

NAME = *REMOVED
Outputs all programs which have been removed (and which can be restored because their description exists on a higher hierarchical level).

NAME =<structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the named programs or of the programs whose name matches the wildcard selector are to be output.

OBJECT = *STATEMENT(...)
Specifies that the definitions of statements are to be output.

NAME = *ALL(...)
The definitions of all statements are output.

EXCEPT = *NONE / <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the statements named here are not output.

NAME = *REMOVED
Outputs all statements which have been removed (and which can be restored because their description exists on a higher hierarchical level).

NAME = *CURRENT
Outputs the current statements (if any).

NAME = <structured-name 1..30 with-wild> / list-poss(2000): <structured-name 1..30>
The definitions of the named statements or the definitions of the statements which match the wildcard selector are output.

PROGRAM = <structured-name 1..30>
Name of the program to which the statements pertain.

OBJECT = *PRIVILEGE(...)
Specifies that the definitions of privileges are to be output.

NAME = *ALL / <structured-name 1..30>
The definitions of all privileges or of the privileges named are to be output.

OBJECT = *OPERAND(...)
Specifies that the definition of an operand is to be output. If this operand is included in a structure, it is specified by the path leading to it, i.e. by specifying the operands and operand values that introduce the structure in this path. If the name of one of the operands in the path is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified (and may even be omitted). An operand that is not absolutely essential to identify the operand definition to be output, as well as the operand value pertaining to it, can be omitted.
An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first VALUE-Li = *NO, SDF-A takes the operand defined by OPERAND-Li as the one whose definition is to be output. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for VALUE-Li, the value specified for OPERAND-Li+1 must also be other than *NO.

OPERAND-L1 = *CURRENT / <structured-name 1..20>
Specifies the operand whose definition is to be output (VALUE-L1 = *NO) or an operand in the path leading to it (VALUE-L1 ≠ *NO). *CURRENT means that OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L1 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO means that the definition of the OPERAND-L1 is to be output. Otherwise, an operand value that introduces a structure is to be specified. This structure must directly or indirectly contain the operand whose definition is to be output. If the operand value introducing the structure is of the data type KEYWORD-NUMBER, then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Here it must be remembered that this particular value is to be specified in every case without the prefixed asterisk. If the operand value introducing the structure is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that OPERAND-L2 is irrelevant for the specification of the operand whose definition is to be output. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L1 is to be specified. This operand is either the one whose definition is to be output (VALUE-L2 = *NO) or one that is in the path leading to it (VALUE-L2 ≠ *NO).

VALUE-L2 = analogous to VALUE-L1
*NO means that VALUE-L2 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a structure is to be specified. This structure must directly or indirectly contain the operand whose definition is to be output. For further information, see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand whose definition is to be output. Otherwise, the name of an operand that is unique within the structure determined by VALUE-L2 is to be specified. This operand is either the one whose definition is to be output (VALUE-L3 = *NO) or one that is in the path leading to it (VALUE-L3 ≠ *NO).

VALUE-L3 = analogous to VALUE-L1
*NO means that VALUE-L3 is irrelevant for the specification of the operand. Otherwise, an operand value introducing a structure is to be specified. The structure must directly or indirectly contain the operand whose definition is to be output. For further information, see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L2.

VALUE-L4 = analogous to VALUE-L1
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L2.

ORIGIN =
Specifies the command or statement to which the operand definition to be output pertains.

ORIGIN = *CURRENT
The operand definition to be output pertains to a command or statement that currently either is itself the current object or else contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)
The operand definition pertains to a command.

> **NAME = <structured-name 1..30>**
> Name of the command.

ORIGIN = *STATEMENT(...)
The operand definition belongs to a statement.

> **NAME = <structured-name 1..30>**
> Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *VALUE(...)
The definition of the operand value is to be output. This operand value is specified by the path leading to it, i.e. by specifying the operands and operand values introducing the structure in this path. If the operand value pertains to an operand that is not attached to any structure, the path contains only this operand. If the operand value does pertain to an operand attached to a structure, the path also includes the higher-ranking operands as well as the associated operand values introducing the structure. If the name of one of the operands is unique, not only within its structure, but also with respect to the higher-ranking structure (or globally within the command or statement), the path need not be completely specified. An operand that is not absolutely essential to identify the operand value definition to be output, as well as the operand value pertaining to it, can be omitted. An operand value specified for VALUE-Li (i=1,...,5) must pertain to the operand defined by OPERAND-Li. After the first OPERAND-Li + 1 = *NO, SDF-A takes the operand value defined by VALUE-Li as the one whose definition is to be output. Subsequently, SDF-A does not interpret the specifications for any other OPERAND-Lj, VALUE-Lj. If a value other than *NO is specified for OPERAND-Li, the value specified for VALUE-Li must also be other than *NO.

OPERAND-L1 = *ABOVE-CURRENT / <structured-name 1..20>
Specifies the operand to which the operand value whose definition is to be output pertains (OPERAND-L2 = *NO) or an operand in the path leading to this operand value (OPERAND-L2 ≠ *NO). *ABOVE-CURRENT means that a value pertaining to OPERAND-L1 is the current object. <structured-name> must be a globally unique operand name within the command or statement.

VALUE-L = *CURRENT / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING/ *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME /<composed-name 1..30>
Specifies the operand value whose definition is to be output (OPERAND-L2 = *NO) or an operand value that introduces a structure in the path leading to it (OPERAND-L2≠*NO). *CURRENT means that VALUE-L1 is the current object. If it is not the current object and of the data type KEYWORD(-NUMBER), then the particular value defined for it is to be specified (see ADD-VALUE TYPE=*KEYWORD,...,VALUE=<c-string>). Here it must be remembered that this particular value is to be specified in every case without the prefixed asterisk. If the operand value is not of the type KEYWORD(-NUMBER), then the data type defined for it is to be specified.

OPERAND-L2 = *NO / <structured-name 1..20>
*NO means that the definition of VALUE-L1 is to be output. Otherwise, the name of the operand to which the operand value whose definition is to be output pertains (OPERAND-L3 = *NO) or the name of an operand in the path leading to this operand value (OPERAND-L3 ≠ *NO). If an operand name is specified, this must be unique within the structure defined by VALUE-L1.

VALUE-L2 = *NO / *COMMAND-REST / *INTEGER / *X-STRING / *C-STRING / *NAME / *ALPHANUMERIC-NAME / *STRUCTURED-NAME / *FILENAME / *PARTIAL-FILENAME / *TIME / *DATE / *TEXT / *CAT-ID / *LABEL / *VSN / *COMPOSED-NAME / *X-TEXT / *FIXED / *DEVICE / *PRODUCT-VERSION / *POSIX-PATHNAME / *POSIX-FILENAME / <composed-name 1..30>
*NO means that the VALUE-L2 is irrelevant for the specification of the operand value to be output. Otherwise, an operand value is to be specified. This operand value is either the one whose definition is to be output (OPERAND-L3=*NO) or an operand value introducing a structure in the path leading to it (OPERAND-L3 ≠ *NO). For further information, see VALUE-L1.

OPERAND-L3 = *NO / <structured-name 1..20>
*NO means that OPERAND-L3 is irrelevant for the specification of the operand value definition to be output. Otherwise, the name of the operand to which the operand value whose definition is to be output pertains (OPERAND-L4 = *NO) or the name of an operand in the path leading to this operand value (OPERAND-L4 ≠ *NO) is to be specified. If an operand name is specified, this must be unique within the structure defined by VALUE-L2.

VALUE-L3 = analogous to VALUE-L2
*NO means that VALUE-L3 is irrelevant for the specification of the operand value definition value to be output. Otherwise, an operand value is to be specified. This operand value is either the one whose definition is to be output (OPERAND-L4 = *NO) or an operand value introducing a structure in the path leading to it (OPERAND-L4 ≠ *NO). For further information, see VALUE-L1.

OPERAND-L4 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L4 = analogous to VALUE-2
see VALUE-L2.

OPERAND-L5 = *NO / <structured-name 1..20>
see OPERAND-L3.

VALUE-L5 = analogous to VALUE-2
see VALUE-L2.

ORIGIN =
Specifies the command or statement to which the operand value definition to be output pertains.

ORIGIN = *CURRENT
The operand value definition to be output pertains to a command or statement that currently either is itself the current object or else contains an operand or operand value that is the current object.

ORIGIN = *COMMAND(...)
The operand value definition pertains to a command.

NAME = <structured-name 1..30>
Name of the command.

ORIGIN = *STATEMENT(...)
The operand value definition pertains to a statement.

NAME = <structured-name 1..30>
Name of the statement.

PROGRAM = <structured-name 1..30>
Name of the program to which the statement pertains.

OBJECT = *CORRECTION-INFORMATION(...)
This operand value is reserved for special purposes for Fujitsu Siemens Computers Software Development and consequently is not described in this manual.

ATTACHED-INFORMATION =
Specifies which of the definitions pertaining to the specified object is to be output.

ATTACHED-INFORMATION = *YES
The definition of the specified object is output along with the definitions of all objects associated with the specified object. (In other words: domain with associated commands, program with associated statements, command or statement with associated operands, operand with associated operand values, operand value with associated structure, global information with language-dependent texts.)

ATTACHED-INFORMATION = *NO
The definition of the specified object is output without the definitions of the objects associated with the specified object. (In other words: domain without associated commands, program without associated statements, command or statement without associated operands, operand without associated operand values, operand value without associated structure, global information without language-dependent texts.)

ATTACHED-INFORMATION = *IMMEDIATE
The definition of the specified object is output together with the definitions of the objects which are immediately associated with it. Domains and programs are output with the associated commands or statements, but without the associated operands and operand values. For all other objects, *IMMEDIATE has the same effect as *YES.

SIZE =
Specifies the extent of output. The actual effect of the SIZE operand depends on the OBJECT operand. If IMPLEMENTATION-INFO=*YES is specified, *MEDIUM has the same effect as *MAXIMUM. SIZE has no effect on the output of global information. For OBJECT=*COMMAND or *STATEMENT, and analogously for *OPERAND and *VALUE, the following information is output:

SIZE = <u>*MINIMUM</u>
If IMPLEMENTATION-INFO=*NO:
the command/statement name, all associated operand names, all associated default values, and the operand values introducing structures are output. The additional information regarding the operand values and help texts is not output.
If IMPLEMENTATION-INFO=*YES:
parts of the definition defined with the default values of the associated ADD statements are not output.

SIZE = *MAXIMUM
If IMPLEMENTATION-INFO=*NO:
the command/statement name, all associated operand names, all associated default values, and all other associated operand values are output. In addition, the additional information regarding the operand values and the help texts is output.
If IMPLEMENTATION-INFO=*YES:
all parts of the definition are output.

SIZE = *MEDIUM
If IMPLEMENTATION-INFO=*NO:
the command/statement name, all associated operand names, all associated default values, and all other associated operand values are output. The additional information regarding the operand values and help texts is not output.
If IMPLEMENTATION-INFO=*YES:
parts of the definition defined with the default values of the associated ADD statements are not output.

IMPLEMENTATION-INFO =
Specifies how the output is to be edited. This operand has no effect for global information.

IMPLEMENTATION-INFO = <u>*NO</u>(...)
The definitions of the specified objects are output in a form similar to that of a manual. Operands assigned the value PRESENCE=*INTERNAL-ONLY are not listed here.

FORM =
Specifies whether the definitions of objects that are not permitted for guided dialog are output.

FORM = <u>*UNGUIDED</u>
The definitions are output.

FORM = *GUIDED
The definitions are not output.

LANGUAGE = E / <name 1..1>
Specifies in which language the help texts are output (E = English, D = German). This operand has no effect for global information.

IMPLEMENTATION-INFO = *YES
Those SDF-A statements are listed with which the specified objects could be defined in a syntax file.

LINE-LENGTH = *STD / <integer 72..132>
Specifies the line length for the output.

LINE-LENGTH = *STD
The default value is 74 characters for output on a display terminal and 72 characters for output to a file.

LINES-PER-PAGE = *STD / *UNLIMITED(...) / <integer 1..200>
Specifies the number of lines per page. This value does not include the two lines for the header which SDF-A generates on each new page. The header is generated only if OUTPUT = *SYSLST.

LINES-PER-PAGE = *STD
The default value is 24 lines for output on a display terminal and 55 lines for output to a file.

LINES-PER-PAGE = *UNLIMITED(...)
No control by SDF-A (no header is generated).

OUTPUT-FORM =
Specifies which characters can be output at the beginning of the lines.

OUTPUT-FORM = *STD
The first character of this line is a blank.

OUTPUT-FORM = *FOR-INPUT
Two slashes (//) are output at the beginning of each line.
This specification can be used together with IMPLEMENTATION=*YES to generate SDF-A statements with which a syntax file or a syntax file object can be reconstructed.
Example: the following SDF-A statement creates a file containing the SDF-A statements for reconstructing a syntax file:

```
SHOW OBJECT=*ALL,IMPLEMENTATION-INFO=*YES,
LINES-PER-PAGE=*UNLIMITED(OUTPUT-FORM=*FOR-INPUT),
OUTPUT= *SYSLST(07),LINE-LENGTH=72
```

OUTPUT =
Specifies the output medium for the requested information.

OUTPUT = *SYSOUT
The output is sent to the logical system file SYSOUT; in interactive mode, this is generally the display terminal.

OUTPUT = *SYSLST(...)
The output is sent to the logical system file SYSLST.

SYSLST-NUMBER = *STD / <integer 1..99>
Specifies the number of the logical system file SYSLST. If *STD is specified, the logical system file SYSLST does not receive a number.

PRIVILEGE = *ANY / list-poss(64): <structured-name 1..30>
Only those objects are output to which at least one of the listed privileges is assigned. If *ANY is specified, the objects are output irrespective of their privileges.

SHOW-CORRECTION-INFORMATION
Show correction information of syntax file

The SHOW-CORRECTION-INFORMATION statement provides information about the corrections contained in the syntax file.

The statement is intended exclusively for diagnostic purposes.

```
SHOW-CORRECTION-INFORMATION

CORRECTION-ID = *SOURCE (...) / *OBJECT(...)
   *SOURCE(...)
     |   PM-NUMBER = *ALL / list-poss(100): <alphanum-name 8..8>
   *OBJECT(...)
     |   PM-NUMBER = *ALL / <alphanum-name 8..8>
     |  ,JULIAN-DATE = *ANY / <integer 1..999>
,PRODUCT-NAME = *ANY / <structured-name 1..15>(...)
   <structured-name 1..15>(...)
     |   VERSION = *ANY / <product-version>
```

There is no description of the operands here because the statement is only intended for diagnostic purposes.

SHOW-STATUS
Display status of opened syntax file

The SHOW-STATUS statement causes SDF-A to output the name of the currently open syntax file and information on one or more associated reference syntax files.

SHOW-STATUS

This statement has no operands.

STEP
Define restart point

The STEP statement is used to define a restart point for error handling in a sequence of SDF-A statements. The STEP statement is valid only in procedures and batch runs.

STEP

This statement has no operands.

If SDF-A encounters a syntax error or a serious logical error, it instigates the following steps:

– an error message is output

– the current SDF-A statement is terminated

– the subsequent statements are skipped until STEP or END is reached.

If the next command reached by SDF-A is an END command, an abnormal program termination (TERM UNIT=STEP,MODE=ABNORMAL) is generated, and the spin-off mechanism is activated. If the next command reached by SDF-A is a STEP command, it continues with the statement following STEP. If an error has no effect on the normal execution of the current job, the user must intercept abnormal program termination by inserting a STEP command.

SDF corrects minor logical errors automatically, accompanying the correction by an appropriate warning message. The spin-off mechanism is not activated.

6 SDF program interface

SDF also provides facilities for reading statements via the SDF user interface, and for analyzing and correcting these statements, for user-defined programs. These programs can thus be operated via the same user interface as the operating system itself.

When a user program is to be used with SDF, the following requirements must be satisfied:

– a syntax description in a syntax file must be created with SDF-A, using statements such as ADD-PROGRAM and ADD-STMT;

– within the program, SDF must be called in order to request the statements, using the SDF macros described in this section or the interface's function calls between SDF and high-level languages.

6.1 Macro calls for implementing statements

The SDF macros described in this section cause SDF to request statements.

The macros are stored in the macro library SYSLIB.SDF.045, which is supplied with SDF. The macros are described in this manual because they can be used only if a suitable syntax can be defined with SDF-A.

Processing of the macros depends on the SDF version running on the system.

Changes for programs under SDF V4.1A and higher

The layout of the standardized transfer area was changed for SDF V4.1. For this reason the CMDSTRUC macro for generating the transfer area was replaced by the CMDTA macro. In addition, the macros RDSTMT, TRSTMT and CORSTMT for processing statements were replaced by the new macros CMDRST, CMDTST and CMDCST.

The format of the standardized transfer area used up to SDF V4.0 and the macros CMDSTRUC, RDSTMT, TRSTMT and CORSTMT should no longer be used in the new programs; however, for the convenience of the user descriptions can still be found in the appendix (see section “Changes to the SDF program interface” on page 593ff).

i The SDF interface to higher programming languages only supports the standardized transfer area in the old format. The new format can only be used via the Assembler interface!

6.2 Macro calls in simultaneously open syntax file hierarchies

With the aid of the OPNCALL macro, user programs open a syntax file hierarchy parallel to the syntax file hierarchy which was opened for the user task during LOGON processing. Parallel to the work context created by SDF for the user task, a new work context is thus created.

The new context is opened by the program and is used to analyze subsequent inputs. The context is closed again with the CLSCALL macro or when the program is terminated.

The contexts opened by the OPNCALL macro are therefore called “program contexts”. The user program determines the syntax file hierarchy. The context created by SDF when the user task is started is called the “system context”. This context uses the system administration defaults stored in the SDF parameter file.

- The system context is used by the command processor to read and analyze commands before they are executed by the command processor. The syntax file hierarchy and the SDF options are processed as described (see page 25ff). The identifier of the system context in the various macros is the default NULL address.
- The program context can be used by a program to read and analyze its statements before execution. The syntax file hierarchy is defined by the user program with OPNCALL for the level of the system and group syntax file(s). The SDF options can be modified locally within the program context by means of MODIFY-SDF-OPTIONS. The identifier of the program context is returned by the OPNCALL macro and is used by subsequent macro calls in this context.

The command processor can in no case execute a command that is being processed in a program context. It is not possible to create a program context with the CMD macro.

With the OPNCALL macro, a user program can

– compile commands (i.e. analyze them without executing them)

– read, compile or correct statements

– retrieve information

in or from the program contexts.

The program context has the following characteristics:

– The system syntax files may either be the current system syntax files of the system context of the user task, or the OPNCALL macro can specify explicitly that another basic system syntax file is to be opened. If system administration switches the current system syntax files of the system context dynamically with MODIFY-SDF-PARAMETERS, then this change also applies, after the next transaction, to the program context which has used the system syntax files in question.

– The group syntax file may either be the current group syntax file of the system context of the user task or be specified explicitly with the OPNCALL macro. It cannot be switched in the generated program context.

– The user syntax file is the current user syntax file of the user task. It is valid both for the system context and for the program context and can be dynamically switched using the MODIFY-SDF-OPTIONS command or statement. MODIFY-SDF-OPTIONS is read by CMDRST or, in the buffer, by CMDTST (EXECUTE=*YES).

– If user-defined syntax files are opened in the program context, SDF standard statements from the program context are used. SDF takes standard statements which are not available in the program context out of the $CMDPGM program defined in the basic system syntax file in the system context.

Changes to the user syntax file and to the options in the program context have no effect on the system context. A user program may open up 255 contexts, including the system context, at any one time.

The following SDF macros may be used in a program context or may process a program context:

– OPNCALL:
  creates the new context and determines the group and system syntax files to be activated. The identifier of the program context is returned to the user program in the form of a return code. This identifier is determined for each call which is processed in a program context.

– CMDTST, CMDRST, CMDCST and CMDSTA:
  the identifier of the program context is determined whenever work is carried out in the context.

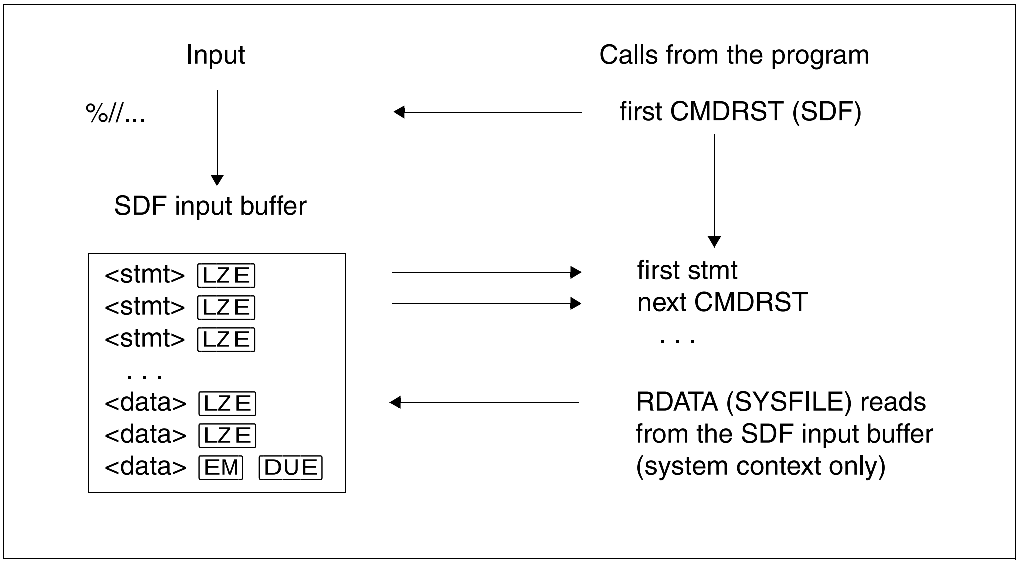
– CLSCALL:
  closes the program context. If no CLSCALL is issued, all program contexts are released when the program is terminated.

Notes on the programming context

The program context provides a new SDF working environment with its own input and syntax memories. For this reason, the following points must be noted:

– If blocked input is used for a list of statements, the input has only a local effect on the current context. Blocked input for CMDRST in a program context cannot call another context.

– Blocked input of statements may also include records. These can then be read with RDATA (SYSFILE) after CMDRST, but only in the *system* context:



RDATA (SYSFILE) can only read records from the terminal buffer of the system context. RDATA should therefore not be called after CMDRST in the program context.

– The EOF condition for statement input and the spin-off mechanism work only in the context in which the statements are read.

– The statements are not compiled in the environment of the current user task (CMDTST). The user must specify the simulated user task type (batch or dialog task) in the OPNCALL macro and the simulated procedure mode (procedure or primary) in the CMDTST macro.
The default is that CMDTST does not need to check the user task type or the procedure mode when checking the input. CMDRST reads the statements in the environment of the current user task.

CHKPT and RESTART-PROGRAM for program contexts

After a checkpoint of a program in the program context, the following behavior of the reopened hierarchies must be expected after RESTART-PROGRAM:

– If no special system syntax file was defined for the program context at the time of the CHKPT (see OPNCALL SFSYSTM=*STD), then the system syntax file of the current system context is used after the restart.
  If a special system syntax file was defined, then this file is reopened.

– If no special group syntax file was defined for the program context at the time of the CHKPT (see OPNCALL SFGROUP=*STD), then the group syntax file of the current system context which is assigned to the current profile ID is used after the restart.
  If a special group syntax file was defined, then this file is reopened.

– The user syntax files which were open at the time of the checkpoint are reopened after the restart.

After RESTART-PROGRAM, the contexts are reconstructed as if the system syntax file had been modified dynamically by system administration.

6.3 Format of the standardized transfer area

Standardized transfer areas are needed for three purposes:

1. SDF passes an analyzed statement to the program.
   Memory space for at least one such area must be reserved in the program. The space to be allocated must be large enough to accommodate any possible statement input that SDF has analyzed for the program.
   The address of this area is to be entered in the OUTPUT operand of the CMDRST and CMDTST macros, and in the INOUT operand of the CMDCST macro.
2. The program passes semantically incorrect statements back to SDF.
   No special memory space for this need be reserved in the program. The program uses the same area into which SDF had previously written the analyzed statement. After the program has detected semantic errors in the statement, it supplements the statement with information for the semantic error dialog, and returns it to SDF.
   The address of this area is to be specified in the INOUT operand of the CMDCST macro.
3. The program passes values to SDF to replace the specified operand values or the default value. For each statement in which such operand values can appear (see ADD-VALUE..., VALUE=<string>(OVERWRITE-POSSIBLE=*YES),... and ADD-OPERAND..., OVERWRITE-POSSIBLE=*YES) one such area is to be allocated in the program and be supplied with values by the program. The addresses of these areas are to be specified in the DEFAULT operand of the CMDRST and CMDTST macros.

i The format of the standardized transfer area has been changed as of SDF V4.1. The new layout is created using the CMDTA macro and must be used with the new CMDRST, CMDTST and CMDCST macros.

In all three cases, the transfer area has the same formal structure (see Figure 7).

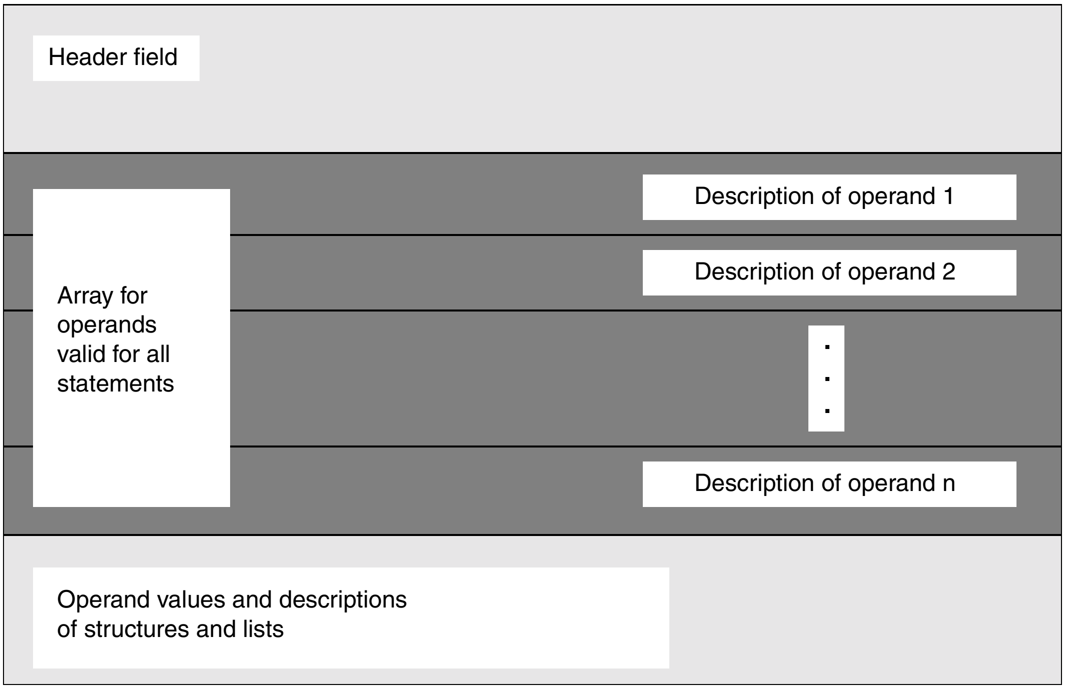
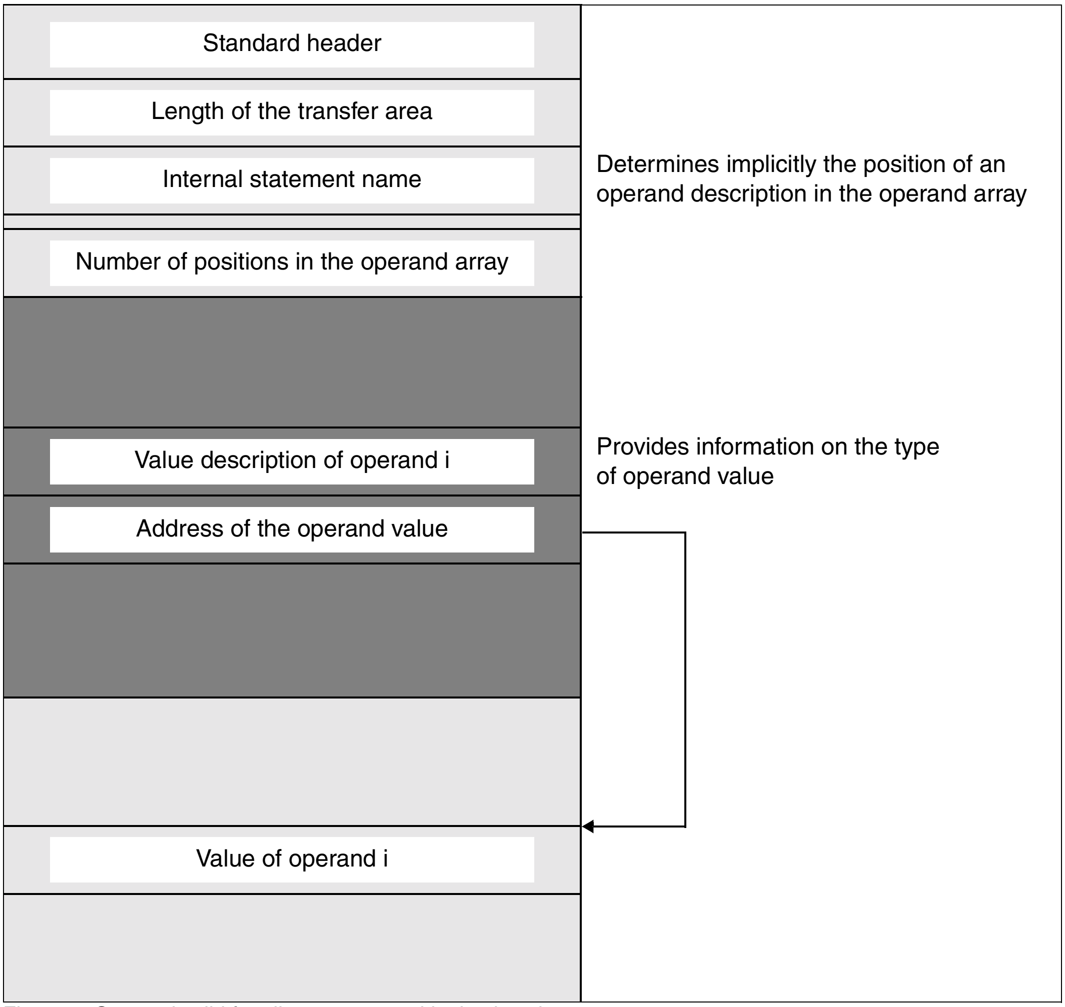

Header field
Description of operand 1
Description of operand 2
Array for operands valid for all statements
.
.
.
Description of operand n
Operand values and descriptions of structures and lists

Figure 7: Formal structure of standardized transfer area

The standardized transfer area begins on a word boundary. The header field is in bytes 0 through 39. It contains, amongst other things, the internal statement name and the statement version (see ADD-STMT ...,INTERNAL-NAME=...,STMT-VERSION=...). The array for operands valid for all statements, i.e. for operands defined with RESULT-OPERAND-LEVEL=1 (see ADD-OPERAND), begins with byte 40. It accommodates an 8-byte description for each of these operands. The operand descriptions are arranged in the order resulting from the operand positions established in the statement definition (see ADD-OPERAND ...,RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)). Each operand description contains, among other things, the absolute address where the associated operand value, or the description of the associated list or non-linearized structure, is stored in the transfer area. The description of a non-linearized structure contains an operand array describing the operands contained in the structure. It has the same format as the array for the operands valid for all statements.

In the simplest case, an operand has only one simple value (see Figure 8). An operand description with a simple value likewise contains the syntax attributes for this value.


Standard header
Length of the transfer area
Internal statement name
Determines implicitly the position of an operand description in the operand array
Number of positions in the operand array
Value description of operand i
Provides information on the type of operand value
Address of the operand value
Value of operand i
Figure 8: Operand valid for all statements, with simple value

If the operand value introduces a structure (Figure 9), there is a structure description for this value. This contains an operand array with descriptions for all operands of the structure, as well as for the operands from linearized substructures. The operand descriptions are arranged in the order resulting from the structure-oriented operand positions established in the statement definition (see ADD-OPERAND...,RESULT-OPERAND-NAME=*POSITION(POSITION=<integer>)).
The operand values corresponding to the operands of the structure may introduce further structures and/or consist of a list of values.

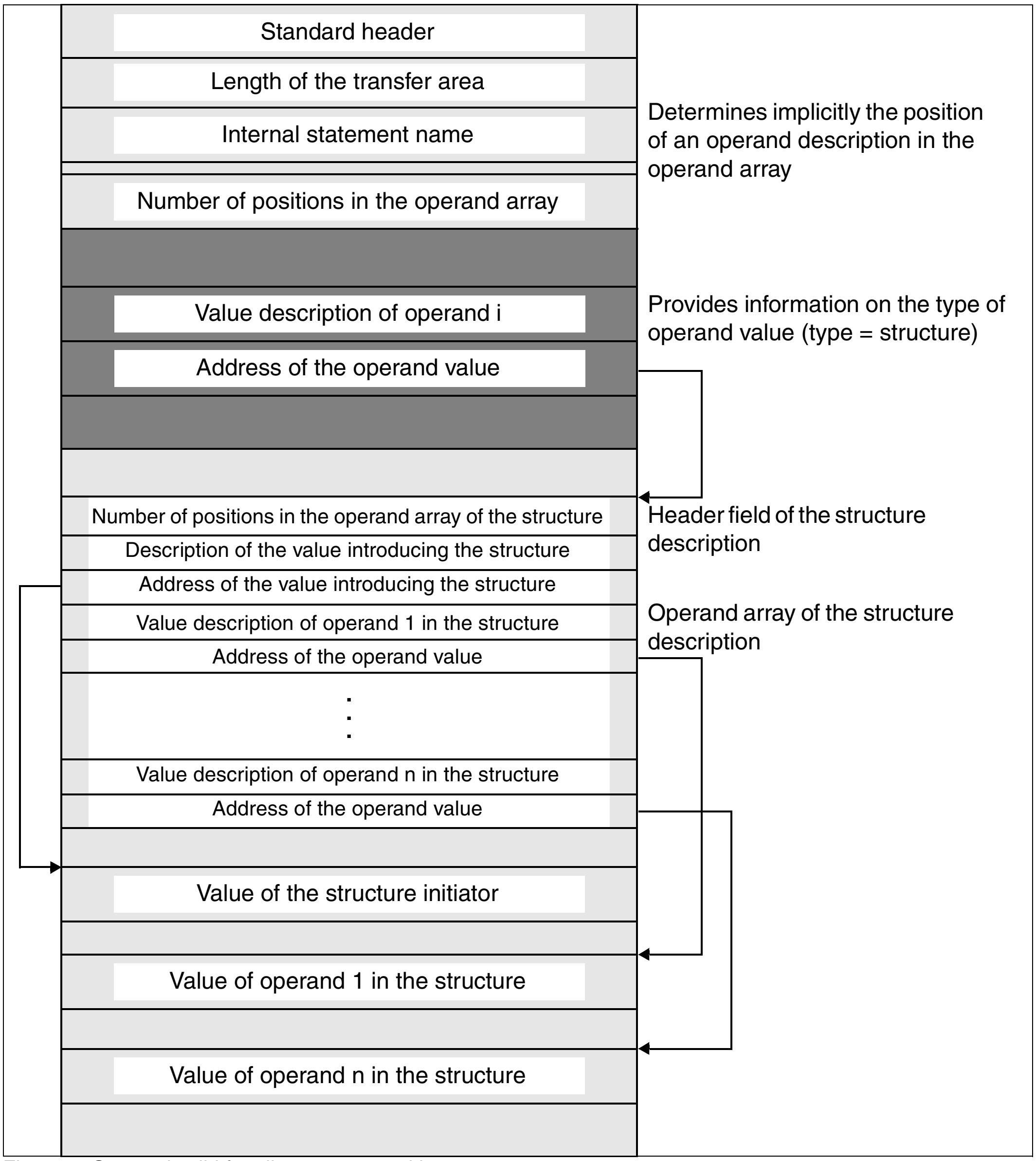

Standard header
Length of the transfer area
Internal statement name
Number of positions in the operand array
Determines implicitly the position of an operand description in the operand array
Value description of operand i
Address of the operand value
Provides information on the type of operand value (type = structure)
Number of positions in the operand array of the structure
Header field of the structure description
Description of the value introducing the structure
Address of the value introducing the structure
Value description of operand 1 in the structure
Address of the operand value
Operand array of the structure description
Value description of operand n in the structure
Address of the operand value
Value of the structure initiator
Value of operand 1 in the structure
Value of operand n in the structure

Figure 9: Operand valid for all statements, with structure

Values for operands defined with ADD-OPERAND ...,LIST-POSSIBLE=*YES (...,FORM=*NORMAL) are transferred in the form shown in Figure 10. A structure may be attached to a list element. In this case, “value of the list element” is a structure description.

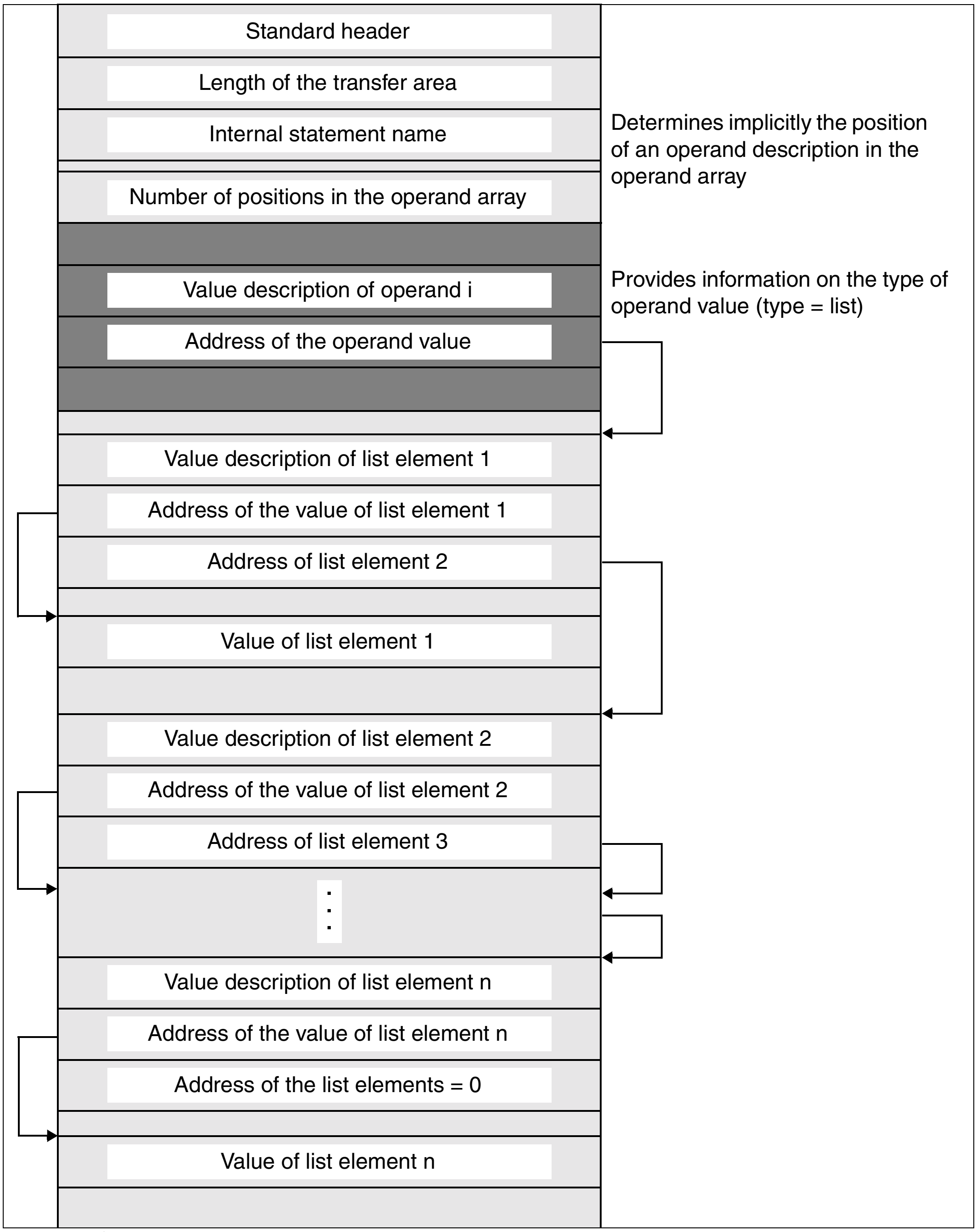

Standard header
Length of the transfer area
Internal statement name
Number of positions in the operand array
Determines implicitly the position of an operand description in the operand array
Value description of operand i
Address of the operand value
Provides information on the type of operand value (type = list)
Value description of list element 1
Address of the value of list element 1
Address of list element 2
Value of list element 1
Value description of list element 2
Address of the value of list element 2
Address of list element 3
Value description of list element n
Address of the value of list element n
Address of the list elements = 0
Value of list element n

Figure 10: Operand valid for all statements, with value list

Header field of a standardized transfer area

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 7	Standard header	Program	unchanged	Program
8 to 11	Length of the transfer area	Program	unchanged	Program
12 to 19	Internal name of the statement	SDF	unchanged	Program
20 to 23	Reserved	–	–	–
24 to 26	Version of the statement	SDF	unchanged	Program
27 to 35	Reserved	–	–	–
36 to 37	Number of positions in the operand array	SDF	unchanged	Program
38 to 39	Reserved	–	–	–

The internal name of the statement, the statement version and the number of positions in the operand array are stored in the syntax file in the statement definition (see ADD-STMT...,INTERNAL-NAME=...,MAX-STRUC-OPERAND=...,STMT-VERSION=...).
If the actual number of operands is larger than the value specified in MAX-STRUC-OPERAND, the actual number of operands in the transfer area is registered.

If the program passes default values for a statement, the version of the statement in the transfer area must agree with the version of the statement in the syntax file. If no version is registered in the syntax file, 000 (or binary zeros) must be entered, otherwise the default value will be rejected.

If the program supports several versions of a statement, a default value must be passed for each version.

Description of the operands

The operand array and the descriptions therein for the operands of a structure have exactly the same format as the one for the operands valid for all statements.

Byte	**Contents**	**Source of field contents in case of**		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 3	Value description	See below	See below	See below
0	Additional information			
1	Description of type	SDF	Unchanged	Program 1)
2	Global syntax attributes	SDF	-	-
3	Type-specific attributes	SDF	-	-
4 to7	Absolute address (stored aligned) of the value assigned to the operand or, in the case of structures or lists, of the further description	SDF	unchanged	Program 1)

1 Entry only for operands for which there are values to be converted, i.e. when bit 0 of the additional information is set

Additional information

The additional information is contained in the first byte of the value description. The following specifications regarding the additional information apply regardless of whether the additional information appears in an operand description, in the header field of a structure description, in the description of a list element or in an OR list description.

Bit	**Value**	**Meaning**	**Source of field contents in case of**		
			analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0	0	Value not available	SDF 1)	unchanged	Program 2)
0	1	Value available	SDF	unchanged	Program 2)
1	0	Value changeable	–	Program3)	–
1	1	Value not changeable	–	Program3)	–
2	0	Value not errored	–	Program	–
2	1	Value errored	–	Program	–

continued ➠

Bit	Value	Meaning	Source of field contents in case of		
			analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
3	0	Value is not to be used as default value	–	–	Program
3	1	Value is to be used as default value	–	–	Program
4 to 7	–	Reserved	–	–	Program

[1] For example: values for operands defined with ADD-OPERAND..., PRESENCE= *EXTERNAL-ONLY, or values in structures not referenced.

[2] 0 for operands for which there are neither operand values to be converted nor structures containing operands with values to be converted.
1 for operands for which there are either operand values to be converted or structures containing operands with values to be converted.

[3] All list values coming after the first changeable list value are considered changeable by SDF, regardless of whether bit 1 is set. In this way, list elements that have already been processed are protected against being overwritten.

Type description

The type description is contained in the second byte of the value description. The following specifications regarding the type description apply regardless of whether the type description appears in an operand description, in the header field of a structure description, in the description of a list element or in an OR list description.

Structure descriptions can be entered by a program in order to specify local default values. Default values can be entered:

– in the internal format (like the OUTPUT operand in the SDF-A statement ADD-VALUE) or
– in the external format as a string analogous to the operand description. The auxiliary data type <input-text> must be used, and the value is analyzed as if it had been entered via the user interface.

Value (decimal)	Meaning
1	Command rest
2	Integer
4	X-string
5	C-string
6	Name
7	Alphanumeric name

continued ➠

Value (decimal)	Meaning
8	Structured name
9	Label
11	File name
12	Partially-qualified file name
13	Time
14	Date
15	Composed name
16	Text
17	Catalog identifier (cat-id)
18	Input text
19	Structure
20	List
21	OR list
22	Keyword
23	Reserved for internal use
24	Volume serial number (VSN)
25	X-text
26	Fixed-point number
27	Device
28	Product version
29	POSIX path name
35	POSIX file name

Global syntax attributes

The description of the global syntax attributes of an operand value is found in the third byte of the value description. These are attributes which are specified for several data types (see ADD-VALUE, page 164ff).

Global attributes are always output attributes. They are ignored if the program supplies default values, or in correction dialogs.

Bit	Meaning when the bit is set
0	Value is a wildcard selector
1	Value is a wildcard constructor
2 to 7	Reserved

Syntax attributes specific to data type

The description of the data-specific syntax attributes is found in the fourth byte of the value description. These are attributes which are specified for one data type or for a small number of data types.

Data type-specific attributes are likewise output attributes. They are ignored if the program supplies default values, or in correction dialogs.

Data type FILENAME / PARTIAL-FILENAME:

Bit	Meaning when the bit is set
0	File name contains catalog ID
1	File name contains user ID
2	File name contains file generation or file generation group
3	File name contains version
4	File name is temporary
5 to 7	Reserved

Data type NAME:

Bit	Meaning when the bit is set
0	Name contains underscore (_)
1 to 7	Reserved

Data type COMPOSED-NAME:

Bit	Meaning when the bit is set
0	Composed name contains underscore (_)
1	Composed name contains catalog ID
2 to 7	Reserved

Data type TEXT:

Bit	Meaning when the bit is set
0	Text contains separator
1 to 7	Reserved

Data type X-TEXT:

Bit	Meaning when the bit is set
0	X-TEXT has an odd number of bytes
1 to 7	Reserved

Data type POSIX-PATHNAME / POSIX-FILENAME:

Bit	Meaning when the bit is set
0	POSIX path name of file name is absolute
1	POSIX path name or file name is relative
2	POSIX path name or file name was entered in single quotes
3 to 7	Reserved

Data type C-STRING:

Bit	Meaning when the bit is set
0	C-STRING contains a single quote
1 to 7	Reserved

Data type PRODUCT-VERSION:

Bit	Meaning when the bit is set
0	Product version contains correction status
1	Product version contains release status
2 to 7	Reserved

Header field of a structure description

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1	Number of positions in the operand array	SDF	unchanged	Program
2 to 3	Reserved			
4 to 7	Value description for the operand value introducing the structure Additional information			
4	Type description	–	see above	see above
5	Global syntax attributes	SDF	unchanged	Program
6	Type-specific syntax attributes	SDF	-	-
7		SDF	-	-
8 to 11	Absolute address (stored aligned) of the operand value introducing the structure	SDF	unchanged	Program

The number of positions in the operand array is stored in the statement definition in the syntax file (see ADD-VALUE...,STRUCTURE=*YES(..,MAX-STRUC-OPERAND=...).

The operand array belonging to the structure begins immediately after the header field (see Figure 9). It has exactly the same format as the one for operands valid for all statements.

List element

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 3 0 1 2 3	Value description Additional information Description of type Global syntax attributes Type-specific syntax attributes	 see above SDF SDF SDF	 see above unchanged - -	 see above Program[1)] - -
4 to 7	Absolute address (stored unaligned) of the value assigned to the list element or, in the case of structures, of the further description	SDF	unchanged	Program
8 to 11	Absolute address (stored aligned) of the next list element	SDF	unchanged	Program

[1] Specified only for operands with operand values to be converted, i.e. when bit 0 of the additional information is set.

In the last element of the list, the “absolute address of the next list element” has the value 0 (see Figure 10).

An OR list consists of a single element. The “absolute address of the next list element” is redundant in this case.

For an operand defined with LIST-POSSIBLE=*YES(FORM=*NORMAL), the number of list elements must be restricted (LIMIT=...) so as to prevent an overflow in the standardized transfer area. The size of a list in the standardized transfer area can be calculated by using the following formula.

$n * (10 + 2 + l)$

where:	n	is the number of list elements, and
	l	is the length of a single list element (rounded to a multiple of 2).

Example:

```
ADD-OPERAND ... LIST-POSSIBLE=*YES(LIMIT=100,FORM=*NORMAL)
  ADD-VALUE *NAME(1,8)
```

A list that is defined in this way can occupy up to 2000 bytes in the standardized transfer area (100 * (10 + 2 + 8) = 2000).

How values are stored

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1	Length specifications	SDF	unchanged	Program
2 to 3	Reserved	-	-	-
4 to ...	Value	SDF	unchanged	Program

How the values are passed depends on the definition in the syntax file (see ADD-VALUE..., OUTPUT=*NORMAL(...). In this regard, the following points apply:

– A value defined with ADD-VALUE TYPE=*INTEGER(...,OUT-FORM=*BINARY) is stored as a signed four-byte string.

– A value defined with ADD-VALUE TYPE=*TIME is stored as a four-byte string, with 2 bytes (binary) for the hours and one byte each for the minutes and seconds.

6.4 SDF macros

6.4.1 Macro types

To avoid confusion, it should be noted that the term “macro type” does not refer to the terms “action macro” and “definition macro”. These two terms refer to the function of a macro:

An **action macro** is a macro from which the execution of certain actions is expected; a typical example is the **PRNT** macro, which controls a print operation.

A **definition macro** is a macro from which definitions (such as addressing aids, DSECTS), rather than actions, are expected. The **CUPAB** macro is an example for the generation of symbolic names for addressing operand tables.

Macros are divided into **types** according to the way in which they **pass operands**, namely the R type (where the operands are passed in registers), the S type (where they are passed in memory), and the so-called O type, which do not have a specific type assignment. Macros of type S can be either action macros (MF=E) or definition macros (MF=D).

R type macros

Operation	Operands
macro	$\left\{\begin{matrix}\text{operand1}\\\text{(r1)}\end{matrix}\right\}, \left\{\begin{matrix}\text{operand2}\\\text{(r2)}\end{matrix}\right\}$

A macro is an R type macro if all necessary operand values can be loaded into the two registers (R0 and R1) used for this purpose. An R type macro therefore generates no parameter area.

S type macros

For an S type macro, the operand values specified in the macro are passed to the function module in the form of a data area which is part of the macro expansion. It contains the data and memory definitions (DC and DS statements) necessary for passing the operand values.

Name	Operation	Operands
[opaddr]	macro	$\left\{\begin{array}{l} \text{operand}_1\text{,.,operand}_n\text{[,MF=}\left\{\begin{array}{l}\underline{\text{S}}\\ \text{L}\\ \text{C[,PREFIX=p],[MACID=mac]}\\ \text{(C,p)}\\ \text{D[,PREFIX=p]}\\ \text{(D,p)}\\ \text{M[,PREFIX=p],[MACID=mac]}\end{array}\right\} \\ \\ \text{MF=}\left\{\begin{array}{l}\text{(E,opaddr)}\\ \text{(E,(r))}\\ \text{E[,PARAM=addr]}\end{array}\right\}\end{array}\right\}$

The S type supports specification of the MF operand. In accordance with the different functions of the various macros, the MF operands can be represented in different ways:

– Standard form (MF=S):

  MF=S is the default value. The command section is generated first, followed by the data area, taking into account the operand values specified in the macro. The data area does not contain any field names or any explanatory equates. The standard header is initialized.

– L form (MF=L):

  Only the data area is generated, taking into account the operand values specified in the macro. The data area does not contain any field names or any explanatory equates. The standard header is initialized. The macro is usually issued in the definition section of the program. For shared-code programming, this call must not be in the invariant part of the program if it contains variable data. In the invariant part of the program the data area is initialized with constant values, copied to a local data area before the E form is called, and modified in this local area if necessary. Modification can be handled, for example, with the M form if it is available for the relevant interface.

– C form (MF=C / MF=(C,p)):

  Only the data area is generated; any operand values specified in the macro are not evaluated. Each field has a field name and, if necessary, explanatory equates. The first characters of each field name can be specified (p = a prefix which replaces the first characters). The data area is terminated with a length equate. The standard header must generally be initialized by the user.

– D form (MF=D / MF=(D,p)):

  A DSECT is generated. Each field has a field name and, if necessary, explanatory equates. The first characters of each field name can be specified (p = a prefix which replaces the first characters). The DSECT describes the structure of a memory area without taking up any memory space itself. It is terminated with a length equate. There is no switch to the initial location counter. The symbolic name specified for the DSECT is entered in an ESD record. The location counter is set to 0.

– E form:

  Only the commands necessary for calling the function module are generated. The command section usually ends with an SVC. In the macro, the address of the data area must be specified with the operand values; the following forms are commonly used:

MF=(E,addr)	addr = address of the data area
MF=(E,(r))	r = register containing the address of the data area
MF=E,PARAM=addr	addr = address of the data area

  No further operands are evaluated in the E form

– M form (MF=M)

  Commands (such as MVCs) are generated. During program execution, these overwrite fields in a data area already initialized with MF=M or, in the case of shared-code programming, in a local copy of the data area initialized with MF=L with the operand values specified in the macro. This therefore makes it easy to dynamically modify the operand values with which a macro is called, in order to adapt it to the requirements of the program.
  Since the commands generated for this purpose use addresses and equates in the C form or D form, care must be taken, when using the M form, that these names are available for addressing of the operand list to be modified. In particular, care must be taken that the operands PREFIX and MACID, if specified in a macro with MF=M, receive the same values as those specified in the related macro with MF=C or MF=D.

A detailed description of the MF forms can be found in the “Executive Macros” manual [8].

O type macros

Some macros cannot be classified as R type or S type: they are macros without a type assignment.

O type macros are, for example, macros for which **one** register (often only R1) is specified in the operand field. This register contains the start address of an operand list.

The operand list is defined in the data section of the program (with DC statements) and contains the actual operand values.

6.4.2 Standard header

All new macros and, generally, all macros extended to handle the 31-bit interface use the standard header to identify their interface.
The standard header is an 8-byte field at the beginning of the operand list (parameter list) with the (standardized) designation of the interface and 4 bytes to contain the return code. The standard header is generated and initialized, i.e. filled with the valid values for UNIT, FUNCTION and VERSION by the macro. For E form macros which refer to the operand list, the user may have to initialize the standard header. Further details can be found in the macro description.

Format of the standard header:

Byte	Contents and meaning
0 - 1	Name of the function unit (UNIT) with the required function
2	Name of the function (FUNCTION) within the function unit
3	Name of the version (VERSION) of the function
4	Subvalue 2 of the return code (SC2).
5	Subvalue 1 of the return code (SC1).
6 - 7	Main value of the return code (Maincode).

The following return code values are conventions:

(SC2)	SC1	Maincode	Meaning
00	00	0000	Function executed successfully. There is no additional information for MAINCODE.
01	00	0000	Function executed successfully. No further actions were necessary.
00	01	FFFF	The requested function is not supported (invalid specification for UNIT or FUNCTION in the standard header). This error cannot be rectified.
00	02	FFFF	The requested function is not available. This error cannot be rectified.
00	03	FFFF	The specified interface version is not supported (invalid version specification in the standard header). This error cannot be rectified.
00	04	FFFF	The parameter list is not aligned on a word boundary
00	41	FFFF	The subsystem does not exist; it must be generated explicitly.
00	42	FFFF	The calling task is not connected to this interface; it must be connected explicitly.
00	81	FFFF	The subsystem is currently not available.
00	82	FFFF	The subsystem is in the DELETE or HOLD state.

The maincode shows the result of execution of the function. SC1 classifies the maincode. SC2 either subdivides the errors into error classes or contains additional diagnostic information.

In all new macros introduced since BS2000 V9.0, the return code should only ever be passed in the standard header. However, for a transitional period it can, for certain macro interfaces, also be passed in register R15 or in both the standard header and register R15. In order to check whether a return code was passed in the standard header, the return code field should be preset to X'FFFFFFFF'.

6.4.3 Metasyntax for macro call formats

The following table explains the specific metacharacters and declarations that are used to represent formats of macro calls:

Formal notation	**Explanation**	**Example**
UPPERCASE LETTERS	Uppercase letters designate constants and must be entered by the user exactly as shown.	DATATYPE=CATID Required input: DATATYPE=CATID

Lowercase letters	Lowercase letters designate variables that must be replaced by the user with current values on input, i.e. they represent values that may differ from case to case.	SHORTST=integer
		Required input:
		SHORTST=10 or
		SHORTST=54
		etc.

$\left\{ \quad \right\}$	Braces enclose multiple alternatives, i.e. one of the enclosed values must be selected.	CCSNAME= $\left\{\begin{array}{l}\text{*NO}\\ \text{*EXTEND}\\ \text{name}\end{array}\right\}$
		Required input:
		CCSNAME=*NO or
		CCSNAME=*EXTEND or
		CCSNAME=xxxxx

/	A slash is used as a separator between two alternatives from which one must be selected	SYMTYP=CSECT/ENTRY
		Required input:
		SYMTYP=CSECT or
		SYMTYP=ENTRY

<u>Underscoring</u>	Underscoring is used to indicate the default value, i.e. the value assumed by the system if no entry is made by the user.	PATTERN=<u>*NO</u> / addr
. . .	Ellipses are used to indicate repetitions, i.e. the preceding unit may be repeated in succession more than once.	(module,...) Required input: (MODULE1) or (A,B,C) or (A1,A2,A3,A4) etc.

list-poss(n)	Means that a list can be constructed from the values that follow list-poss(n). If (n) is specified, the list may include a maxi-mum of n elements. If the list contains more than one element, it must be enclosed within paren-theses.	DEVCLAS=list-poss(2):
		DISK / TAPE
		Required input:
		DEVCLAS=DISK or
		DEVCLAS=TAPE or
		DEVCLAS=(DISK,TAPE)

Table 4: Elements of metasyntax

Formal notation	**Explanation**	**Example**

[ ]	Square brackets enclose optional entries, i.e. specifications that may be omitted. If a comma is included with an optional entry within the square brackets, it must be entered only if that optional entry is used and may be omitted for the first operand. If the comma is located outside the brackets, it must always be entered even if none of the optional entries are made (parentheses, if any, must always be specified).	`MAP=S[YSOUT]`
		
		Required input:
		`MAP=SYSOUT or`
		in abbreviated form:
		`MAP=S`

equals sign =	The equals sign (=) separates the operand from the operand value.	`SPIN=*NO / *YES`
< >	Angle brackets identify the data type of the operand.	`INOUT=<var:pointer>`

Table 4: Elements of metasyntax

Data types of operand values

Data type	Character set	Notes
c-string	EBCDIC characters	Must be enclosed within single quotes and specified without a leading “C”. Single quotes appearing in the string must be duplicated. The specification is followed by the meaning of its contents in SDF notation, separated by a colon. The suffix n..m defines the input length in bytes. Example: Shown in the syntax diagram as: `SELECT`=<c-string 1..255> Input: `SELECT`='testfile'
x-string	Hexadecimal 00..FF	Must be enclosed within single quotes and preceded by the letter X, i.e. in the form X´xxxx´. The suffix n..m defines the input length in bytes. Example: Shown in the syntax diagram as: `PASSWORD`=<x-string 1..10> Input in Assembler: `PASSWORD=X'FF00AA1122'` Input in C/C++: `PASSWORD=0xFF00AA1122`
name	A..Z, 0..9, $, #, @	Identifier. The appropriate format is given under the relevant operand description. Example: Shown in the syntax diagram as: `PARAM=<name 1..8>` Input: `PARAM=MYPARAM`
label	A..Z 0..9 $,#,@	Designates a label. Example: `OUTAREA=structure (2):` (1) address: <label>
integer	0..9,+,-	A “+” or “-” may only be the first character. The suffix n..m defines the permitted range of values. Example: Shown in the syntax diagram as: `MAXLEN=<integer 1..255>` Input: `MAXLEN=200`
var:		Introduces the specification of a variable. The data type of the variable is given after the colon (see following table). Example: Shown in the syntax diagram as: `SELECT = <var: char:255>` Input (to specify the variable name): `SELECT=VARSELECT`
reg:		Introduces a register specification (Register 0..15). The data type of the register contents is given after the colon. A register or register EQUATE may be used on input.

Data types of operand values

Data types of variables and register contents

Data type	Description
char:n	Designates a character string of length n. The string may be shorter, but only if a specific condition is satisfied. The applicable conditions are given under the respective operand descriptions. If the length is not specified, n=1 is assumed.
int:n	Designates an integer with a length of n bytes, where n<=4. If the length is not specified, n=1 is assumed.
bit:n	Designates a bit string of length n. If the length is not specified, n=1 is assumed. (in C: declaration as 'unsigned')
enum-of E:n	The variable is the enumeration E with a length of n bytes, where n<=4. If the length is not specified, n=1 is assumed.

pointer	Pointer (the address is passed).
	Area reference with the aid of pointer variables:
	with MF=L: address notation:
	`...,MF=M,<operand>=A(area)`
	with MF=M: register notation, address notation or symbolic address of a field containing the address of an area:
	`LA 2,area`
	`... MF=M,<operand>=(2)`
	or
	`... MF=M,<operand>=A(area)`
	or
	`LA 2,area`
	`ST 2,areaADD`
	`... MF=M,<operand>=areaADD`
	`areaADD DS A`

Data types of variables and register contents

6.4.4 Functional overview

Data definitions

CMDALLW	Generates a list of valid operations for the macros CMDRST and CMDTST
CMDANALY	Generates EQUATE statements for return codes
CMDMEM	Generates a transfer area for status information
CMDRETC	Generates a DSECT for command return codes
CMDTA	Generates a transfer area for an analyzed statement

Statement processing

CMDCST	Initiates the semantic error dialog
CMDRST	Reads and analyzes a statement
CMDTST	Analyzes a statement

Calls in simultaneously open syntax file hierarchies

OPNCALL	Creates a program context
CLSCALL	Closes a program context

Miscellaneous

CMDRC	Sets command return codes
CMDSEL	Creates selection mask for guided dialog
CMDSTA	Specifies information about active syntax files
CMDVAL	Checks an input value for a data type or for a wildcard search string match
CMDWCC	Check syntax of wildcard patterns and perform pattern matching
CMDWCO	Generate new names with the aid of wildcard constructors
TRCMD	Analyzes a command

6.4.5 Macro descriptions

The macros are described in alphabetical order.

CLSCALL
Close program context

CLSCALL closes the program context opened with OPNCALL. The specified context identifier is then no longer valid.

Operation	Operands
CLSCALL	CALLID = addr / reg [,MF = { L / (E,(1)) / (E,opaddr) }]

CALLID = addr / (reg)
The address of a 4-byte field or the register which contains the context identifier.

MF=
defines special requirements for macro expansion (see the manual “Executive Macros” [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This is either in register 1 or is specified directly.

Register usage

Register 1: address of the parameter list

Return information and error flags

Register 15 contains a return code in the right-most byte:

X’00’	normal execution
X’08’	call ID does not exist
X’40’	call ID in incorrect environment

CMDALLW
Generate list of valid operations

The CMDALLW macro generates a list of the valid operations that can be used when the macros CMDRST and CMDTST are called and STMT=A(identifier) is specified for these macros.

Operation	Operands
CMDALLW	LIST = (name, ...) [,SIZE = <u>1</u> / i] [,PREFIX = <u>C</u> / p] [,MACID = <u>MDA</u> / mac] [,MF = <u>L</u> / D / C]

LIST = (name,...)
Outputs the list of valid statements for the macros CMDRST and CMDTST. The layout is generated as required by the STMT parameter in these macros. Only those statements for which the internal statement name is listed are valid. The internal name is stored in the statement definition in the syntax file (see ADD-STMT). The internal name is at least one and no more than 8 bytes long. Irrespective of the specifications here, the SDF standard statements are always valid.

SIZE = <u>1</u> / i
Specifies the number of names that can be entered in the list of valid operations. i must be an integer greater than zero.

For a description of the parameters PREFIX, MACID and MF see section “Macro types” on page 379ff.

Possible combinations

```
[label]  CMDALLW MF=D[,SIZE=i][,PREFIX=p][,MACID=mac]
[label]  CMDALLW MF=C[,SIZE=i][,PREFIX=p][,MACID=mac]
[label]  CMDALLW MF=L[,LIST=(name,...)]
```

Return information and error flags

The macro does not return a return code.

Examples

– Request memory space for a maximum of 20 names:

```
ALLC  CMDALLW MF=C,SIZE=20
```

– Declare the model for the list of valid statements in a DSECT:

```
ALLD  CMDALLW MF=D,PREFIX=D
```

– Declaration of a list of 20 names (in this example) in the static code. In this instance, the length of the memory space is defined by the number of elements in the list.

```
ALL20 CMDALLW LIST=(A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T),-
      PREFIX=I
```

– The space in memory can be initialized by the static code. Processing continues with a pointer to the DSECT:

```
MVC     ALLC(IMDALL#),ALL20
L       R5,ALLC
USING   ALLD,R5
MVC     DMDALNR,=H'15'
MVC     DMDAL1+10*DMDALNL,=CL8'V'
...
```

(IMDALL#, DMDALNR, DMDAL1 and DMDALNL are generated by CMDALLW with PREFIX=I and D)

– Call with static lists:

```
        ...
        LA 2,MYLIST
        LA 1,RST1
        USING RSTD,1
        CMDRST MF=M,PARAM=RST1,STMT=(2)     * Update parameter area RST1
                                            * using register notation
        ...
RST1    CMDRST MF=L,STMT=A(MYLIST)          * direct initialization of
                                            * parameter area RST1
MYLIST  CMDALLW LIST=(A,B,C)

RSTD    CMDRST MF=D
```

CMDANALY
Generate EQUATE statements for return codes

The CMDANALY macro generates a series of EQUATE statements. These define the return codes for the macros CMDSTA, CORSTMT, RDSTMT and TRSTMT. This is not necessary for the macros CMDRST, CMDTST and CMDCST (see, for example, “Migration from RDSTMT to CMDRST” on page 416).

Operation	Operands
CMDANALY	[P = <u>CMD</u> / mac]

P = <u>CMD</u> / mac
Prefix of the macro identifier. By default, the macro identifier starts with the string ’CMD’.

Notes on certain return codes

1. DIALOGUE_IMPOSSIBLE_:
   No correction dialog could be provided for correction of the current input:
   - the input was unknown
   - batch task
   - spin-off state
   - procedure mode and PROCEDURE-DIALOG=*NO specified
2. DIALOGUE_REJECTED_:
   A correction dialog was started but the requested correction was rejected by the caller in guided or unguided mode.

CMDCST
Initiate semantic error dialog

The CMDCST macro causes SDF to conduct a dialog with the user in which the user corrects semantic errors in a statement. Immediately beforehand, SDF has analyzed the statement and passed it to the program as being syntactically correct.
Secret operand values that were entered in blanked input fields by the user must be repeated during the correction process.

Prerequisites for the semantic error dialog are:

- The program is running in an interactive task and an error dialog was permitted in the syntax analysis, i.e.:
  - temporary or permanent guided dialog must be set
  - the SDF option PROCEDURE-DIALOGUE=*YES must be set in procedures
  - if CMDCST is called after CMDTST, DIALOG=*ERROR must be set for CMDTST
- The same syntax file is available as when the statement was first analyzed (no intervening change of syntax files).

If these prerequisites are not satisfied, SDF rejects the dialog (error code X’20’).

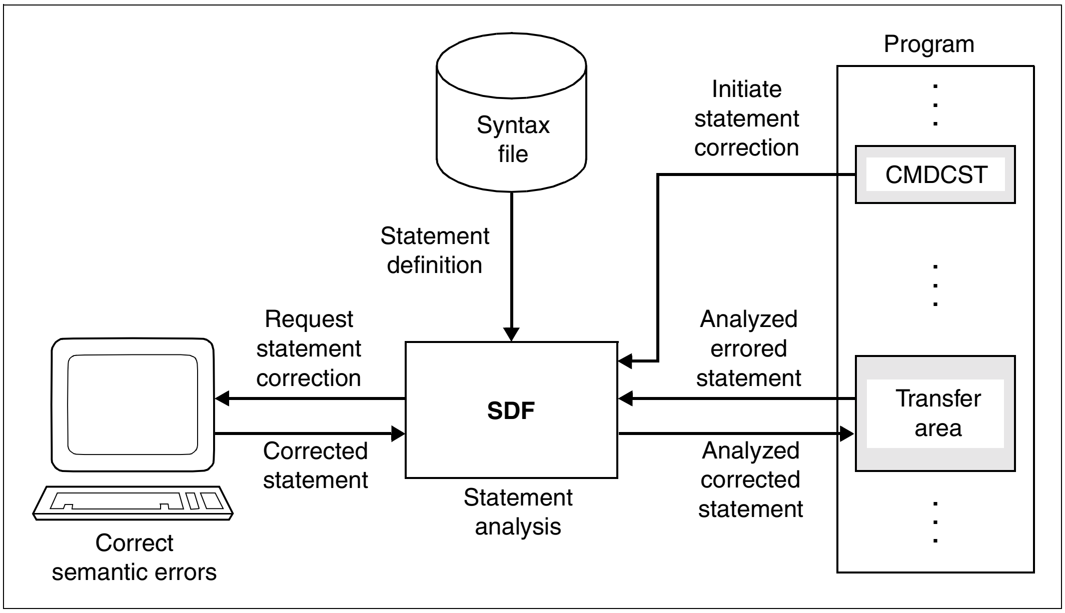


Figure 11: Effects of the CDMCST macro

Operation	Operands
CMDCST	INOUT = <var: pointer>
	,MESSAGE = <var: pointer>
	,DEFAULT = *NO / <var: pointer>
	,INVAR = *NO / <var: pointer>
	,CALLID = *NO / <var: pointer>
	,CCSNAME = *NO / *EXTEND / <c-string 1..8> / <var: char:8>
	,MF = C / D / L / M / E

INOUT=<var: pointer>
Address of the standardized transfer area, which must begin on a word boundary. It contains the results of analysis of the incorrect statement, passed previously to the program by SDF. The program has identified the operand values it has found to be errored. SDF does not accept changes in input values made by the program. Following the error dialog and renewed analysis, SDF stores the modified analysis results back into this area (see section "Format of the standardized transfer area" on page 365).

MESSAGE=<var: pointer>
Address of the text to be output for the error dialog. In guided dialog, this text is integrated into the statement menu.This area must be aligned on a halfword boundary and have the following format:

2 bytes:	Absolute length of record (n+4)
2 bytes:	(Reserved)
n bytes:	Message text

The maximum text length is 400 characters. Only the first 280 characters are displayed on an SDF-formatted screen. The menu mask can be destroyed If the text contains screen control characters.

DEFAULT=
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
– operand values entered or
– operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES,... and ADD-VALUE...,VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value.

In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:

In the MODIFY statements entered, SDF-A replaces the value *UNCHANGED by the current value.

***NO**

The operand values entered are not replaced by values generated dynamically by the program.

<var: pointer>

Address of a list aligned on a word boundary which contains addresses containing conversion descriptions for statements. A formatted transfer area of the type “structure” is used as conversion description (see section “Format of the standardized transfer area” on page 365ff). Only one conversion description can be specified for each statement. A conversion description contains, among other things, the internal statement name and information regarding which of the operand values entered are to be changed and what values they are to be changed to. The list of addresses of conversion description is structured as follows:

2 bytes:	Number of conversion descriptions in the list (n)
2 bytes:	(Reserved)
4 bytes:	Address of the first conversion description
	. . .
4 bytes:	Address of the nth conversion description

The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

If the operands to be given default values are in a structure introduced by a value defined with LIST-ALLOWED=*YES (see ADD-VALUE), the following situation may arise: the conversion description contains several list elements to which structures with operands to be defaulted are attached. On the other hand, the user likewise enters several list elements to which structures with operands to be defaulted are attached. SDF first tries to match the structures entered by the user to those specified in the conversion description by means of the values introducing the structures. If an unambiguous allocation cannot be made on the basis of the values introducing the structures because none of the values entered matches any of the ones in the conversion description or because the user has entered the matching value more than once, the allocation is then made on the basis of the list position of the introductory value.

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the user-defined operands, all operands having default values and all operands currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement available to a user who has certain privileges and who is working in the selected dialog mode.

In contrast to LOGGING=*INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form **does not** mask out keywords and secret operands.

***NO**
The INVARIANT-INPUT form of the statement is not stored.

<var: pointer>
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword. The contents of the buffer are then as follows:

2 bytes:	Maximum length of the buffer
2 bytes:	Output length written by SDF (n+4)
2 bytes:	(Reserved)
n bytes:	INVARIANT-INPUT form of the statement, starting at the seventh byte

CALLID =
Refers to a context (= syntax file hierarchy) which was opened by an OPNCALL macro. The name of the syntax file hierarchy (callid) must have the 4-byte value returned by SDF to the field which was designated by the CALLID operand in the Open Context macro. This function applies to the OPNCALL and CLSCALL macros.

***NO**
The current syntax file hierarchy (context) of the task is used for analyzing the statement.

<var: pointer>
Address of the call check field or register containing this address.
The area must be aligned on a word boundary.

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the “XHCS” manual [11]). This parameter has no effect on message output.

<u>*NO</u>
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

<c-string 1..8> / <var: char:8>
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long and can be passed as a c-string constant or as a string variable.

Description of the MF, PARAM, MACID and PREFIX parameters: see the “Executive Macros” manual [8] for details.

Return information and error flags

The format of the transfer area is described on page 365ff. The format used for the transfer area up to SDF V4.0 can be found in chapter “Appendix” on page 593.

Information on the INVARIANT-INPUT form of a statement can be found under the description of the CMDRST macro on page 405.

The return code is passed in the standard header of the parameter list.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	Normal termination
00	20	0004	Unrecoverable system error
	01	0008	Parameter error:
00			– wrong parameter list
01			– INOUT
03			– DEFAULT
04			– MESSAGE
06			– INVAR
07			– CALLID
00	40	000C	Transfer area too small
00	40	0010	End-of-file (EOF), or error in statement, end-of-file (EOF) was then detected
00	40	0014	Error in statement, a command was then detected
00	40	0018	Statement is correct but the default values provided by the system are errored
00	40	001C	Error in statement, //STEP was then detected
00	40	0020	Error dialog not possible
00	40	0024	Error dialog rejected by user
00	40	002C	END statement was read
00	40	0034	Error in statement, END was then detected
00	40	0040	Specified CALLID not found
00	40	0044	Syntax file in DSSM catalog not found
00	40	005C	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
00	20	0064	XHCS error during statement entry

Migration from CORSTMT to CMDCST

Migration from CORSTMT to CMDCST is only necessary when the user wishes to use the new functions of CMDCST, CMDRST and CMDTST. In this case the same points must be borne in mind as for CMDRST (see “Migration from RDSTMT to CMDRST” on page 416ff).

The macro return code is transferred in the standard header of the parameter list. The maincode of CMDCST is equivalent to the values which for CORSTMT were transferred in the right-most byte in register 15. The creation of equates for return codes with the CMDANALY macro is no longer necessary because they are automatically created with CMDCST MF=D. The following list shows the old (CORSTMT) and new (CMDCST) field names:

CORSTMT	**CMDCST**
&P.NOERR	&PREFIX.MDCSUCCESSFUL
&P.SYERR	&PREFIX.MDCSYSTEM_ERROR
&P.PAERR	&PREFIX.MDCPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDCAREA_TOO_SMALL
&P.EOF	&PREFIX.MDCEOF
&P.SCMD	&PREFIX.MDCSTMTERROR_CMD
&P.DFLT	&PREFIX.MDCWRONG_DEFAULTS
&P.STEP	&PREFIX.MDCSTMTERROR_STEP
&P.DIMP	&PREFIX.MDCDIALOG_IMPOSSIBLE
&P.DREJ	&PREFIX.MDCDIALOG_REJECTED
&P.END	&PREFIX.MDCEND_STMT
&P.EERR	&PREFIX.MDCSTMTERROR_END
&P.NOPRG	&PREFIX.MDCPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDCINVALID_CALLID
&P.NOFND	&PREFIX.MDCSYNTAX_FILE_NOT_FOUND
&P.ITRC	&PREFIX.MDCINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDCXHCS_ERROR

CMDMEM
Generate transfer area for status information

The CMDMEM macro generates a DSECT or a CSECT. These define the transfer area into which, after the CMDSTA macro has been called, information is written regarding the activated syntax files and the current specifications for command/statement input and processing.

Operation	Operands
CMDMEM	<u>D</u> / C
	[,P = <u>CMD</u> / prefix]

D / C
Specifies whether a DSECT (D) or a CSECT (C) is generated.

P = <u>CMD</u> / prefix
Specifies a character string that is to be concatenated with the beginning of all names in the program section. The string may be up to three characters long. Unless otherwise specified, the string “CMD” will be concatenated with the name of the section.

i The size of the transfer area has been increased as of SDF V2.0A. The CMDSTA macro supplied by SDF V4.0A uses this new layout. The transfer area used by the caller must match the call, i.e. the function call and the definition of the transfer area must have been created with the same SDF environment. Non-compliance with this can lead to a parameter error, which is returned by the CMDSTA macro. Earlier versions of the CMDSTA macro used with the old transfer area are still supported.

CMDRC
Set command return codes

The CMDRC macro enables user programs to save certain values as “command return codes” for the program. When the program is ended, the command processor returns these values instead of the official return codes. The values may be interpreted as the command return codes of the program or those of the command with which the program was started (START-PROGRAM or RESUME-PROGRAM). This command return code can then be used further by other system functions (e.g. SDF-P).

The return code defined with CMDRC is not available until the program is ended, because until then it is buffered by the command processor. Only the return code most recently set by CMDRC is of significance. The official command return code in the command processor is not modified by CMDRC during program execution.

Operation	Operands
CMDRC	RCADDR = addr [,MF = $\left\{\begin{array}{l}\text{L}\\ \text{(E,(1))}\\ \text{(E,opaddr)}\end{array}\right\}$]

RCADDR=addr
Address of the command return code.
The DSECT for the command return code is created with the CMDRETC macro (see page 404).

MF=
defines special requirements for macro expansion (see the “Executive Macros” manual [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This is either in register 1 or is specified directly.

Register usage

Register 1: address of the parameter list

Return information and error flags

Register 15 contains a return code in the right-most byte.

X’00’	normal execution
X’01’	abnormal execution

i If CMDRC is not called, SDF will independently generate the applicable return codes indicated above in accordance with the UNIT parameter of the TERM macro.

CMDRETC
Generate DSECT for command return codes

This macro generates a DSECT or CSECT for the input area of the CMDRC macro.

Operation	Operands
CMDRETC	[PREFIX = <u>C</u> / p]
	[,MACID = <u>MDR</u> / mac]
	[,MF = <u>D</u> / C]

See section “Macro types” on page 379ff for a description of the parameters.

CMDRST
Read and analyze statement

The CMDRST macro causes SDF to

– read in a program statement from SYSSTMT. (For the system file SYSSTMT the same assignment applies as was made for the system file SYSDTA. With regard to continuation lines, continuation characters and notes, the same rules apply to statement input from SYSSTMT as to command input from SYSCMD.)

– analyze the statement read in, and

– pass the results of the analysis to the program.

This presupposes that an activated syntax file contains the definition of the program and its statements.
The input length for a statement read via CMDRST is 16348 bytes.

In addition to normal reading and analysis of statements, CMDRST can also read records. However, this function is intended solely for purposes of migration. SDF merely passes the records on, i.e. no formatting takes place.

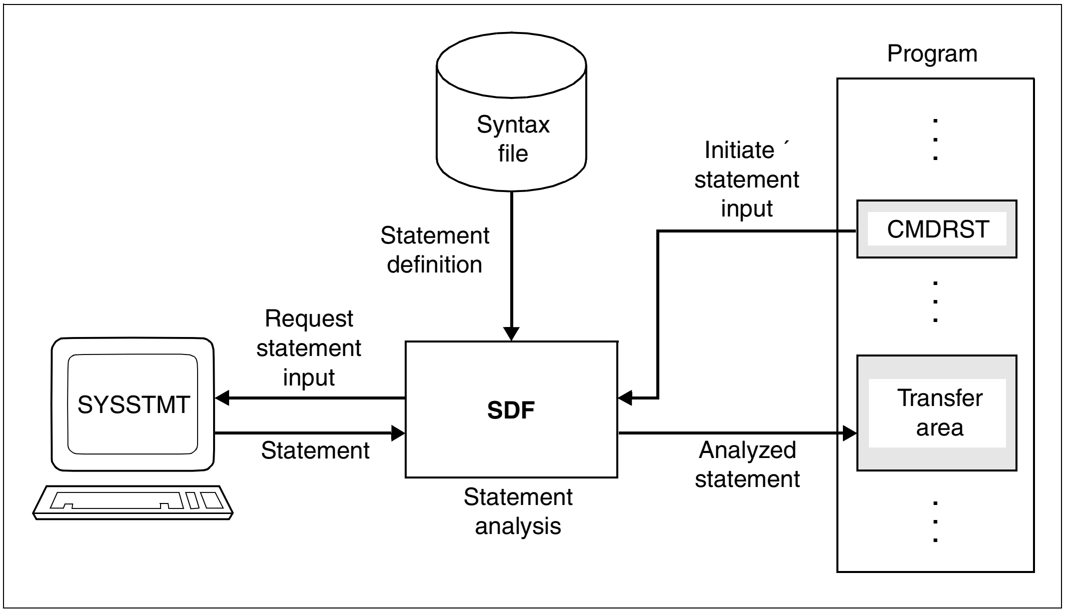


Figure 12: Effects of the CMDRST macro

Operation	**Operands**
CMDRST	PROGRAM = <c-string 1..8> / <var: char:8>
	,OUTPUT = <var: pointer>
	,STMT = *ALL / <var: pointer>
	,PREFER = *ALL / <c-string 1..8> / <var: char:8>
	,DEFAULT = *NO / <var: pointer>
	,MESSAGE = *NO / <var: pointer>
	,PROT = *YES / *NO / <var: bit:1>
	,BUFFER = *NO / <var: pointer>
	,INVAR = *NO / <var: pointer>
	,SPIN = *NO / *YES / <var: bit:1>
	,ERRSTMT = *STEP / *NEXT / <var: bit:1>
	,CALLID = *NO / <var: pointer>
	,CCSNAME = *NO / *EXTEND / <c-string 1..8> / <var: char:8>
	,DATA_RECORD = *NO / <var: pointer>
	,STMTRC = *NO / <var: pointer>
	,OUTFORM = *NEW / *OLD / <var: bit:1>
	,MF = C / D / L / M / E

PROGRAM=<c-string 1..8> / <var: char:8>
Internal name of the program that generates the macro. This name is stored in the program definition in the syntax file (see ADD-PROGRAM). It is at least one byte and at most eight bytes long and can be transferred as a c-string constant or a string variable.

OUTPUT=<var:pointer>
Address of the standardized transfer area, which must begin on a word boundary. The transfer area must be generated with the CMDTA macro (or for OUTFORM=*OLD with CMDSTRUC, see chapter “Appendix” on page 593).

STMT =
specifies which statements are permitted as input.
Only statements with specified internal application names are permitted. The internal statement name is stored in the syntax file in the statement definition (see ADD-STMT). It is at least one and not more than eight bytes long. The SDF default statements are always permitted, regardless of the specification made here.

***ALL**
All statements are permitted.

<var:pointer>
Address of the list of the permitted statements. This list can have been generated with the CMDALLW macro.
The list must be aligned on a halfword boundary and have the following format:

2 bytes:	Number of internal names in the list (n)
8 bytes:	First internal statement name
	. . .
8 bytes:	nth internal statement name

PREFER =
Relevant only for guided dialog; specifies whether a particular statement is expected as the next input.

***NO**
No particular statement is expected. SDF asks the user via the statement menu which statement is to be entered.

<c-string 1..8>
Internal name of the statement most likely to be entered. SDF does not display a statement menu in which the user selects the statement to be entered, but instead immediately displays the form listing the operand values for the expected statement. The user may of course enter another statement instead of the one expected.

Example:
Following MODIFY-OPERAND, SDF-A expects MODIFY-VALUE as the next statement. The internal statement name is stored in the statement definition in the syntax file (see ADD-STMT). It is at least one byte and at most eight bytes long.

<var: char:8>
Address of an area, 8 bytes long, containing the internal name of the expected statement. The name must be left-justified and padded with blanks as necessary (X’40’).

DEFAULT =
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
- operand values entered or
- operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD OPERAND...,OVERWRITE-POSSIBLE=YES,... and ADD-VALUE..., VALUE= <c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value.
In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:

In the entered MODIFY statements SDF replaces the value *UNCHANGED by the current value.

***NO**
SDF is not to replace the entered operand values by values generated dynamically by the program.

<var: pointer>
Address of a list aligned on a word boundary which contains the addresses of conversion descriptions for statements. The conversion descriptions for these statements are a formatted transfer area of the type "structure" (see section "Format of the standardized transfer area" on page 365ff). Only one conversion description can be specified per statement. A conversion description contains, among other things, the internal statement name and information as to which of the operand values entered are to be changed and what values they are to be changed to. The list of addresses of conversion descriptions has the following format:

2 bytes:	Number of conversion descriptions in the list (n)
2 bytes:	(Reserved)
4 bytes:	Address of the first conversion description
	. . .
4 bytes:	Address of the nth conversion description

The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

If the operands to be given default values are in a structure introduced by a value defined with LIST-ALLOWED=*YES (see ADD-VALUE), the following situation may arise: the conversion description contains several list elements to which structures with operands to be defaulted are attached. On the other hand, the user likewise enters several list elements to which structures with operands to be defaulted are attached. SDF first tries to match the structures entered by the user to those specified in the conversion description by means of the values introducing the structures. If an unambiguous allocation cannot be made on the basis of the values introducing the structures because none of the values entered matches any of the ones in the conversion description or because the user has entered the matching value more than once, the allocation is then made on the basis of the list position of the introductory value.

MESSAGE =
specifies whether SDF is to issue a message when requesting statement input. In guided dialog this message is integrated into the statement menu.

<u>*NO</u>
SDF is not to issue a message.

<var: pointer>
Address of the message text to be issued, or a register that contains this address. The text is expected in the form of a variable-length record.

2 bytes:	Absolute length of the record (n+4)
2 bytes:	(Reserved)
n bytes:	Message text

The text may be a maximum length of 400 characters. However, only the first 280 characters are displayed on SDF-formatted screens. If the text contains screen control characters, the menu mask may be destroyed.

PROT =
Specifies whether SDF is to log input and messages to SYSOUT. If they are not written to SYSOUT, the user of the program should be informed of this in the program documentation. A log buffer can be provided.

<u>*YES</u>
SDF is to log input and messages to SYSOUT.

***NO**
SDF is not to perform any logging.
The following behavior may be expected:

Result of analysis	**PROT parameter**	
	***YES**	***NO**
No error	Input statement	– / –
Syntax error	1. Input statement 2. Syntax error message 3. Spin-off message	Spin-off message

BUFFER =
The statement log and the error messages can be written into an area provided by the user.

NO
No buffer area is provided.

<var: pointer>
Address of an area in which the log of the entered statements and the messages are written, regardless of what was specified for PROT. The area must be aligned on a word boundary and has the following format:

2 bytes:	Maximum length of logging area
2 bytes:	Length of logging area actually used
n bytes:	Log records

Each individual log set has the following format:

2 bytes:	Absolute length of the log record (m+4)
2 bytes:	(Reserved)
m bytes:	Contents of the log record

If the buffer is not empty, the first record is generally the log of the input command. Subsequent records contain messages. If no input log is available, or if the input log cannot be output, two slashes
(“//”) are written into the output area.

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the specified operands, all operands having default values and all operands currently allowed for this task. In contrast to LOGGING=*INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form **does not** mask out keywords and secret operands. More details on this topic can be found on page 415.

NO
The INVARIANT-INPUT form of the statement is not stored.

<var: pointer>
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword. The contents of the buffer are then as follows:

2 bytes:	Maximum length of the buffer
2 bytes:	Length of the record written by SDF (n+4)
2 bytes:	(Reserved)
n bytes:	INVARIANT-INPUT form of the statement, starting at the seventh byte

SPIN =
Specifies which statement, in batch mode, SDF is to read and analyze next.

<u>*NO</u>
SDF is to read and process the next statement in the statement sequence.

***YES**
SDF is to skip all statements until the next STEP statement (or, as the case may be, until the END statement) and, if there is a STEP statement, continue processing with the statement following it.

<var: bit1>

Bit variables:	bit = 0: as *NO
	bit = 1: as *YES

ERRSTMT =
defines which statement terminates the spin-off mechanism if SDF senses a syntax error for the read statement.

<u>*STEP</u>
SDF initiates spin-off until STEP or END is recognized.
The return code is X’1C’, X’34’,...

***NEXT**
SDF does not initiate spin-off. The next statement is read on the next CMDRST call. The return code is then X’50’.

<var: bit1>

Bit variables:	bit = 0: as *STEP
	bit = 1: as *NEXT

CALLID =
This function applies to the OPNCALL and CLSCALL macros.
CALLID specifies the program context (=syntax file hierarchy opened by an OPNCALL macro) in which the statement must be read and analyzed. The name of the syntax file hierarchy (CALLID) must have the 4-byte value returned by SDF to the field which was designated by the CALLID operand in the OPNCALL macro.

***NO**
The current syntax file hierarchy (context) of the task is used for analyzing the statement. This can, for example, be the syntax file hierarchy opened for the task at LOGON.

<var: pointer>
Address of the call check field or register containing this address. The area must be aligned on a word boundary.

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the "XHCS" manual [11]). This parameter has no effect on message output.

***NO**
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

<c-string 1..8> / <var: char:8>
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long and can be passed as a c-string constant or as a string variable.

DATA_RECORD=
determines whether SDF stores data which was entered instead of statements. If this is the case, SDF returns a special return code (see "Return information and error flags" below). This parameter may only be used for migration from RDATA to CMDRST (not for programs which already contain RDSTMT or CMDRST calls). In this way a program which only offers the SDF interface in the dialog can be called compatibly in procedures and batch jobs. The SDF statements of the program must be preceded by "//" in the procedures and batch jobs.

***NO**
No data storage. The result of the CMDRST call is dependent on the input:

Input from:	Type of input	Output by CMDRST
Terminal	//<stmt>	in OUTPUT parameter
Terminal	<data>	not possible
Procedure/Batch	//<stmt>	in OUTPUT parameter
Procedure/Batch	<data>	in OUTPUT parameter 1)
S procedure	//<stmt>	in OUTPUT parameter
S procedure	<data>	Error 2)

1 Data treated as statement

2 '//' is mandatory before statements in S procedures

<var: pointer>
Address of an area aligned on a word boundary in which SDF stores the data which was read instead of statements. The area has the following layout:

2 bytes: maximum length of the area
2 bytes: Output length actually used (n+4)
2 bytes: (Reserved)
n bytes: Data records

The result of the CMDRST call is independent of the input:

Input from:	**Type of input**	**Output by CMDRST**
Terminal	//<stmt>	in OUTPUT parameter
Terminal	<data>	not possible
Procedure/Batch	//<stmt>	in OUTPUT parameter
Procedure/Batch	<data>	in DATA_RECORD parameter
S procedure	//<stmt>	in OUTPUT parameter
S procedure	<data>	in DATA_RECORD parameter

STMTRC =
Specifies whether the statement supplies a return code. This return code can be evaluated with SDF-P functions in S procedure like a command return code.

***NO**
The statement does not generate a return code. SDF transfers some default return codes according to the specifications made for ERRSTMT and SPIN:

(SC2)	**SC1**	**Maincode**	**Meaning**
0	0	CMD0001	No errors
0	1	CMD0230	SDF is in spin-off
1	0	CMD0232	//STEP was recognized during spin-off, control returned to program

<var: pointer>
Address of a structure which contains the return code of the statement. SDF checks the following return code requirements to ensure that the ones returned to the user are meaningful. If these are not adhered to, SDF generates its own return codes.

1. In the case of an error in the read statement, SDF should first be called as CMDRST with SPIN=*YES and a statement return code with SC1 not equal to zero.
2. In the case of no errors, SDF should first be called with SPIN=*NO and a return code with SC1 equal to zero.

3. In the case of semantic errors, CMDRST should first be called with SPIN=*YES and a return code with SC1 not equal to null. If no return codes are generated or if the return code is positive, SDF uses the return code CMD0230.
4. If CMDRST is called with ERRSTMT=*NEXT and the macro return code X'50' is returned, the next CMDRST can be called without spin-off and a return code with SC1 equal to null. The program should not start a spin-off since it has already suppressed a spin-off initiated by SDF.

OUTFORM =
Specifies which format of the standardized transfer area is output by SDF. If default values are input (see the DEFAULT parameter) SDF identifies the format of the transfer area itself.

<u>***NEW**</u>
The standardized transfer area has the new format (see page 365ff).

***OLD**
The transfer area has the format used up to SDF V4.0 (see chapter “Appendix” on page 593ff).

<var: bit:1>

Bit variable:	bit = 0: same as *NEW
	bit = 1: same as *OLD

Description of the parameters MF, PARAM, MACID and PREFIX: see the “Executive Macros” manual [8].

Return information and error flags

The format of the transfer area is described on page 365ff. The transfer area format used up to SDF V4.0 can be found in chapter “Appendix” on page 593ff.

In the INVARIANT-INPUT form, the statement is stored with all specified operands, all operands having default values, and all the operand values currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement that is available to a user who has certain privileges and who is working in the selected dialog mode. Keywords and secret operands are not masked out. The INVARIANT-INPUT form is not the same as the logging format LOGGING=*INVARIANT-FORM (see MODIFY-SDF-OPTIONS).

The INVARIANT-INPUT form provides a portable format for SDF inputs. Since only standard names are used, this form can be transmitted without problems to a remote partner even if other external names or other default values are used at that location.

The INVARIANT-INPUT form can, however, be rejected by the remote partner if certain commands, operands, etc. have been removed from there (see the REMOVE statement) or are not allowed because of inappropriate privileges. Default values which are not allowed in the local environment are not transmitted to the remote partner, since the default values that apply in the environment of the remote system are inserted on executing the command or statement.

The return code is passed in the standard header of the parameter list.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	Normal termination
00	20	0004	Unrecoverable system error
	01	0008	Parameter error:
00			– wrong parameter list
01			– OUTPUT
02			– STMT list
03			– DEFAULT
04			– MESSAGE
05			– PROT
06			– INVAR
07			– CALLID
08			– DATA_RECORD
10			– STMTRC
00	40	000C	Transfer area too small
00	40	0010	End of input recognized
00	40	0014	Error in statement, command was recognized

continued ➠

(SC2)	SC1	Maincode	Meaning
00	40	0018	Statement is correct but the default values provided by the system are errored
00	40	001C	Error in statement, //STEP was recognized
00	40	0028	Not enough space in buffer, log truncated
00	40	002C	END statements read
00	40	0034	Error in statement, the next statement to be processed is END
00	40	003C	Program not known in syntax file
00	40	0040	Specified CALLID not found
00	40	0044	Syntax file entered in DSSM catalog not found
00	40	0050	Error in statement, spin-off was not initiated
00	40	005C	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
00	20	0064	XHCS error during statement entry
00	40	0068	Data record was read instead of a statement and stored in the DATA_RECORD buffer

CMDRST also provides the origin of the statement. In the case of a error-free parameter list, the SYSSTMT assignment is stored in the parameter list in field <prefix><macid>SYSSTMT_STATE.

The following values can occur:

Value	SYSSTMT =
X’01’	Data display terminal
X’02’	SYSCMD in non-S procedure or file
X’03’	Card reader
X’04’	Floppy disk
X’05’	SYSCMD in S procedure
X’06’	S variable

Migration from RDSTMT to CMDRST

Migration from RDSTMT to CMDRST is only necessary if the user wants to use the new functions of CMDRST. In doing so, the following points should be noted:

- In the case of a CMDRST call, the MF parameter must always be specified. The CMDDRST parameter list must be declared and initialized with MF=L, separate from the executing code.The DSECT for the parameter list must be generated with MF=D. Direct modifications (via an offset from the beginning of the parameter list) are not permitted. More information about the parameters MF and PARAM can be found in the “Executive Macros” manual [8].

Comparison of RDSTMT and CMDRST

RDSTMT	CMDRST
`<Program code> . . .`	`<Program code> . . .`

	`LA 1,LABEL`
	`USING D,1`

`<Code for manipulating the parameter list using LABEL+Offset>`
`RDSTMT MF=(E,(LABEL))`
	`<Code for manipulating the parameter list with the name from DSECT D>`
	`CMDRST MF=M,PARAM=(1),<params>`
	`CMDRST MF=E,PARAM=LABEL`

`<Program code> . . .`	`<Program code> . . .`
`<Data>`	`<Data>`

`LABEL RDSTMT MF=L,<params>`	`LABEL CMDRST MF=L,<params>`
	`<dsect>`
	`D     CMDRST MF=D`

- The macro return code is transferred in the standard header of the parameter list. The maincode of CMDRST is equivalent to the values which were transferred in RDSTMT in the right-most byte in register 15. The generation of equates for return codes with the CMDANALY macro is no longer necessary since these are automatically generated with CMDRST MF=D. The following table shows a comparison of the old (RDSTMT) and new (CMDRST) field names:

RDSTMT	CMDRST
&P.NOERR	&PREFIX.MDRSUCCESSFUL
&P.SYERR	&PREFIX.MDRSYSTEM_ERROR
&P.PAERR	&PREFIX.MDRPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDRAREA_TOO_SMALL
&P.EOF	&PREFIX.MDREOF
&P.SCMD	&PREFIX.MDRSTMTERROR_CMD
&P.DFLT	&PREFIX.MDRWRONG_DEFAULTS
&P.STEP	&PREFIX.MDRSTMTERROR_STEP
&P.PTRC	&PREFIX.MDRPROTOCOL_TRUNCATION
&P.END	&PREFIX.MDREND_STMT
&P.EERR	&PREFIX.MDRSTMTERROR_END
&P.NOPRG	&PREFIX.MDRPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDRINVALID_CALLID
&P.NOFND	&PREFIX.MDRSYNTAX_FILE_NOT_FOUND
&P.NEXT	&PREFIX.MDRSTMTERROR_NEXT
&P.ITRC	&PREFIX.MDRINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDRXHCS_ERROR

- The syntax for the parameters of CMDRST only differs from RDSTMT in the following points:
  - keywords must be preceded by an asterisk (e.g. *YES)
  - character strings must be enclosed in single quotes (e.g. 'test')
  - the parameter value *ADDR is no longer permitted; only identifiers can be specified.

Examples of different entries

RDSTMT	**CMDRST**
`PROGRAM = name`	`PROGRAM = 'name'`
`OUTPUT= addr/(r)`	`OUTPUT = identifier` [2]
`STMT = *ALL/(name,...)/ *ADDR(addr/(r))` [1]	`STMT = *ALL/identifier`
`PREFER = *NO/name/ *ADDR(addr/(r))`	`PREFER = *NO/'name'/identifier`
`DEFAULT = *NO/(addr,...)`	`DEFAULT = *NO/identifier`
`MESSAGE = *NO/addr/(r)`	`MESSAGE = *NO/identifier`
`PROT = YES/NO`	`PROT = *YES/*NO`
`BUFFER = *NO/addr|(r)`	`BUFFER = *NO/identifier`
`INVAR = *NO/addr|(r)`	`INVAR = *NO/identifier`
`SPIN = NO/YES`	`SPIN = *YES/*NO`
`ERRSTMT = STEP/NEXT`	`ERRSTMT = *STEP/*NEXT`
`CALLID = *NO/addr/(r)`	`CALLID = *NO/id`
`CCSNAME = *NO/*EXTEND/name`	`CCSNAME = *NO/*EXTEND/'name'/ identifier`
`MF = L`	`MF = L`
`MF = (E,(1))/(E,param)`	`MF = E,PARAM=identifier`
`MF not specified (standard form)`	`(not supported)`

[1] List entries are no longer possible for STMT and DEFAULT

[2] identifier can be a label in the program (address) or, with MF=M, a register

- The format of the standardized transfer area was modified at the same time as CMDRST was introduced. Considerably more information can be stored in the new transfer area and the alignment of the data is guaranteed. It is therefore recommended that only transfer areas in the new format are specified for OUPUT and DEFAULT. For reasons of compatibility, SDF continues to support the old format of the transfer area (see the OUTFORM parameter). If the program passes its own default values to SDF, SDF identifies the format of the transfer area itself.
  - In the case of OUTFORM=*OLD, the program section for analyzing the transfer area must not be modified. However, the transfer area must still be generated with CMDSTRUC (see chapter “Appendix” on page 593ff).

– In the case of OUTFORM=*NEW, it is sometimes sufficient to substitute CMDTA for CMDSTRUC and to recompile the program. The fields in the DSECTs which are generated by CMDSTRUC are to some extent also contained in the DSECTs generated by CMDTA. A list showing the old and new DSECTs can be found in the description of the CMDTA macro ("Migration from CMDSTRUC to CMDTA" on page 430).

CMDSEL
Create selection mask for guided dialog

The CMDSEL macro creates a selection mask similar to the SDF forms, in which the user can select one or more items in the guided dialog.

Figure 13 shows the main layout of a selection mask created with CMDSEL.

```
SELECTION BOX TITLE
--------------------------------------------------------------------------------
     ITEMS TITLE
 (.) ITEM1(1)      ITEM2(1)      ITEM3(1)      ITEM4(1)      ITEM5(1)
 ...
 ...
 ...
 ...
 ...
 ...
 (.) ITEM1(i)      ITEM2(i)      ITEM3(i)      ITEM4(i)      ITEM5(i)
 ...
 ...
 ...
 ...
 ...
 ...
 ...
 ...
 (.) ITEM1(n)      ITEM2(n)      ITEM3(n)      ITEM4(n)      ITEM5(n)
--------------------------------------------------------------------------------
NEXT = *EXECUTE
       *EXECUTE or *NONE or *CANCEL
```

Figure 13: Layout of a selection mask

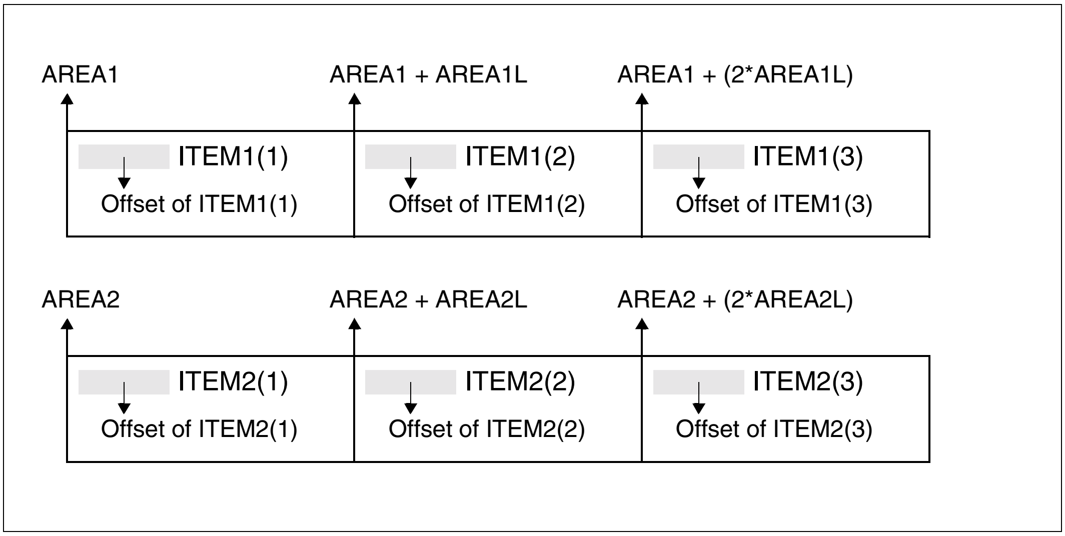
A maximum of 24 lines can be output on BS2000 screens. The number of lines needed for the selection masks depends on whether a title is output and how many NEXT lines there are. If there are more than 20 lines in the selection mask, the display is spread over the appropriate number of screen pages. The NEXT line then also contains + (scroll forward) and - (scroll backward).

A line on the screen can contain a maximum of 5 columns. The contents of a column in a line must be available in a data area, the initial address and length of which are passed in the program. The space needed for all the lines in a column is calculated from the number of lines multiplied by the length of the data area of a column.

An item to be output with a certain offset relative to the beginning of the data area and with a specific length is found in each data area of a column. Unless otherwise specified, the item has no offset relative to the beginning of the data area.

Example

The following data structure occurs in a selection box with 2 columns and 3 lines:



Operation	Operands
CMDSEL	SELECT = *SINGLE / *MULTIPLE / <var: enum-of SELECTION_S:1>
	,LINENBR = <integer 1..65536> / <var: int:4>
	,OUTPUT = <var: pointer>
	,TITLE@ = NULL / <var: pointer>
	,TITLEL = <integer 1..75> / <var: int:4>
	,TITEMS = NULL / <var: pointer>
	,TITEMSL = 0 / <integer 1..65535> / <var: int:4>
	,AREA1 = NULL / <var: pointer>
	,AREA1L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF1L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM1L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA2 = NULL / <var: pointer>
	,AREA2L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF2L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM2L = *AREA_LENGTH / <integer 1..75> / <var: int:1>

Operation	Operands
	,AREA3 = NULL / <var: pointer>
	,AREA3L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF3L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM3L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA4 = NULL / <var: pointer>
	,AREA4L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF4L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM4L = *AREA_LENGTH / <integer 1..75> / <var: int:1>
	,AREA5 = NULL / <var: pointer>
	,AREA5L = 0 / <integer 1..65535> / <var: int:2>
	,ITOFF5L = *AREA_START / <integer 1..65535> / <var: int:2>
	,ITEM5L = *AREA_LENGTH / <integer 1..75> / <var: int:1>

SELECT =
Specifies whether single or multiple selections are permitted in the mask.

***SINGLE**
Only one item can be selected in the mask.

***MULTIPLE**
Two or more items can be selected in the mask.

<var: enum-of SELECTION_S:1>
Variable that can assume the following values:
0: as *SINGLE
1: as *MULTIPLE

LINENBR = <integer 1..65536> / <var: int:4>
Number of lines that can be displayed in the selection mask.

OUTPUT = <var: pointer>
Address of an area in which the result of the selection can be entered. One byte must be reserved for each of the selectable items, i.e. the area must have the byte length specified in LINENBR and be initialized with zero or blank. If the user selects the item(i), byte(i) receives a value other than zero/blank. If a byte is assigned a value other than zero or blank before the user selection, the corresponding item is the default value in the selection mask.

TITLE@ =
Title of the selection box which is output in the first line of the screen.

NULL
No title in the selection box.

<var: pointer>
Address of the character string with the title.

TITLEL = <integer 1..75> / <var: int:4>
Length of the character string which is output as the title. This parameter is only evaluated if an address other than 0 is specified for TITLE@.

TITEMS = NULL / <var: pointer>
Address of an item title which is displayed in the sixth position on the third screen line (with 3270 terminals: 8th position).

TITEMSL = 0 / <integer 1..65535> / <var: int:4>
Length of the item title (maximum 73 characters).

AREA*x* =
with *x*=(1..5):
Data area in which the contents of a line are stored in a column, i.e. there must be precisely one of these data areas in each column.

NULL
No item is output in a specific line in column *x*.

<var: pointer>
Address of the data area in which the item to be output is stored.

AREA*x*L = 0 / <integer 1..65535> / <var: int:2>
with *x*=(1..5):
Length of a data area (in bytes) in column *x*.
The total length of all data areas of column *x* is calculated by multiplying the number of lines with the value of AREA*x*L.

ITOFF*x*L =
with *x*=(1..5):
Offset of the item relative to the data area specified in AREA*x*.

***AREA_START**
No offset.

<integer 1..65535> / <var: int:2>
Offset in bytes.

ITEM*x*L =
with *x*=(1..5): Length of the item (maximum 75 characters).

***AREA_LENGTH**
The item fills the entire data area AREA*x*.

<integer 1..75> / <var: int:1>
Length of the item (in bytes).

Return information and error flags

The return code is passed in the standard header of the parameter list.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	Normal termination
00	20	0004	Unrecoverable system error
	01	0008	Parameter error:
00			– wrong parameter list
01			– LINENBR
02			– SELECT
03			– OUTPUT
04			– TITLE@
05			– TITLEL
06			– Item1
07			– Item2
08			– Item3
09			– Item4
0A			– Item5
00	40	000C	User has not made a selection

CMDSTA
List information on activated syntax files

The CMDSTA macro causes the program to be provided with information on

– the activated syntax files and

– the specifications in effect for command/statement input and processing.

Operation	Operands
CMDSTA	OUTAREA = addr / (r) ,CALLID = <u>*NO</u> / addr / (r) ,FORM = <u>SHORT</u> / LONG / USER [,MF = { L / (E,(1)) / (E,opaddr) }]

OUTAREA=addr / (r)
Address of a transfer area, or register containing this address. The information is written to this area. The area may be generated with the CMDMEM macro. This field must be aligned on a halfword boundary.

i The size of the transfer area has been increased as of SDF V2.0A. See page 401 for further details.

CALLID =
Refers to a context (= syntax file hierarchy) which was opened by an OPNCALL macro. The name of the syntax file hierarchy (CALLID) must have the 4-byte value returned by SDF to the field which was designated by the CALLID operand in the Open Context macro. This function applies to the OPNCALL and CLSCALL macros and is reserved for BS2000 development.

<u>*NO</u>
The currently activated syntax file hierarchy is used.

addr / (r)
Address of a 4-byte field or register containing this address. The caller transfers the CALLID to the context (= syntax file hierarchy) to be used. This field must be aligned on a word boundary.

FORM =
The form (short or long or additionally with all user syntax files) in which the output information is to be written to the transfer area OUTAREA.

<u>SHORT</u>
Output consists only of the activated basic system syntax files (without the activated subsystem syntax files), the activated group syntax file, the most recently activated user syntax file and the current options for entering and processing commands and statements. A short-form output consists of 530 bytes.

LONG
as SHORT; but in addition all information relevant to the subsystem syntax files is output. In this case, the first two bytes of the transfer area must define the length of the transfer area. The transfer area for long-form output must be larger than 530 bytes. The amount of subsystem information in the output depends on the space available in the transfer area. If the subsystem information overflows the transfer area, the corresponding return code is returned.

USER
as LONG; but in addition information relevant to all activated user syntax files is output.

MF =
Defines special requirements for macro expansion (see the manual “Executive Macros” [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This either appears in register 1 or is specified directly.

Return information and error flags

Register 15 contains a return code in the right-most byte. EQUATE statements for this can be generated by means of the CMDANALY macro.

X’00’	Normal termination
X’04’	Unrecoverable error
X’08’	Operand error in the macro
X’0C’	Insufficient space in transfer area
X’38’	SDF not available
X’4C’	The program is not executable above the 16-Mbyte boundary, since SDF is not loaded. Please notify system administration.

The transfer area has the following structure:

Byte	Contents	
0	Bit 0 = 0	UTILITY-INTERFACE = NEW-MODE
	Bit 0 = 1	UTILITY-INTERFACE = OLD-MODE
	Bit 1 = 0	PROCEDURE-DIALOGUE = NO
	Bit 1 = 1	PROCEDURE-DIALOGUE = YES
	Bit 2 = 0	CONTINUATION = NEW-MODE
	Bit 2 = 1	CONTINUATION = OLD-MODE
	Bit 3 = 0	CMD-STATISTICS = NO
	Bit 3 = 1	CMD-STATISTICS = YES
	Bit 4 = 0	MENU-LOGGING = NO
	Bit 4 = 1	MENU-LOGGING = YES
	Bit 5 = 0	MODE = EXECUTION
	Bit 5 = 1	MODE = TEST
	Bit 6 = 0	No spin-off for CMDRST (read statement)
	Bit 6 = 1	Spin-off for CMDRST
	Bit 7 = 0	INPUT-HISTORY = OFF
	Bit 7 = 1	INPUT-HISTORY = ON
1	Scope of user guidance (see SET-GLOBALS ...,GUIDANCE=... and MODIFY-SDF-OPTIONS ...,GUIDANCE=...)	
	X'04'	NO
	X'05'	EXPERT
	X'06'	MAXIMUM
	X'07'	MINIMUM
	X'08'	MEDIUM
2	Type of input logging (see SET-GLOBALS ...,LOGGING=... and MODIFY-SDF-OPTIONS ...,LOGGING=...)	
	X'04'	INPUT-FORM
	X'05'	ACCEPTED-FORM
	X'06'	INVARIANT-FORM
3 to 56	Name of the activated basic system syntax file	

Byte	Contents	
57 to 68	Version number of the activated basic system syntax file (see SET-GLOBALS VERSION= ...)	
69 to 122	Name of the group syntax file activated for the task	
123 to 134	Version number of the group syntax file activated for the task (see SET-GLOBALS VERSION= ...)	
135 to 188	Name of the user syntax file last activated	
189 to 200	Version number of the user syntax file last activated (see SET-GLOBALS VERSION= ...)	
201 to 230	Name of the program which the command processor uses for analysis of the entered statements in test mode	
231 to 234	Current SDF version, format is nn.n	
235	Function key assignment (see SET-GLOBALS...,FUNCTION-KEYS= and MODIFY-SDF-OPTIONS..., FUNCTION-KEYS=....)	
	X'04' X'05'	OLD-MODE STYLE-GUIDE-MODE
236 to 237	Number of inputs to be stored (see SET-GLOBALS..., NUMBER-OF-INPUTS=... and MODIFY-SDF-OPTIONS... INPUT-HISTORY=*ON(NUMBER-OF-INPUTS=...))	
238	Security-relevant settings, see MODIFY-SDF-OPTIONS... ,MODE=*TEST(...),...,INPUT-HISTORY=*ON(...)	

	bit 0=0
	bit 0=1
	bit 1=0
	bit 1=1
		CHECK-PRIVILEGES = *NO
		CHECK-PRIVILEGES = *YES
		PASSWORD-PROTECTION = *NO
		PASSWORD-PROTECTION = *YES

239 to 243	Reserved	
244 to 245	Number of activated user syntax files	
238 to 531	Reserved for future extensions	

i Bit 6 in byte 0 (spin-off for CMDRST) cannot be used by a user program since CMDRST always executes a return after a spin-off. For this reason, this flag is always reset for user programs. This feature is reserved for system programming.

If aliases are substituted when MODIFY-SDF-OPTIONS is executed, the syntax file names are stored in the CMDSTA transfer area with <cat-id> and <user-id>.

For FORM=LONG, the transfer area is extended as follows:

Byte	Contents
239	Reserved
240 to 243	Address of the list of all the user syntax files activated for the task
244 to 245	Number of activated user syntax files
246 to 531	Reserved for future extensions
532 to 533	Number of activated subsystem syntax files
534 to 587	Name of the first activated subsystem syntax file
588 to 599	Version number of the first activated subsystem syntax file
600 to 653	Name of the second activated subsystem syntax file
654 to 665	Version number of the second activated subsystem syntax file
. . .	

The transfer area has been extended as follows for FORM=USER:

Byte	Contents
(following the information on subsystem syntax files)	Name of the first activated user syntax file (54 bytes)
	Version number of the first activated user syntax file (2 bytes)
	Name of the second activated user syntax file (54 bytes)
	Version number of the second activated user syntax file (2 bytes)
. . .	

CMDTA
Generate transfer area for analyzed statement

The CMDTA macro generates a DSECT for the standardized transfer area. This defines and initializes the data area. The standardized transfer area is needed for the following three purposes:

1. SDF passes an analyzed statement to the program (see CMDCST, CMDRST and CMDTST)
2. The program returns a semantically incorrect statement to SDF (see CMDCST)
3. The program passes values to SDF that are to replace specified operand values (see CMDRST and CMDTST).

The standardized transfer area is described in detail on page 365ff.

Operation	Operands
CMDTA	MAXLEN = <integer 0..2147483647> / <var: int 0..2147483647>
	,MF = C / D / L / M

MAXLEN = <integer 0..2147483647> / <var: int:8>
Specifies the length of the standardized transfer area (in bytes).

Description of the parameters MF, MACID and PREFIX see the “Executive Macros“ manual [8].

Migration from CMDSTRUC to CMDTA

In the new transfer area all the data is correctly aligned (addresses on word boundary, halfwords on halfword boundary, etc.). As a result, ICM instructions can be replaced by L and LH instructions.
The length field of the new transfer area occupies 4 bytes (after the standard header). In the old transfer area this was 2 bytes right at the start of the transfer area.

On the next pages you will find a comparison of the DSECTs generated by CMDSTRUC and CMDTA. The names changed when CMDTA is used are shown in **bold** print.

```
*-     ****************************
*-     * STRUCTURED DESCRIPTION    *
*-     ****************************
*-     CMDSTRUC                       CMDTA

CMDSDES  DSECT                        CMDSDES  DSECT
CMDML    DC    XL2'0'                 CMDFHDR  FHDR  MF=(C,CMD),EQUATES=NO
CMDINTN  DC    CL8' '                 CMDML          DS    F
CMDVER   DC    XL4'0'                 CMDINTN        DS    CL8
CMDLAB   DC    AL4(0)                 CMDLAB         DS    A
CMDNRMO  DC    XL2'0'                 CMDVER         DS    CL3
CMDMAINO EQU   *                      CMDUNU1        DS    XL1
CMDSDESL EQU   *-CMDSDES              CMDUNU2        DS    XL8
                                      CMDNRMO        DS    H
                                      CMDUNU3        DS    XL2
                                      CMDMAIN        EQU   *
                                      CMDSDEL        EQU   *-CMDFHDR

*-       ***************************
*-       * OPERAND DESCRIPTOR      *
*-       ***************************

CMDODES  DSECT                        CMDODES        DSECT
CMDGSTAT DC    X'00'                  CMDGSTA        DS    AL1
CMDOCC   EQU   X'80'                  CMDOCC         EQU   X'80'
CMDUCH   EQU   X'40'                  CMDUCH         EQU   X'40'
CMDERR   EQU   X'20'                  CMDERR         EQU   X'20'
CMDRDEF  EQU   X'10'                  CMDRDEF        EQU   X'10'
*                                     CMDOTYP        DS    FL1
CMDOTYPE DC    X'00'                  CMDGATT        DS    AL1
*                                     CMDWILD        EQU   X'80'
CMDODESL EQU   *-CMDODES              CMDCWIL        EQU   X'40'
                                      CMDSATT        DS    0XL1
                                      CMDFNAT        DS    AL1
                                      CMDFNCA        EQU   X'80'
                                      CMDFNUS        EQU   X'40'
                                      CMDFNGE        EQU   X'20'
                                      CMDFNVE        EQU   X'10'
                                      CMDFNTP        EQU   X'08'
                                               ORG   CMDSATT
                                      CMDNAAT        DS    AL1
                                      CMDNAUN        EQU   X'80'
                                               ORG   CMDSATT
                                      CMDCNAT        DS    AL1
                                      CMDCNUN        EQU   X'80'
                                      CMDCNCA        EQU   X'40'
                                               ORG   CMDSATT
                                      CMDTXAT        DS    AL1
```

```
                                  CMDTXSE        EQU   X'80'
                                           ORG   CMDSATT
                                  CMDXTAT        DS    AL1
                                  CMDXTOD        EQU   X'80'
                                           ORG   CMDSATT
                                  CMDPXAT        DS    AL1
                                  CMDPXAB        EQU   X'80'
                                  CMDPXPO        EQU   X'40'
                                  CMDPXQU        EQU   X'20'
                                           ORG   CMDSATT
                                  CMDSTAT        DS    AL1
                                  CMDSTQU        EQU   X'80'
                                           ORG   CMDSATT
                                  CMDPVAT        DS    AL1
                                  CMDPVCO        EQU   X'80'
                                  CMDPVUI        EQU   X'40'
                                           ORG   CMDSATT+1
                                  CMDODEL        EQU   *-CMDODES

*-        **************************
*-        * OPERAND HEADER         *
*-        **************************
*-
CMDHEAD  DSECT                    CMDHEAD        DSECT
CMDDES   DC    XL2'0'                            DS    0A
CMDOPTR  DC    AL4(0)             CMDDES         DS    XL4
CMDHEADL EQU   *-CMDHEAD          CMDOPTR        DS    A
                                  CMDHEAL        EQU   *-CMDHEAD

*-        **************************
*-        * STRUCTURE OPERAND      *
*-        **************************
*-
CMDSOP   DSECT                    CMDSOP   DSECT
CMDNREL  DC    XL2'0'                      DS    0A
CMDSTRT  DC    XL6'0'             CMDNREL  DS    H
CMDSTEL  EQU   *                  CMDUNU4  DS    XL2
CMDSOPL  EQU   *-CMDSOP           CMDSTRT  DS    XL8
                                  CMDSTEL  EQU   *
                                  CMDSOPL  EQU   *-CMDSOP
```

```
*-        ***************************
*-        * OPERAND VALUE           *
*-        ***************************
*-
CMDOVAL  DSECT                       CMDOVAL  DSECT
CMDLVAL  DC    XL2'0'                         DS    0A
CMDAVAL  EQU   *                     CMDLVAL  DS    H
CMDTIME  EQU   CMDAVAL               CMDUNU5  DS    XL2
CMDHOUR  DC    XL2'0'                CMDAVAL  EQU   *
CMDMINU  DC    X'0'                  CMDTIME  EQU   CMDAVAL
CMDSEC   DC    X'0'                  CMDHOUR  DS    H
         ORG   CMDAVAL               CMDMINU  DS    X
CMDIVAL  DC    XL4'0'                CMDSEC   DS    X
CMDOVALL EQU   *-CMDOVAL                      ORG   CMDAVAL
                                     CMDIVAL  DS    F
                                     CMDOVLL  EQU   *-CMDOVAL

*-        ***************************
*-        * LIST ELEMENT            *
*-        ***************************
*-
CMDLE    DSECT                       CMDLE    DSECT
CMDETYPE DC    XL6'0'                         DS    0A
CMDORL   EQU   *                     CMDETYP  DS    XL8
CMDNEL   DC    AL4(0)                CMDORL   EQU   *
CMDELOP  EQU   *                     CMDNEL   DS    AL4
CMDELVAL EQU   *                     CMDELOP  EQU   *
CMDLEL   EQU   *-CMDLE               CMDELVA  EQU   *
                                     CMDLEL   EQU   *-CMDLE

*-        *****************************
*-        * EQUATES FOR OPERAND TYPES *
*-        *****************************
*-
CMDC#RES EQU   1                     CMDCRES  EQU   1
CMDINT   EQU   2                     CMDINT   EQU   2
CMDX#STR EQU   4                     CMDXSTR  EQU   4
CMDC#STR EQU   5                     CMDCSTR  EQU   5
CMDNAME  EQU   6                     CMDNAME  EQU   6
CMDA#NAM EQU   7                     CMDANAM  EQU   7
CMDS#NAM EQU   8                     CMDSNAM  EQU   8
CMDLABEL EQU   9                     CMDLABE  EQU   9
CMDSTAR  EQU   10                    CMDSTAR  EQU   10
CMDF#FIL EQU   11                    CMDFFIL  EQU   11
CMDP#FIL EQU   12                    CMDPFIL  EQU   12
CMDTIM   EQU   13                    CMDTIM   EQU   13
CMDDATE  EQU   14                    CMDDATE  EQU   14
CMDCNAME EQU   15                    CMDCNAM  EQU   15
```

```
CMDTEXT  EQU   16                  CMDTEXT  EQU   16
CMDCATID EQU   17                  CMDCATI  EQU   17
CMDI#TXT EQU   18                  CMDITXT  EQU   18
CMDSTRUC EQU   19                  CMDSTRU  EQU   19
CMDLIST  EQU   20                  CMDLIST  EQU   20
CMDOR#LI EQU   21                  CMDORLI  EQU   21
CMDKEYW  EQU   22                  CMDKEYW  EQU   22
CMDVSN   EQU   24                  CMDVSN   EQU   24
CMDXTEXT EQU   25                  CMDXTEX  EQU   25
CMDFIXD  EQU   26                  CMDFIXD  EQU   26
CMDDEV   EQU   27                  CMDDEV   EQU   27
CMDPVER  EQU   28                  CMDPVER  EQU   28
CMDX#PAT EQU   29                  CMDXPAT  EQU   29
CMDX#FIL EQU   35                  CMDXFIL  EQU   35
```

CMDTST
Analyze statement

The CMDTST macro causes SDF to

– analyze a program statement stored in the program itself, and

– pass the results of the analysis to the program.

Additional statements may result from the analysis of the transferred statement, due to the fact that a system exit may replace the transferred statement by several statements. Dealing with these additional statements is the responsibility of the program.

An activated syntax file must contain the definition of the program and its statements.

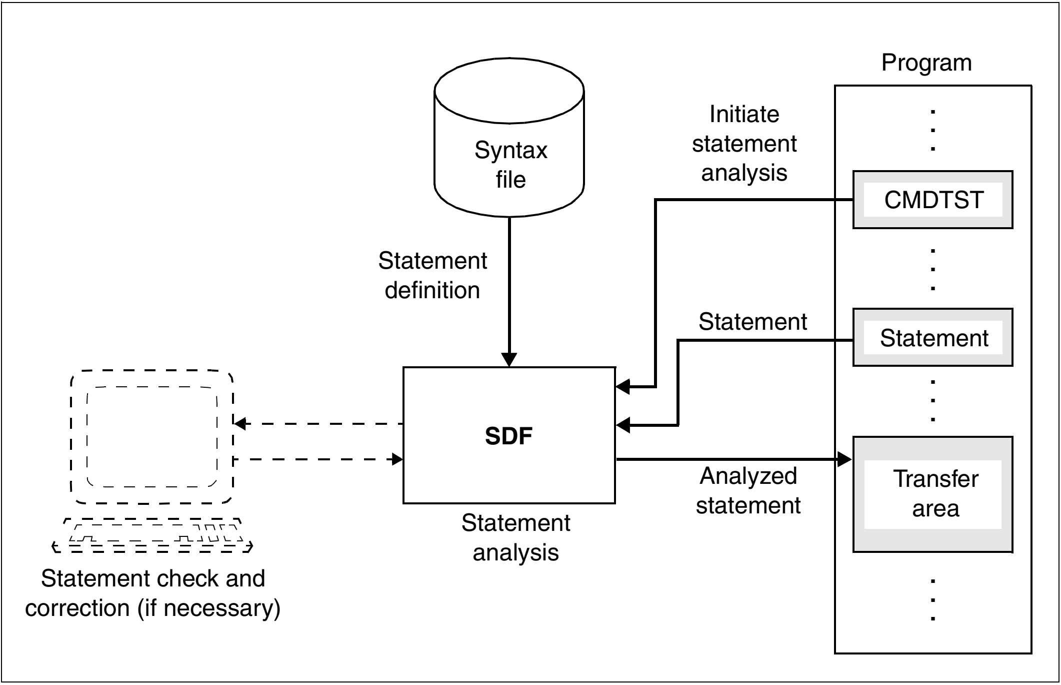


Figure 14: Effects of the CMDTST macro

Operation	Operands
CMDTST	PROGRAM = <u>*NONE</u> / <c-string 1..8> / <var: char:8>
	,EXTNAME = <u>*NONE</u> / <c-string 1..8> / <var: char:8>
	,INPUT = <var: pointer> / *NO
	,OUTPUT = <var: pointer>
	,STMT = <u>*ALL</u> / <var: pointer>
	,DIALOG = <u>*NO</u> / *YES / *ERROR / <var: enum-of DIALOG_S:1>
	,MESSAGE = <u>*NO</u> / <var: pointer>
	,PROT = <u>*YES</u> / *NO / <var: bit:1>
	,BUFFER = <u>*NO</u> / <var: pointer>
	,INVAR = <u>*NO</u>
	,DEFAULT = <u>*NO</u> / <var: pointer>
	,ERROR = <u>*NO</u> / *YES / <var: bit:1>
	,CALLID = <u>*NO</u> / <var: pointer>
	,EXECUTE = <u>*NO</u> / *YES / <var: bit:1>
	,PROCMOD = <u>*ANY</u> / *NO / *YES / <var: enum-of PROCEDURE_S:1>
	,CCSNAME = <u>*NO</u> / *EXTEND / <c-string 1..8> / <var: char:8>
	,INPUTSV = <u>*NO</u> / *YES / <var: bit:1>
	,OUTFORM = <u>*NEW</u> / *OLD / <var: bit:1>
	,DEFDEF = <u>*NO</u> / *YES / <var: bit:1>
	,MF = C / D / L / M / E

PROGRAM = <c-string 1..8> / <var: char:8>
Internal name of the program that is executed by the macro. This name is stored in the program definition[1] in the syntax file (see ADD-PROGRAM).

<u>***NONE**</u>
The internal name of the program is not specified. In this case the external name of the program must be specified in the EXTNAME operand.

<c-string 1..8> / <var: char:8>
The internal name of the program can be passed as a c-string constant or a string variable. It is at least one byte and at most eight bytes long.

[1] If the program-specific syntax file is even assigned with CMDTST, then CMDEDIT is to be specified as the program name.

EXTNAME =
External name of the program that issued the macro call. This name is stored in the program definition in the syntax file (see the ADD-PROGRAM statement, NAME operand, page 136).

***NONE**
The external name of the program is not specified. In this case the internal name of the program must be specified in the EXTNAME operand.

<c-string 1..8> / <var: char:8>
The internal name of the program can be passed as a c-string constant or a string variable. It is at least one byte and at most eight bytes long.

INPUT =
specifies which statement SDF is to analyze.

***NO**
SDF is not to analyze any statement stored in the program, but is instead to analyze an additional statement provided by a system exit.

<var:pointer>
SDF is to analyze the statement whose address is specified. The area with the specified address must be aligned on a halfword boundary. SDF expects the statement in the following format.

2 bytes:	absolute length of the record (n+4)
2 bytes:	(Reserved)
n bytes:	Message text

OUTPUT = <var:pointer>
Address of the standardized transfer area. The area must begin on a word boundary. The transfer area is generated by means of the CMDTA macro (or in OUTFORM=*OLD by means of CMDSTRUC, see chapter “Appendix” on page 593ff).

STMT =
Specifies which statements are permitted as input. Only statements with specified internal statement names are permitted. The internal statement name is stored in the syntax file in the statement definition (see ADD-STMT). It is at least one and not more that 8 bytes in length. The SDF standard statements are always permitted, regardless of the specification made here.

<u>*ALL</u>
All statements are permitted.

<var:pointer>
Address of the list of permitted statements. This list can have been generated with the CMDALLW macro. The list must be aligned on a halfword boundary and structured as follows:

2 bytes:	Number of internal names in the list (n)
8 bytes:	First internal statement name
	...
n bytes:	nth internal statement name

DIALOG =
Specifies whether SDF is to conduct a dialog when analyzing statements. This operand is relevant only when the program is executing in an interactive task.

<u>*NO</u>
SDF is not to conduct a dialog.

***YES**
SDF is to present the statement given to it by the program to the user in dialog for possible modification, provided this is compatible with the current SDF specifications for the dialog (see MODIFY-SDF-OPTIONS and SET-GLOBALS).

***ERROR**
SDF is to conduct a dialog only when it has detected a syntax error. If the statement contains a semantic error, the program can initiate a semantic error dialog by means of CMDCST.

<var:enum-of DIALOG_S:1>
Enumeration variable which can assume the following values:

0 :	same as *NO
1 :	same as *YES
2 :	same as *ERROR

MESSAGE =
Specifies whether SDF is to issue a message when presenting the statement to the user for checking and possible modification (relevant only for DIALOG ≠ NO). SDF integrates this message into the form.

<u>*NO</u>
SDF is not to issue a message.

<var:pointer>
Address of the message text to be issued, aligned to a halfword boundary. The text is expected in the form of a variable-length record.

2 bytes:	absolute length of the record (n+4)
2 bytes:	(Reserved)
n bytes:	Message text

The message text has a maximum length of 400 characters. However, only the first 280 characters are displayed on SDF-formatted screens. If the text contains screen control characters, the menu mask may be destroyed.

PROT =
specifies whether SDF is to log input and messages to SYSOUT. If they are not written to SYSOUT, the user of the program should be informed of this in the program documentation. A logging buffer is available (see BUFFER).

<u>*NO</u>
SDF is not to perform any logging.

***YES**
SDF is to log input and messages to SYSOUT.

<var: bit: 1>

Bit variable:	bit = 0: same as *NO
	bit = 1: same as *YES

BUFFER =
The log of the statement and error messages can be written to an area defined by the user.

<u>*NO</u>
No buffer is defined.

<var:pointer>
Address of an area in which the log of the specified statement and the messages are stored, irrespective of the values specified for PROT. The area must be aligned to a half-word boundary, and has the following format.

2 bytes:	Absolute length of the logging area (n+4)
2 bytes:	Actual used length of the logging area
n bytes:	Log records

Each individual log record has the following format:

2 bytes:	Absolute length of the log record (n+4)
2 bytes:	(Reserved)
m bytes:	Contents of the log record

If the buffer is not empty, the first record is generally the log of the input command. Subsequent records contain messages. If no input log is available, or if the input log cannot be output, two slashes
("//") are written to the output area. The alignment of the log records is not guaranteed.

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the defined operands, all operands having default values and all operands currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement available to a user who has certain privileges and who is working in the selected dialog mode. In contrast to LOGGING=*INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form *does not* mask out keywords and secret operands.

<u>*NO</u>
The INVARIANT-INPUT form of the statement is not stored.

<var:pointer>
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword.
The contents of the buffer are then as follows:

2 bytes:	Maximum length of the buffer
2 bytes:	Output length written by SDF (n+4)
2 bytes:	Reserved
n bytes:	INVARIANT-INPUT form of the statement, as of the 7th byte

DEFAULT =
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
- operand values entered or
- operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD OPERAND..., OVERWRITE-POSSIBLE=YES,... and ADD-VALUE...,VALUE= <c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value.
In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:
In the MODIFY statements issued to SDF-A, the value *UNCHANGED is replaced by the current value.

***<u>NO</u>**
SDF is not to replace the operand values entered by values generated dynamically by the program.

<var:pointer>
Address of a list aligned on a word boundary that contains addresses of conversion descriptions for statements. A formatted transfer area of the type “structure” is used as a conversion description (see section “Format of the standardized transfer area” on page 365ff). Only one conversion description can be specified per statement. A conversion description contains, among other things, the internal statement name and information regarding which of the operand values entered are to be changed and what values they are to be changed to. The list of addresses of conversion description is structured as follows:

2 bytes:	Number of conversion descriptions in the list (n)
2 bytes:	(Reserved)
4 bytes:	Address of the first conversion description
	...
4 bytes:	Address of the nth conversion description

The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

If the operands to be given default values are in a structure introduced by a value defined with LIST-ALLOWED=*YES (see ADD-VALUE), the following situation may arise: the conversion description contains several list elements to which structures with operands to be defaulted are attached. On the other hand, the user likewise enters several list elements to which structures with operands to be defaulted are attached. SDF first tries to match the structures entered by the user to those specified in the conversion description by means of the values introducing the structures. If an unambiguous allocation cannot be made on the basis of the values introducing the structures because none of the values entered matches any of the ones in the conversion description or because the user has entered the matching value more than once, the allocation is then made on the basis of the list position of the introductory value.

ERROR =
Specifies how the message text specified with the MESSAGE operand is to be issued.

***NO**
SDF is to issue the message text as a message.

***YES**
SDF is to issue the message text as an error message.

<var: bit:1>

Bit variable:	bit = 0: same as *NO
	bit = 1: same as *YES

CALLID =
Defines the context to be used by SDF for analyzing the command. CALLID defines the program context (=syntax file hierarchy opened by an OPNCALL macro) in which the statement must be analyzed.

***NO**
The currently active syntax file hierarchy is used.

<var. pointer>
Address of a 4-byte field which is aligned on a word boundary. The caller transfers the CALLID of the context to be used.

EXECUTE =*NO / *YES / <var: bit:1>
Specifies whether standard SDF statements are to be executed. EXECUTE is irrelevant if the CMDTST macro does not refer to a new syntax file hierarchy (CALLID=*NO), e.g. if the current syntax file hierarchy is used. In this case, the standard SDF statements are always executed by CMDTST, provided they exist in the current hierarchy. The operand belongs to the multihierarchical attribute introduced with the preceding CALLID operand. If CMDTST refers to a hierarchy opened in parallel (CALLID=<var: pointer>), the standard SDF statements are executed if EXECUTE = *YES.

<var: bit:1>

Bit variable:	bit = 0: same as *NO
	bit = 1: same as * YES

PROCMOD =
Defines the environment in which the user works. SDF performs a check, as a result of which it rejects commands which are illegal in the specified environment. The operand belongs to the multihierarchical attribute referred to by means of the CALLID operand and is only of relevance if the macro call refers to a new hierarchy opened in addition to the current one.
If no CALLID has been defined (CALLID=*NO), PROCMOD is irrelevant. In other words, the value *ANY is set automatically and reference is made to the current procedure mode.

<u>*ANY</u>
No check is made. Statements are always analyzed.

***YES**
Statements are handled as if they were read from a procedure file. They are analyzed if they have been defined in the syntax file with DIALOG-PROC-ALLOWED=*YES and if the program runs in interactive mode, or if BATCH-PROC-ALLOWED=*YES applies in batch jobs.

***NO**
Statements are handled as if they were read from a primary level, e.g. from terminal input or from a batch job. They are analyzed if they have been defined in the syntax file with DIALOG-ALLOWED=*YES and if the program runs in interactive mode, or if BATCH-ALLOWED=*YES applies in batch jobs.

<var: enum-of PROCEDURE_S:1>
Enumeration variable which can assume the following values:

0 :	same as *ANY
1 :	same as *YES
2 :	same as *NO

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the “XHCS” manual [11]). This parameter has no effect on message output.

<u>*NO</u>
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

<c-string 1..8> / <var:char:8>
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long and can be passed as a c-string constant or as a string variable.

INPUTSV =
Specifies whether a history of past inputs is to be saved in the form of a list that can be subsequently accessed again via the built-in RESTORE mechanism (see the MODIFY-SDF-OPTIONS and RESTORE-SDF-INPUT statements).

<u>*NO</u>
The inputs are not saved.

***YES**
The inputs are saved in a buffer.

<var: bit:1>

Bit variable:	bit = 0: same as *NO
	bit = 1: same as *YES

OUTFORM =
Specifies which format of the standardized transfer area is output by SDF. When default values are input, (see the DEFAULT parameter), SDF identifies the format of the transfer area itself.

<u>*NEW</u>
The standardized transfer area has the new format (see page 365ff).

***OLD**
The standardized transfer area has the format used up to SDF V4.0 (see chapter “Appendix” on page 593ff).

<var: bit:1>

Bit variable:	bit = 0: same as *NEW
	bit = 1: same as *OLD

DEFDEF =
Specifies whether statements for setting task-specific default values may be entered (see the “Introductory Guide to the SDF Dialog Interface SDF” [1]).

***NO**
Statements for setting task-specific values are not accepted.

***YES**
Statements for setting task-specific values are accepted.

<var: bit:1>
Bit variable: bit = 0: same as *NO
bit = 1: same as *YES

Description of the parameters MF, PARAM, MACID and PREFIX: see the “Executive Macros” manual [8].

Return information and error flags

The format of the transfer area is described on page 365ff. The format used for the transfer area up to SDF V4.0 can be found in chapter “Appendix” on page 593ff.

Information on the INVARIANT-INPUT form of a statement can be found under the description of the CMDRST macro on page 405.

The return code is passed in the standard header of the parameter list.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
	00	0000	Normal termination
00			– no special occurrences
01			– program supplies default values
00	20	0004	Unrecoverable system error
	01	0008	Parameter error:
00			– wrong parameter list
01			– OUTPUT
02			– STMT list
03			– DEFAULT
04			– MESSAGE
05			– PROT
06			– INVAR
07			– CALLID
09			– INPUT
00	40	000C	Transfer area too small

(SC2)	SC1	Maincode	Meaning
00	40	0018	Statement is correct but the default values provided by the system are errored
00	40	001C	Error in statement
00	40	0020	Error dialog not possible
00	40	0024	Error dialog rejected by user
00	40	0028	Not enough space in buffer, log truncated
00	40	002C	END statements read
00	40	003C	Program not known in syntax file
00	40	0040	Specified CALLID not found
00	40	0044	Syntax file entered in DSSM catalog not found
00	40	005C	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
00	20	0064	XHCS error during statement entry
00	40	006C	Statement for setting task-specific defaults was entered instead of normal statement

Indicators of additional statements (created by a system exit):

In the case of an error-free parameter list, an indicator for additionally generated statements is stored in the parameter list in the field <prefix><macid>EXIT_ACTIVE.

The following values can occur:

Value	Meaning
X'00'	There are no further statements
X'01'	There are further statements

Migration from TRSTMT to CMDTST

Migration from TRSTMT to CMDTST is only necessary if the user wishes to use the new functions offered by CMDTST. In this case, the same points must be taken into account as for CMDRST (see “Migration from RDSTMT to CMDRST” on page 416ff). There are also the following changes:

TRSTMT	CMDTST

`PROT = YES/NO/addr/(r)`	`PROT = *YES/*NO`
	`BUFFER = *NO/identifier`

`INPUTSAV = NO/YES`	`INPUTSV = *NO/*YES (renamed!)`

The macro return code is transferred in the standard header of the parameter list. The maincode of CMDTST is equivalent to the values which were transferred in TRSTMT in the right-most byte in register 15. The generation of equates for return codes by means of the

CMDANALY macro is no longer necessary, as these are automatically generated with CMDTST MF=D. The following table shows a comparison of the old (TRSTMT) and new (CMDTST) field names:

TRSTMT	**CMDTST**
&P.NOERR	&PREFIX.MDTSUCCESSFUL
&P.SYERR	&PREFIX.MDTSYSTEM_ERROR
&P.PAERR	&PREFIX.MDTPARAMETER_ERROR
&P.TRUNC	&PREFIX.MDTAREA_TOO_SMALL
&P.CERR	&PREFIX.MDTERROR_IN_STMT
&P.DFLT	&PREFIX.MDTWRONG_DEFAULTS
&P.DIMP	&PREFIX.MDTDIALOG_IMPOSSIBLE
&P.DREJ	&PREFIX.MDTDIALOG_REJECTED
&P.PTRC	&PREFIX.MDTPROTOCOL_TRUNCATION
&P.END	&PREFIX.MDTEND_STMT
&P.NOPRG	&PREFIX.MDTPROGRAM_NOT_IN_SYNTAX_FILE
&P.INCID	&PREFIX.MDTINVALID_CALLID
&P.NOFND	&PREFIX.MDTSYNTAX_FILE_NOT_FOUND
&P.ITRC	&PREFIX.MDTINVARIANT_INPUT_TRUNCATED
&P.XHCS	&PREFIX.MDTXHCS_ERROR

CMDVAL
Check value for data type

The CMDVAL macro checks whether a value matches an SDF data type description. The macro passes on the result of the check and, if requested, an SDF error message to the PROT buffer. The error messages are not output to SYSOUT. This macro can also be used to check whether an input string matches a predefined wildcard search pattern.

Operation	Operands
CMDVAL	INPUT = addr
	[, DATATYP = { <u>NOCHECK</u> / INTEGER / XSTRING / CSTRING /NAME / ALPHANAME / STRUCNAME / FILENAME / PARTFILE / TIME / DATE / COMPONAME / TEXT / CATID / KEYWORD / KEYNUMBER / VSN / XTEXT / FIXED / DEVICE /PRODVERS / POSIXPATH / POSIXFILE }]
	[,SHORTST = <u>*ANY</u> / integer]
	[,LONGEST = <u>*ANY</u> / integer]
	[,WCLOGL = <u>*NONE</u> / integer]
	[,LOWDEC = <u>0</u> / integer]
	[,HIGDEC = <u>0</u> / integer]
	[,CONST = <u>*NO</u> / addr / (addr, ...)]
	[,PATTERN = <u>*NO</u> / addr]
	[,ATTRIB = { <u>*NONE</u> ([NOCATID] [,NOUSERID] [,NOGENERATION] [,NOVERSION] [,WILDCARD] [,KEYSTAR] [,NOSEPERATORS] [,UNDERSCORE] [,NOODD] [,NOALIAS] [,VOLUMEONLY] [,NOUSERINT] [,NOCORSTATE] [,ANYCORSTATE] [,WILDCONST] [,LOWERCASE] [,TEMPFILE] [,QUOTESMAND] [,ANYUSERINT] [,STDDISK]) }]
	[,DEVCLAS = list-poss(2): DISK / TAPE]
	[,EXCDISK = *NONE / addr / (text8,...)]
	[,EXCTAPE = *NONE / addr / (text8,...)]
	[,PROT = <u>*NO</u> / addr]

Operation	**Operands**
CMDVAL (continued)	[,PREFIX = <u>C</u> / p] [,MACID = <u>MDV</u> / mac] [,MF = D / C / L / E] [,PARAM = addr]

INPUT = addr
String representing a value in variable record format (first halfword: length of the record; second halfword: fill characters used to pad the record to the specified length).
No blanks are allowed as part of the input (exception: blanks within a value of the type <c-string> or <text>).
addr must be aligned on a word boundary.

DATATYP =
Specifies the data type for which the value is to be checked (see statement ADD-VALUE TYPE=...).

<u>NOCHECK</u>
No data type check. Instead, the value is checked to verify whether it matches a wildcard search pattern. The PATTERN parameter is mandatory in conjunction with NOCHECK.

The following table shows the possible data type checks:

DATATYPE=	**Data type whose value is checked**
INTEGER	<integer>
XSTRING	<x-string>
CSTRING	<c-string>
NAME	<name>
ALPHANAME	<alphanum-name>
STRUCNAME	<structured-name>
FILENAME	<filename>
PARTFILE	<partial-filename>
TIME	<time>
DATE	<date>
COMPONAME	<composed-name>
CATID	<cat-id>
TEXT	The value is checked for the data type <text> if it is a positional operand in an operation. The value is invalid if it contains incorrectly placed separators.

DATATYPE=	**Data type whose value is checked**
KEYWORD	<keyword>
KEYNUMBER	<keyword-number>
VSN	<vsn>
XTEXT	<x-text>
FIXED	<fixed>
DEVICE	<device>
PRODVERS	<product-version>
POSIXPATH	<posix-pathname>
POSIXFILE	<posix-filename>

SHORTST =
Specifies a minimum length (if any) for the string (see ADD-VALUE TYPE=...(SHORTEST-LENGTH=...)). For the data types DATE, TIME, CATID, KEYWORD and KEYNUMBER, this parameter is irrelevant. If the data type is XSTRING, SHORTST is the number of bytes in the value specified as INPUT. If the data type is INTEGER, SHORTST is the lower limit of the value range; if the data type is FIXED, SHORTST is combined with the LOWDEC parameter. An integer complete with sign can be specified for both these data types.

***ANY**
The limits specified by SDF apply to the data type.

integer
Explicitly specifies the minimum length.

LONGEST =
Specifies a maximum length (if any) for the string (see ADD-VALUE TYPE=... (LONGEST-LENGTH=...)). For the data types DATE, TIME, CATID, KEYWORD and KEYNUMBER, this parameter is irrelevant. If the data type is XSTRING, LONGEST is the number of bytes in the value specified as INPUT. If the data type is INTEGER, LONGEST is the upper limit of the value range; if the data type is FIXED, LONGEST is combined with the HIGDEC parameter. An integer complete with sign can be specified for both these data types.

***ANY**
The limits specified by SDF apply to the data type.

integer
Explicitly specifies the maximum length.

WCLOGL =
Relevant only with ATTRIB=WILDCARD. The value specified as INPUT may include wildcards. WCLOGL specifies the maximum length of the value matching the wildcard search pattern, whereas LONGEST specifies the actual length of the input value.
This parameter has no significance for the data types POSIXPATH and POSIXFILE.

***NONE**
The limits specified by SDF apply.

integer
Explicitly specifies the maximum length.

LOWDEC = 0 / integer
Specifies the number of decimal places for SHORTST. Relevant only if the data type is FIXED.

HIGDEC = 0 / integer
Specifies the number of decimal places for LONGEST. Relevant only if the data type is FIXED.

CONST = *NO / addr / (addr,...)
The INPUT value is compared with the constants specified here. CONST is relevant only with DATATYP=NOCHECK. The constants must be stored in records of variable length. If the data type of the INPUT value is KEYWORD or KEYNUMBER, the value can be an abbreviation of one of the constants. Note, however, that lists are not supported. The number of constants is limited to 2000. The minimum and maximum lengths of the constants depend on the defaults for SDF data types.

addr
The record addresses are grouped in a field beginning with the address addr. This field must be aligned on a word boundary and its format must be as follows:

Byte	Meaning
0	Number of elements in the field (N, 2 bytes long)
2	Filler
4	Address of 1st record (aligned on word boundary)
8	Address of 2nd record (aligned on word boundary)
...	
N*4	Address of nth record (aligned on word boundary)

(addr,...)
The record addresses are listed.

PATTERN = *NO / addr
The INPUT value is compared with a wildcard search pattern.
If checking for a data type is requested, the syntax is subject to the conventions for wildcards for the data type in question (including the maximum length of the wildcard expression).
With DATATYP=NOCHECK, the INPUT value is checked to verify only whether it matches the specified wildcard search pattern (no data type check). The length of the wildcard expression is not restricted in this case. Wildcards are not valid for all data types (see statement ADD-VALUE TYPE=...). PATTERN cannot be used together with ATTRIB=(...,WILDCARD,...).
With DATATYPE=POSIXPATH or POSIXFILE, a special POSIX wildcard syntax applies instead of the BS2000 wildcard syntax. PATTERN cannot be specified for either of these data types.

PATTERN = addr
Specifies the address of a record of variable length in which the wildcard search pattern is stored. addr must be aligned on a word boundary. The value in the record length field of the V record is equal to the sum of the exact length of the search pattern and the length of the record field (e.g. the record length field for “ABC*” will have the value 4+4=8). Trailing blanks are not allowed in the pattern.

ATTRIB = (...)
Specifies a list of attributes valid for the data type. Invalid combinations are rejected and an error message issued. The attributes must be enclosed by parentheses. This applies even if the list contains only one attribute. See the ADD-VALUE statement for more detailed information.

Attributes	**Meaning and possible data type**
*NONE	The SDF defaults apply for the attributes.
NOCATID	Specification of a catalog ID is not permissible (data types <filename>, <partial-filename>)
NOUSERID	Specification of a user ID is not permissible (data types <filename>, <partial-filename>).
NOGENERATION	Specification of a generation number is not permissible (data type <filename>).
NOVERSION	Specification of a generation number is not permissible (data type <filename>).
WILDCARD	Wildcards may be used. The WILDCARD attribute must not be used together with the PATTERN parameter (data types <alphanum-name>, <composed-name>, <filename>, <name> and <partial-filename> and <structured-name>).
KEYSTAR	The input string must be preceded by an asterisk (applicable only to data types KEYWORD and KEYNUMBER).

Attributes	Meaning and possible data type
NOSEPERATORS	Separators are not allowed (data type <text>).
UNDERSCORE	The underscore character (_) is allowed (<name>, <composed-name>)
NOODD	The string must not consist of an odd number of characters (data type <x-text>).
NOALIAS	Aliases are not allowed (data type <device>)
VOLUMEONLY	Volume type is accepted (data type <device>).
NOUSERINT	Specification of the user interface is not allowed (data type <product-version>)
NOCORSTATE	Specification of the correction state is not allowed (data type <product-version>). If NOUSERINT was specified in the list, NOCORSTATE automatically applies.
ANYCORSTATE	Specification of the correction state is permitted for <product-version>. ANYCORSTATE must not be specified with NOCORSTATE and vice versa.
WILDCONST	The value can be a wildcard constructor (see ADD-VALUE TYPE=...(...,WILDCARD=*YES(TYPE=*CONSTRUCTOR=...)). (Data types <alphanum-name>, <composed-name>, <filename>, <name>, <partial-filename> and <structured-name>)
LOWERCASE	Lowercase characters are retained (data type <name>).
NOTEMPFILE	Temporary file names are not permitted (data type <filename>).
QUOTESMAND	POSIX path names and file names must be enclosed in single quotes
ANYUSERINT	The release status of the user interface can be specified (data type <product-version>). If NOUSERINT is specified at the same time, ANYUSERINT is not permitted, and vice versa.
STDDISK	Only standard disk devices may be used (data type <device>)

DEVCLAS = DISK / TAPE
Specifies the device class (data type <device>, see statement ADD-VALUE TYPE=*DEVICE(...)).

EXCDISK = *NONE / addr / (text8,...)
Specifies the excluded disk devices (data type <device>). The list may contain a maximum of 50 device names.

addr
Specifies the address of a field aligned on a word boundary and containing the device names. The first halfword in the field contains the number of elements in the field, the second halfword contains the filler. The device names following the filler are each 8 bytes long; separators are not permitted in the names. If the real device names are shorter than 8 bytes, they must be entered left-justified and padded with blanks to the full length.

EXCDISK = (text8,...)
Specifies a list of device names, each name being 8 bytes long. The rules of syntax for data types <text-without-sep> apply to the device names.

EXCTAPE = *NONE / addr / (text8,...)
Specifies the excluded tape devices. This parameter is used only in conjunction with DEVICE=TAPE (see ADD-VALUE TYPE=*DEVICE(...)). The list may contain a maximum of 50 device names.

addr
Specifies the address of a field aligned on a word boundary and containing the device names. The first halfword in the field contains the number of elements in the field, the second halfword contains the filler. The device names following the filler are each 8 bytes long; separators are not permitted in the names. If the real device names are shorter than 8 bytes, they must be entered left-justified and padded with blanks to the full length.

EXCTAPE = ((text8,...)
Specifies a list of device names, each name being 8 bytes long. The rules of syntax for data types <text-without-sep> apply to the device names.

PROT = <u>*NO</u> / addr
Record of variable length in which SDF stores the error message for the checked value. addr must be aligned on a word boundary. For details of the record, see the description of the BUFFER parameter in the CMDRST macro.

For descriptions of the parameters PREFIX, MACID, MF and PARAM, see section “Macro types” on page 379ff.

Possible calls

```
[label]  CMDVAL MF=D [,PREFIX=p][,MACID=mac]
[label]  CMDVAL MF=C [,PREFIX=p][,MACID=mac]
[label]  CMDVAL MF=L,...
[label]  CMDVAL MF=E,PARAM=addr
```

The CMDVAL macro is implemented with a standard header, which means that the form MF=S is not available.

Register usage

Register 1: Address of the parameter list.

Register 15: Return code that is additionally passed in the standard header as well.

Return information and error flags

The return code is passed in the standard header of the parameter area.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	Checked value matches data type and/or wildcard search pattern.
01	01	0008	The INPUT parameter is invalid.
02	01	0008	Invalid address (not allocated, not aligned on word boundary, ...)
03	01	0008	Invalid specification for – the combination DATATYP/ATTRIB/PATTERN or – the combination DEVCLAS/EXCDISK/EXCTAPE
04	01	0008	Invalid specifications for value ranges: – upper limit < lower limit, – SDF-A limits violated, – values incorrect, – number of decimal places not specified, – limits invalid
05	01	0008	The PATTERN parameter is invalid (length=0, syntax error)
06	01	0008	The CONST parameter is invalid
xx	40	001C	The checked value does not match data type and/ or wildcard search pattern. An SDF error message has been written into the PROT buffer. The input value was:
01	40	001C	– not of the specified data type or was without the specified attributes
02	40	001C	– not one of the specified devices
03	40	001C	– out of the range specified by the user or SDF
04	40	001C	– not one of the specified constants (value or keyword)
05	40	001C	– not a match for the wildcard search pattern.

Notes

– If the PROT buffer is too small, the SDF error messages are truncated. It is up to the caller to recognize this situation and react accordingly.

– The table below should be used for comparing file names (with file generation numbers) with a wildcard search pattern. The table contains the valid combinations for which a match is currently possible. For details on file generation groups (fggs) see the “Introductory Guide to DMS” [7].

INPUT string	**PATTERN string**	**Match possible ?**
without fgg	without fgg with fgg	yes no
with absolute fgg (abs_fgg (*nnnn))	without fgg with abs_fgg (*nnnn) with rel_fgg (+/-nn)	yes yes if (*nnnn) matches no
with relative fgg (rel_fgg (+/-nn)	without fgg with abs_fgg (*nnnn) with rel_fgg (+/-nn)	yes no yes if (+/-nn) matches

Table 5:

abs_fgg :	absolute generation number (*nnnn), 0001≤nnnn≤9999
rel_fgg :	relative generation number (+/-nn), 0≤nn≤99

Future system extensions may render this table invalid. Consequently, it should not be considered as a guaranteed interface.

– Search patterns for a file name may be of the data type <filename> or <partial-filename>.

– Similarly, a partially qualified file name can be used as a wildcard search pattern.

Examples

1. Check a value for its data type:

```
         ...
          CMDVAL MF=E,PARAM=MYPL
 *               returncode must be X'0140001C'
          LA     1,MYPL
          USING  MYPLD,1
          LH     2,DMDVMRET
          LTR    2,2
          BNE    ERRPROC
          ...
MYPL     CMDVAL MF=L,INPUT=BUFF@,DATATYP=FILENAME,ATTRIB=(NOCATID,WILDCARD)
          ...
```

```
BUFF@    DS     0F
BUFFL    DC     Y(BUFFEND-BUFF@)
BUFFFIL  DS     XL2
BUFFCT   DC     C':0Y:$TSOS.SDF-A'
BUFFEND  EQU    *
         ...
MYPLD    CMDVAL MF=D,PREFIX=D
         ...
```

2. Check whether an input value matches a wildcard search pattern:

```
        ...
         CMDVAL MF=E,PARAM=MYPL2
         LA     1,MYPL2
         USING  MYPLD,1
         LH     2,DMDVMRET
         LTR    2,2
         BNE    ERRPROC
         ...
MYPL2    CMDVAL MF=L,INPUT=BUFF@,PATTERN=PATT@
         ...
BUFF@    DS     0F
BUFFL    DC     Y(BUFFEND-BUFF@)
BUFFFIL  DS     XL2
BUFFCT   DC     C':0Y:$TSOS.SDF-A'
BUFFEND  EQU    *
         ...
PATT@    DS     0F
PATTL    DC     Y(PATTEND-PATT@)
PATTFIL  DS     XL2
PATTCT   DC     C':0Y:$TSOS.SD/-*'
PATTEND  EQU    *
         ...
MYPLD    CMDVAL MF=D,PREFIX=D
         ...
```

CMDWCC
Check wildcard syntax and perform pattern matching

The CMDWCC macro can be used to check a predefined wildcard pattern for correct syntax and to perform pattern matching. Examples demonstrating the use of CMDWCC can be found on page 461f.

Operation	Operands
CMDWCC	ACTION = list-poss(2): *CHECK / *MATCH
	,PAT@ = <var: pointer>
	,PATL = <var: int:4>
	,INP@ = NULL / <var: pointer>
	,INPL = 0 / <var: int:4>
	,FGG = *NO / *YES / <var: bit:1>
	,PART_Q = *NO / *YES / <var: bit:1>
	,SYNTAX = *BS2000 / *POSIX
	,WORK@ = NULL / <var: pointer>
	,PREFIX = C / <char:1>
	,MACID = MDW / <char:3>
	,MF = D / C / L / M / E
	,PARAM = <var: pointer>

ACTION =
Defines if only the wildcard syntax is checked or if pattern matching is also performed.

***CHECK**
The wildcard pattern is checked for correct syntax.

***MATCH**
The wildcard pattern is matched with the string specified by INP@. The syntax of the pattern should always be verified with *CHECK beforehand or at the same time.

PAT@ = <var: pointer>
Address of the wildcard pattern.

PATL = <var: int:4>
Length of the wildcard pattern.

INP@=
*Only for ACTION=*MATCH:*
Specifies the address of a string to be matched with the wildcard

<u>NULL</u>
No string is specified for comparison.

<var: pointer>
Address of a string.

INPL=<var: int:4>
Length of the string (only for ACTION=*MATCH).

FGG=
Specifies whether file generation groups are supported when matching the wildcard pattern with the input string.

<u>*NO</u>
File generation groups are not supported.

***YES**
File generation groups are supported.

<var: bit:1>

Bit variable:	bit = 0: file generation groups are not supported.
	bit = 1: file generation groups are supported.

PART_Q =
Specifies whether partially qualified file names are supported when matching the wildcard pattern with the input string.

<u>*NO</u>
Partially qualified file names are not supported.

***YES**
Partially qualified file names are supported.

<var: bit(1)>

Bit variable:	bit = 0: partially qualified file names are not supported.
	bit = 1: partially qualified file names are supported.

SYNTAX =
Defines which type of wildcard syntax is used.

<u>*BS2000</u>
The complete BS2000 wildcard syntax may be used.

***POSIX**
The POSIX wildcard syntax is used.

WORK@=
*Only for ACTION=*MATCH.*
Specifies the address of a working area for comparing the input string with the wildcard search pattern.

NULL
Specifies no work area.

<var: pointer>
the input string with the wildcard pattern. The work area must have a minimum length of (5*PATL/4 + 5)*4 bytes and must be aligned on a word boundary.

For a description of the parameters PREFIX, MACID, MF and PARAM: see section “Macro types” on page 379ff.

Return information and error flags

The return code is passed in the standard header of the parameter area.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	No errors
00	40	0001	Syntax error in pattern
00	40	0002	Input string does not match pattern
00	01	0008	Parameter error
01	00	0000	No wildcard in pattern

Notes

– The data type of a pattern for file names can be <filename> or <partial-filename>.

– A partially qualified file name may also be used as a wildcard pattern.

– The following table must be taken into consideration when matching a wildcard pattern with file names that have generation numbers. The table contains the currently applicable cases in which a match is possible.

For more information on file generation groups (fgg), see the “Introductory Guide to DMS” [7].

INPUT string	PATTERN string	Match possible?
without fgg	without fgg with fgg	yes no
with absolute fgg (abs_fgg (*nnnn))	without fgg with abs_fgg (*nnnn) with rel_fgg (+/-nn)	yes yes if (*nnnn) matches no
with relative fgg (rel_fgg (+/-nn)	without fgg with abs_fgg (*nnnn) with rel_fgg (+/-nn)	yes no yes if (+/-nn) matches

abs_fgg :	absolute file generation group (*nnnn), 0001≤nnnn≤9999
rel_fgg :	relative file generation group (+/-nn), 0≤nn≤99

This table may be rendered obsolete by future system extensions and should hence not be treated as a guaranteed interface.

Examples of the use of CMDWCC

```
EXWCC   START
        BASR  10,0
        USING *,10
*
* HOW TO USE CMDWCC
*
*
* EXAMPLE 1: PL1 IS INITIALIZED USING MF=L
*
EX1     LA 1,PL1
        CMDWCC MF=E,PARAM=PL1
        B EX2
PL1     CMDWCC MF=L,PAT@=A(PAT),PATL=8,INP@=A(INP),INPL=8,            *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(WA)
*
* EXAMPLE 2: PL2 IS INITIALIZED USING MF=L WITH DUMMY VALUES
*            AND MODIFIED USING MF=M BEFORE EXECUTION
* NOTE: MF=M ASSUME THAT THE PARAMETER AREA IS POINTED BY REGISTER 1
*
EX2     LA 1,PL2
        USING CWCC_MDL,1
        CMDWCC MF=M,PATL=PATL,INPL=INPL,PAT@=A(PAT),INP@=A(INP),      *
               WORK@=A(WA)
        CMDWCC MF=E,PARAM=PL2
        B EX3
PL2     CMDWCC MF=L,PAT@=A(0),PATL=0,INP@=A(0),INPL=0,                *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(0)
*
* EXAMPLE 3: PL3 IS INITIALIZED USING MF=L WITH DUMMY VALUES
*            AND MODIFIED USING MF=M BEFORE EXECUTION IN ANOTHER WAY
* NOTE: MF=M ASSUME THAT THE PARAMETER AREA IS POINTED BY REGISTER 1
*
EX3     LA 1,PL3
        USING CWCC_MDL,1
        LA     2,PAT
        ST     2,PATADD
        LA     2,INP
        ST     2,INPADD
        LA     2,WA
        ST     2,WAADD
        CMDWCC MF=M,PATL=PATL,INPL=INPL,PAT@=PATADD,INP@=INPADD,      *
               WORK@=WAADD
        CMDWCC MF=E,PARAM=PL3
        TERM
PL3     CMDWCC MF=L,PAT@=A(0),PATL=0,INP@=A(0),INPL=0,                *
               SYNTAX=*BS2000,ACTION=*MATCH,WORK@=A(0)
PATADD  DS     A
INPADD  DS     A
WAADD   DS     A
*
* PATTERN AND INPUT
*
PAT     DC     C'PATTERN*         '
PATL    DC     F'8'
INP     DC     C'PATTERNA         '
INPL    DC     F'8'
WA      DS     0A
        DS     XL1000
        CMDWCC MF=D
        END
```

CMDWCO
Wildcard constructor

The CMDWCO macro generates a new string from an existing string with the aid of a selector and constructor. The output string and its length are stored in the parameter area in the fields '<prefix><macid>NAM' and '<prefix><macid>LEN', respectively. The input string and the output string may be of any SDF data type. The user should therefore perform a data type analysis independently (by using the CMDVAL macro), since the input string is not compared syntactically with the output string in this case.

Operation	Operands
CMDWCO	SELECT = <c-string 1..255> / <var: char(255)>
	,CONSTR = <c-string 1..255> / <var: char(255)>
	,SRCNAME = <c-string 1..255> / <var: char(255)>
	,MAXLEN =255 / <integer 1..255> / <var: integer(1)>
	,PARTIAL = *NO / *YES / bit(1)
	,SYNTAX = *BS2000 / *POSIX
	,PREFIX = C / <char(1)>
	,MACID = MDW / <char(3)>
	,MF = D / C / L / M / E
	,PARAM = <var: pointer>

SELECT =
Specifies the selector, i.e. a selection string consisting of a combination of a wildcard pattern and constant name components (see the description of the SDF metasyntax in section 1.5). The selector can have a maximum length of 255 characters. If it is shorter, it must be terminated by at least one blank. If an empty selector (blanks) is specified in combination with PARTIAL=*YES, all names are selected, with the same handling as for partially qualified file names.

<c-string 1..255>
Specification in the form of a C string constant.

<var: char(255)>
Specification in the form of a string variable.

CONSTR =
Specifies the constructor, i.e. a construction string consisting of a combination of a construction pattern and constant name components (see the description of the SDF metasyntax in section 1.5). The constructor can have a maximum length of 255 characters. If it is shorter, it must be terminated by at least one blank. If an empty constructor (consisting of blanks) is specified in combination with PARTIAL=*YES, all names are selected, with the same handling as for partially qualified file names.

<c-string 1..255>
Specification in the form of a C string constant.

<var: char(255)>
Specification in the form of a string variable.

SRCNAME =
Specifies the source string referenced by the selection pattern. This string can have a maximum length of 255 characters. If it is shorter, it must be terminated by at least one blank.

<c-string 1..255>
Specification in the form of a C string constant.

<var: char(255)>
Specification in the form of a string variable.

MAXLEN =
Specifies the maximum length of the output string to be generated.
If a generated output string is longer than the value specified here, an error code is returned, and only the number of characters specified for MAXLEN is entered in the parameter area.

255 / <integer 1..255>
Specifies the maximum length. Default a length of 255 bytes.

<var: int(1)>
Specifies the maximum length by means of an integer variable with a length of 1 byte.

PARTIAL =
Partial qualification is allowed. The presence of a period at the end of the selection or construction string (i.e. the selector or constructor) is interpreted as a partial qualification. This is also assumed if the selector and constructor are blank. An empty string (blanks) is interpreted as the wildcard ’*’. This implicitly constructed pattern cannot be referenced by ’*’ in the constructor character string.
The string ’xxxx. ’ is interpreted as ’xxxx.*’.

<u>*NO</u>
A period at the end of the selector or constructor is interpreted as a constant, so blank selectors can only be used to select blank SRCNAMEs, and blank constructors can only be used to generate blank file names.

***YES**
A period at the end of the selector or constructor is interpreted as a partial qualification. This also applies to empty strings.

<var: bit(1)>

Bit variable:	bit = 0: partial qualification is not allowed.
	bit = 1: partial qualification is allowed.

SYNTAX =
Defines which type of wildcard syntax is used.

<u>*BS2000</u>
The complete BS2000 wildcard syntax may be used.

***POSIX**
The POSIX wildcard syntax is used.

<var: bit(1)>

Bit variable:	bit = 0: as for *BS2000
	bit = 1: as for *POSIX

For a description of the parameters PREFIX, MACID, MF and PARAM: see section “Macro types” on page 379.

Return information and error flags

The return code is passed in the standard header of the parameter area.

If SC1=X'00' applies, the output fields will contain valid information.:

<prefix><macid>LEN :	Length of the generated string.
<prefix><macid>NAM :	Generated string, followed by blanks.

Standard header

c	c	b	b	a	a	a	a

cc: Subcode 2 (SC2)
bb: Subcode 1 (SC1)
aaaa: Maincode

(SC2)	SC1	Maincode	Meaning
00	00	0000	No errors
00	40	0002	Invalid selector or SRCNAME not selectable
00	40	0003	Syntax error in constructor or semantic error
00	01	0004	Generated string is longer than MAXLEN. The output area is only filled up to a length of MAXLEN.
00	01	0008	Error in parameter list (access error or parameter not supported)

OPNCALL
Create program context

OPNCALL creates a program context and opens the syntax files specified by the user. If *STD (= default) is specified, the syntax file used by the system context is opened again. The macro returns the context identifier (= CALLID) to the user.

OPNCALL can be used to open a program context corresponding to the system context, in which user programs can activate their own user syntax files without affecting the current user syntax file in the system context. The user syntax file of the program is opened in parallel to the user syntax file of the system concept.

Operation	Operands
OPNCALL	CALLID = addr / (r)] [,TASKTYP = <u>ANYTASK</u> / TERMINAL / BATCH / ALL] [,SFSYSTM = <u>*STD</u> / filename] [,SFGROUP = <u>*STD</u> / filename] [,MF = { L / (E,(1)) / (E,opaddr) }]

CALLID =
Address of the identifier of the opened context, which is determined by further macro calls.

addr / (r)
Address of a 4-byte field or a register in which SDF passes the context identifier. The caller must specify the context identifier when referring to the program context. This field must be aligned on a word boundary.

TASKTYP =
Specifies the environment in which the caller is working when the CMDTST and TRCMD macros are called. SDF checks whether the specified statements or commands are allowed in the defined environment. If not, they are rejected by SDF.

<u>**ANYTASK**</u>
SDF does not check whether the inputs are allowed.

TERMINAL
SDF accepts only the statements and commands which are defined in the syntax files of the context with DIALOG-ALLOWED=*YES or DIALOG-PROC-ALLOWED=*YES (see the PROCMOD operand of the CMDTST and TRCMD macros).

BATCH
The statements must be defined with BATCH-ALLOWED=*YES or BATCH-PROC-ALLOWED=*YES.

ALL
SDF checks for both dialog and batch tasks to ascertain whether the statements are allowed. If a statement is not allowed for either type of task, it is rejected by SDF.

In the case of subsequent CMDTST or TRCMD calls, the following process can be expected:

– in the case of PROCMOD=ANY, all statements and commands are rejected
– in the case of PROCMOD=NO, statements and commands defined with DIALOG-ALLOWED=*NO or BATCH-ALLOWED=*NO are rejected
– in the case of PROCMOD=YES, statements and commands which are defined with DIALOG-PROC-ALLOWED=*NO or BATCH-PROC-ALLOWED=*NO are rejected.
– in the case of PROCMOD=ALL (only TRCMD) all statements and commands which are only permitted in interactive mode (defined with DIALOG-PROC/BATCH-PROC/BATCH-ALLOWED=*NO) are rejected.

SFSYSTM =
Specifies the system syntax files which are activated in the program context being created.

<u>*STD</u>
The system syntax files of the system context are activated in the program context. If system administration switches these system syntax files, the system syntax files of the program context are modified accordingly.

filename
Name of the system syntax file to be activated. The specified system syntax file in the program context can be switched only after a CLSCALL. It applies to the user task which issued the OPNCALL macro. Another user task can access this syntax file only after it has issued its own OPNCALL macro.

SFGROUP =
Specifies the group syntax file to be activated.

***STD**
The group syntax file of the system context is activated. This is assigned to the profile ID of the user task.

***NO**
No group syntax file is activated in the program context.

filename
Name of the group syntax file which is to be opened in the program context.

MF =
Defines special requirements for macro expansion (see the “Executive Macros” manual [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This either appears in register 1 or is specified directly.

Register usage

Register 1: address of the parameter list

Return information and error flags

Register 15 contains a return code in the right-most byte.

X’00’	normal execution
X’44’	syntax file not found
X’54’	maximum permissible number of contexts exceeded

TRCMD
Analyze command

The caller uses the TRCMD macro to transfer a command to SDF as a character string. SDF analyzes this command and generates a log, INVARIANT-INPUT form or ACCEPTED-INPUT form and the internal form.

The internal form (also called the converted form) is precisely the information which is necessary for calling the command server and therefore for executing the command. The internal form is either a character string or a standardized transfer area, depending on how the command is defined in the syntax file.

SDF also transfers the result of the syntax analysis to the program.
In interactive jobs, the command can be corrected in the course of an error dialog.

The INVARIANT-INPUT form of the statement contains all the specified operands, all the operands which have default values and all the operands which are currently permitted for the task (see also CMDRST, page 405).

The ACCEPTED-INPUT form contains all the names in their long form, all the specified operands with their name and value and any corrections. As there are no abbreviations in the ACCEPTED-INPUT form, their uniqueness is guaranteed for later BS2000 versions. Passwords and secret operands are not masked out.

Operation	**Operands**
TRCMD	INPUT = *NO / addr / (r)
	,OUTPUT = *NO / addr / (r)
	,CMD = <u>*ALL</u> / (name,...)
	,DIALOG = <u>NO</u> / YES / ERROR
	,MESSAGE = <u>*NO</u> / addr / (r)
	,ERROR = <u>NO</u> / YES
	,PROT = <u>*YES</u> / *NO / addr / (r)
	,MEMORY = <u>ACTUAL</u> / BASIC
	,CALLINF = <u>*NO</u> / addr / (r)
	,EXIT = <u>YES</u> / NO
	,CALLID = <u>*NO</u> / addr / (r)
	,EXECUTE = <u>NO</u> / YES
	,PROCMOD = <u>ANY</u> / YES / NO / ALL

continued ➠

Operation	Operands
	,CHKPRV = <u>*YES</u> / *NO / addr
	,DUMESC = <u>NO</u> / YES
	,OVERLNG = <u>NO</u> / YES
	,SETVAR = <u>NO</u> / YES
	,INVAR = <u>*NO</u> / addr / (r)
	,ACCEPT = <u>*NO</u> / addr / (r)
	,DEFDEF = <u>NO</u> / YES
	[,MF = { L / (E,(1)) / (E,opaddr) }]

INPUT =
Determines which command SDF should analyze.
Specification of this parameter is mandatory.

***NO**
SDF is not to analyze any commands stored in the program, but rather the first command generated by a system exit. All other commands generated by a system exit can be requested by the repeated call of TRCMD with INPUT=*NO.

addr / (r)
SDF is to analyze the command whose address is specified or is in the specified register. SDF expects the command character string as a variable-length record in the standard BS2000 format. The record area must be aligned on a halfword boundary.

OUTPUT =
determines whether the internal form of the command is generated.
Specification of this parameter is mandatory.

***NO**
The internal form of the command is not generated. The syntax of the command specified at INPUT is checked.

addr / (r)
Address of an area in which SDF saves the internal form, or a register which contains this address. The internal form can be processed by the command server. Depending on the command definition in the syntax file, the internal form is either:
– a character string (see the statement ADD-CMD ... IMPLEMENTOR=*STRING/*PROCEDURE...)

– a standardized transfer area (see the statement ADD-CMD ... IMPLEMENTOR=*NEW/*TRANSFER-AREA...).

The area must begin on a word boundary. Before TRCMD is called, the program must ensure that the maximum possible area length is in the first halfword of the area.

CMD =
Determines which commands are permitted as input.

<u>*ALL</u>
All commands of the current syntax hierarchy are permitted.

(name,...)
Only commands whose RESULT-INTERNAL-NAME is specified are permitted. The RESULT-INTERNAL-NAME is in stored in the command definition of the syntax file (see the ADD-CMD statement). The names specified here must be of the data type <alphanum-name 1..8>

DIALOG =
Determines whether SDF is to conduct a dialog during the command analysis. This operand is only relevant when the program is executing in an interactive task.

<u>NO</u>
SDF is not to conduct a dialog.

YES
SDF is to offer the command transferred by the program to the user in the dialog for any changes, as far as this is compatible with the valid SDF specifications for the dialog (see MODIFY-SDF-OPTIONS and SET-GLOBALS).

ERROR
SDF is only to conduct a dialog when syntax errors are detected.

MESSAGE =
Determines whether SDF is to output a message on SYSOUT if the user is offered the command for checking and any changes (only relevant for DIALOG ≠ NO). SDF integrates this message into the form.

<u>*NO</u>
SDF is not to issue a message.

addr / (r)
Address of the message text to be output (aligned on a halfword boundary) or register which contains this address. The text is expected as a variable-length record with a maximum length of 400 characters. In guided dialog only the first 280 characters are represented. SDF interprets the first byte of the message text as a printer control character. If the text contains screen control characters, the menu mask can be destroyed.

ERROR =
Determines how the message text specified for the MESSAGE operand is output.

<u>NO</u>
SDF is to output the message text as a message.

YES
SDF is to output the message text as an error message.

PROT =
Controls the log output for the input command.

<u>*YES</u>
SDF is to write the log form to SYSOUT.

***NO**
SDF is to not output a log form.

addr / (r)
Address of a log buffer (aligned on a halfword boundary) or register which contains this address. SDF is only to write the log form of the command and messages in the buffer, not to SYSOUT. The length of the buffer is given in the first halfword of the buffer. The length of the buffer actually used is given in the second halfword. Then come the log records with variable record length.
If the buffer is not empty, the first record is normally the log of the input command. The other records contain messages of all types. If no input log is available or can be output, a slash (/) is written in the output area.

MEMORY =
Specifies whether the definition of the command is to be read from the current syntax file hierarchy or only from the basic system syntax file.

<u>ACTUAL</u>
The command is analyzed in the current syntax file hierarchy.

BASIC
Only the command definition in the basic system syntax file is analyzed. Definitions from a group or user syntax file are ignored.

CALLINF =
Determines whether call information is generated for the command server. If SDF is to execute the commands MODIFY-SDF-OPTIONS and SHOW-SDF-OPTIONS, this call information must be transferred by the program.

CALLINF = <u>*NO</u>
The call information is not generated. No SDF command is executed.

CALLINF = addr / (r)
Address of an area which is 102 bytes long and aligned on a halfword boundary, or a register which contains this address. SDF stores the call information for the input command here.

EXIT = <u>YES</u> / NO
Determines whether system exits (80/81) are called for analyzing the input command.

CALLID =
Determines in which context SDF analyzes the command.

<u>*NO</u>
The currently activated syntax file hierarchy is used (the system context).

addr / (r)
Address of a 4-byte field (aligned on a word boundary), or a register which contains this address. In this, the caller transfers the CALLID of the program context that it wishes to use.

The CALLID of the program context is provided from a previous OPNCALL call. With OPNCALL the input mode for the task can also be specified (parameter TASKTYP).

EXECUTE = <u>NO</u> / YES
Determines whether the SHOW-SDF-OPTIONS and MODIFY-SDF-OPTIONS commands are executed. A precondition for this is that in the case of CALLINF the call information is transmitted for the command. EXECUTE has no meaning if TRCMD refers to the system context rather than the program context (i.e. CALLID=*NO). The commands are always executed by TRCMD.

PROCMOD =
Determines which environment the user works in. SDF carries out a check: commands which are not permitted in the specified environment are not accepted by SDF. This operand refers to the possibility of opening several syntax file hierarchies. It is only important when the macro call refers to one of the new syntax hierarchies which are opened in addition to the current one. In the case of CALLID=*NO, PROCMOD is irrelevant, i.e. the value ANY is

set automatically and the current procedure mode is valid.
For further information on the PROCMOD parameter see the OPNCALL macro, parameter TASKTYP=ALL (page 468).

ANY
No check is carried out

YES
Commands are handled as though they are being read from a procedure file, i.e. they are analyzed if they are defined in the syntax file with DIALOG-PROC-ALLOWED=*YES and if the program is running in interactive mode, or in the case of BATCH-PROC-ALLOWED=*YES in batch jobs.

NO
Commands are handled as though they are being read from one of the primary levels, e.g. from the screen input or from a batch job. They are analyzed if they were defined in the syntax file with DIALOG-ALLOWED=*YES and if the program is running in interactive mode or in the case of BATCH-ALLOWED=*YES in batch jobs.

ALL
Commands are checked for all procedure modes. Depending on the TASKTYP parameter for the OPNCALL macro, the following behavior is observed:

– for TASKTYP=ANY:
  Every command is considered to be illegal and is rejected (illegal parameter combination)
– for TASKTYP=BATCH:
  The command is analyzed if it is defined in the syntax file with BATCH-ALLOWED=*YES or BATCH-PROC-ALLOWED=*YES
– for TASKTYP=TERMINAL:
  The command is analyzed if it is defined in the syntax file with DIALOG-ALLOWED=*YES or DIALOG-PROC-ALLOWED=*YES

CHKPRV =
Determines whether SDF checks the privileges of the commands.
The parameter is only meaningful in the program context (i.e. for CALLID ≠ *NO).

<u>*YES</u>
The privileges are checked as a function of the permitted input mode (TRCMD PROC-MOD=... , OPNCALL TASKTYP=...).

***NO**
The privileges of the commands are not checked. The command is not executed, i.e. EXECUTE=NO is always valid.

addr
Address of an 8-byte area in which a 64-bit privilege mask is stored. The command is checked against this privilege mask to determine whether it is permitted in a particular input mode. The order of the privileges in the bit mask is the same as the order in which the privileges are currently defined in the system. These privilege masks can be output with the SDF-I statement SHOW-SYNTAX-FILE for example (see the “SDF Management“ manual [2]).

DUMESC =
Specifies whether expression substitutions which begin with an escape sign can be considered to be part of the command or ignored. Expression substitutions are normally specified in the form '&proc-parameter' or '&(expression)'.

<u>NO</u>
Specifies whether expression substitutions are considered to be part of the command by SDF and syntactically analyzed as such. In most cases (with the exceptions of <text> and <cmd-rest>) SDF returns a syntax error.

YES
Expression substitutions which begin with the escape character are not subjected to a syntax analysis by SDF. The keyword BY-AND is also ignored.

OVERLNG =
Determines whether SDF is to overwrite the length field of the area specified in OUTPUT (first halfword) with the actual length of the internal form (ISP character string or standardized transfer area). However, this only occurs in the case of a successful function call and if the OUTPUT area is not too short.

<u>NO</u>
The length field of the OUTPUT area is not overwritten.

YES
In the case of a successful function call, the length field is overwritten with the actual length (including the length field) of the internal command form.

SETVAR =
Specifies whether the command can be an assignment of the form '<variable-name>=value' if it is entered in an S procedure. In non-S procedures, there may be conflicts with the permissible ISP inputs for /REMARK, /TYPE, /PAUSE etc.

<u>NO</u>
Equals signs are considered to be part of the operand list of the command.
For reasons of compatibility NO is the default value.

YES
Equals signs are interpreted as variable assignments. By default, /SET-VARIABLE is used as the command name. If this command is not defined in the syntax hierarchy, the input command is rejected with a corresponding error message. If SDF could successfully identify the variable assignment, the requested outputs are produced by SET-VARIABLE on the basis of the command definition.

INVAR =
Specifies whether the INVARIANT-INPUT form of the command is saved, i.e. that the statement is stored with all the input operands, all the operands with default values and all the operand values which are permitted for the task dependent on the input mode and on the privileges:

1. The input mode (PROCMOD parameter) is only checked if CALLID is also specified. The dependencies between CALLID and PROCMOD are included in the description of the PROCMOD parameter
2. The privileges in the case of CHKPRV are only taken into account if CALLID is also specified. If CALLID is not specified, TRCMD uses the current task privileges.

Keywords and secret operands are not masked out.
INVAR must not be specified together with ACCEPT.

<u>*NO</u>
The INVARIANT-INPUT form of the statement is not saved.

addr / (r)
Specifies the address of a buffer aligned on a word boundary in which SDF writes the INVARIANT-INPUT form of the command. The first halfword must contain the length of the buffer. If the buffer is too short, the corresponding macro return code is returned. SDF stores the INVARIANT-INPUT form as of the second halfword as a variable-length record.

The buffer contains the following:

2 bytes	2 bytes	2 bytes	
buflen	reclen	filler	invariant-input

buflen:	Length of the buffer
reclen:	Length of the record which SDF writes
filler:	Filler
invariant-input:	INVARIANT-INPUT form of the command, beginning at the seventh byte

ACCEPT =
Specifies whether the ACCEPTED-INPUT form of the command is saved. The ACCEPTED-INPUT form contains all the names in their long form and all the input operands with their names and values (as permitted for the task dependent on the input mode and the privileges):

1. The input mode (PROCMOD parameter) is only checked if CALLID was also specified. The dependencies between CALLID and PROCMOD are outlined in the description of the PROCMOD parameter.
2. The privileges for CHKPRV are only taken into account if CALLID is also specified. If CALLID is not specified, TRCMD uses the current task privileges.

Keywords and secret operands are not masked out.
INVAR must not be specified together with ACCEPT.

ACCEPT = <u>*NO</u>
The ACCEPTED-INPUT form of the statement is not saved.

ACCEPT= addr / (r)
Specifies the address of a buffer aligned on a word boundary in which SDF writes the ACCEPTED-INPUT form of the command. The first halfword must contain the length of the buffer. If the buffer is too short, the corresponding macro return code is returned. SDF stores the ACCEPTED-INPUT form as of the second halfword as a variable-length record.

The buffer contains the following:

2 bytes	2 bytes	2 bytes	
buflen	reclen	filler	invariant-input

buflen:	Length of the buffer
reclen:	Length of the record which SDF writes
filler:	Filler
accepted-input:	ACCEPTED-INPUT form of the command, beginning at the seventh byte

DEFDEF =
Specifies whether commands for setting task-specific default values may be input (see the “Introductory Guide to the SDF DIalog Interface“ [1]).

<u>NO</u>
Commands for setting task-specific default values are not accepted.

YES
Commands for setting task-specific default values are accepted. The set default values remain valid until they are explicitly changed. SDF transfers a special return code if it recognizes a default value definition, and in this case it ignores the output parameters. For syntax errors the normal return code is returned.

MF =
Defines special requirements for the macro expansion (see the “Executive Macros“ manual [8]).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code occur in the macro call. The generated data part has the address given in the name field of the macro call.

(E,(1)) / (E,opaddr)
Only the command part of the macro expansion is generated. The corresponding data part (operand list) is indicated by the address “opaddr“. This is either in register 1 or is specified directly.

Return information and error flags

The format of the transfer area is described on page 365ff.

Register 15 contains a return code in the right-most byte:

X’00’	Normal termination; command is syntactically correct
X’04’	Unrecoverable system error
X’08’	Operand error in the macro
X’0C’	Transfer area too small
X’10’	EOF detected
X’1C’	Error in command or command unknown
X’20’	Error dialog not possible
X’24’	Error dialog rejected
X’28’	Error or log area too small

X’30’	Time out
X’38’	SDF not available
X’44’	Syntax file not found
X’48’	SDF command carried out
X’5C’	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
X’6C’	Command for setting task-specific fault values recognized
X’70’	String passed is empty

Indicator for additional command (via a system exit):

If a 1:n conversion of the input command has been carried out via a system exit, the flag shows FURTH in the call information (CALLINF), indicating that further commands can be requested with TRCMD INPUT=*NO.

6.5 Interface between SDF and high-level languages

As of Version 3, SDF incorporates an interface for high-level languages (HLLs) such as COBOL, FORTRAN and C. This interface makes it possible to access SDF macros from within HLL programs, without writing Assembler programs for calling the SDF macros.

Figure 15 is a block diagram illustrating the HLL interface.

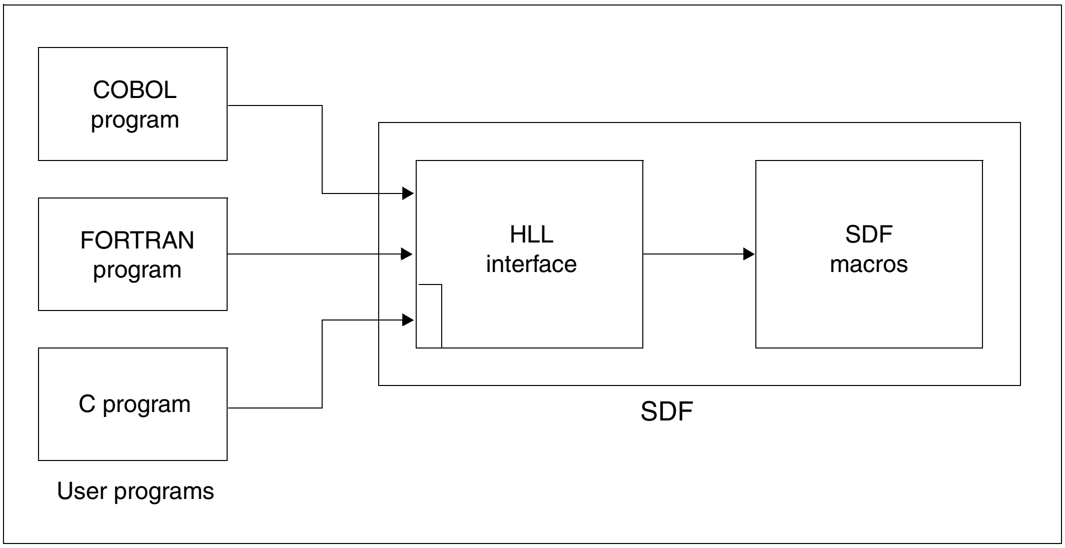


Figure 15: Interface between SDF and high-level languages

The following SDF macros are supported by the HLL interface:

- macros for processing statements (RDSTMT, CORSTMT, TRSTMT)
- the macro for calling a system command (CMD)
- the macro for setting the command return code (CMDRC)
- the macro for outputting information about activated syntax files (CMDSTA).

i The interface to higher programming languages only supports the standardized transfer area in the old format (see page 593). The new format (see page 365ff) can only be used via the Assembler interface!

6.5.1 Interface conventions

The HLL interface is available directly for all programming languages that use the same conventions for transfer parameters to subprograms as COBOL and FORTRAN. For these programming languages, the function calls described on page 483ff apply.

The following conventions apply:

– Register 1 must contain a pointer to the list of addresses of the transfer parameters.

– The most significant bit in the last parameter address must be set (logically ORed with X'80000000').

– Register 13 contains the address of the register backup area (normally set by the program manager).

Calls that comply with the ILCS conventions are also supported. The following conventions apply:

– Register 1 must contain a pointer to the list of parameter addresses.

– Register 0 must contain the number of parameters.

– Register 13 must contain the address of the register backup area (normally set by the program manager).

An interface satisfying these requirements was developed especially for C. This C interface is described in a separate section (see section “C interface” on page 503ff).

Under certain circumstances, the HLL interface can also be used by Assembler programs. The preconditions that apply are as follows:

– The requisite parameter list must be available.

– The address of the parameter list must be passed to SDF in register 1.

– Register 13 must contain the address of a register backup area 18 words long which must be deleted before the call.

– The entry name is 'SDF'.

The interface modules and the include elements are supplied in the library SYSLIB.SDF.045.

6.5.2 COBOL and FORTRAN interface

6.5.2.1 Description of the function calls

Function calls

The notation adopted for describing the function calls is:

```
CALL SDF(<parameterlist>)
```

The contents of <paramlist> are shown in the description of the function call. The exact syntax of the call varies depending on the language.

Notational conventions

Since the exact syntax of the function calls is language-dependent, certain notational conventions have been adopted for the descriptions below. The parameter list always appears in parentheses (). Square brackets inside the parameter list always enclose an optional parameter or parameters.

The list itself is always followed by a tabular description:

Column 1: Name of the parameter

Column 2: Data type of the parameter and its length.
The following data types occur:

char	character string
integer	integer (always 4 bytes long)
ptr	address
V-rec@	address of a variable-length record

Column 3:

in	input parameter
out	output parameter
inout	input/output parameter

Column 4: Explanatory remarks

Information returned by the functions

Each function returns information in the form of a return code. SDF writes this return code into the “error” parameter, which must be present in the parameter list of each function.

The return codes may belong to one of three classes:

error=0:	Command executed without error.	
error>0:	Error in the associated macro. The error code has been copied from register 15 into the ’error’ parameter. The value of the error code corresponds to that of the macro error code.	
error<0:	-1 :	Function unknown
	-2 :	Not enough operands
	-3 :	Last operand in structure reached
	-4 :	Last operand in list reached
	-5 :	Position is negative
	-6 :	Operand is not of the type LIST
	-7 :	Operand is not of the type STRUCTURE

6.5.2.2 Overview of SDF function calls

CCMD	Execute a system command
CORR	Set correction bit
INIT	Initialize buffer
LEVL	Position on an operand array
OPER	Read operand value from the transfer area
READ	Read and analyze a statement
SEMA	Initiate a semantic error dialog
STAT	Output information about syntax files
STMT	Read statement name from the transfer area
STRU	Analyze data type and length of value-introducing structure
TRNS	Analyze a statement
TYPE	Analyze data type and length of operands
WRRC	Set command return code

CCMD
Execute system command

The CCMD function calls a command without exiting the program mode. The CCMD function is based on the CMD macro described in the “Executive Macros” manual [8].

Call

```
CALL SDF('CCMD',area,error,sysout,dialog,list,cmdrc,buff)
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
CCMD	char(4)	in	Function name: keyword CCMD
area	V-rec@	in	Address of a variable-length record containing the command to be executed
error	integer	out	Return code of the function
sysout	integer	in	1 : SYSOUT = YES 0 : SYSOUT = NO
dialog	integer	in	1 : DIALOG = YES 0 : DIALOG = NO
list	integer	in	1 : LIST = YES 0 : LIST = NO
cmdrc	char(9)	in	Address of an area, 9 bytes in length, into which the command processor will write the command return code. Must be aligned on a halfword boundary.
buff	V-rec@	in	Address of a variable-length record for the output buffer (BUFMOD=SHORT, see the CMD macro)

i The parameter list corresponds to PARMOD=31 with specification of the CMDRC or LIST parameter (see the description of the CMD macro in the “Executive Macros” manual [8]).

CORR
Set correction bit

The CORR function sets a correction bit for a given operand. When this correction bit is set, the operand in question is underscored in the error dialog for a statement, if it was invalid.

Call

```
CALL SDF('CORR',area,error,pos)
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
CORR	char(4)	in	Function name: keyword CORR
area	char()	inout	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
pos	integer	in	Position of the operand whose correction bit is to be set

INIT
Initialize buffer

The INIT function initializes the buffer in which the standardized transfer area is stored. The internal name of the program must be specified as it is defined in the syntax file.

The INIT function must be called once before all other SDF functions that use the standardized transfer area.

Call

```
CALL SDF('INIT',area,error,lng,program)
```

Description of the parameter list

Parameter	**Data type (length)**	**Input/ output**	**Meaning**
INIT	char(4)	in	Function name: keyword INIT
area	char(lng)	inout	Buffer for creating the standardized transfer area, must be aligned on a word boundary
error	integer	out	Return code
lng	integer	in	Length of the buffer for the standardized transfer area
program	char(8)	in	Internal program name (as defined in the syntax file)

LEVL Position on operand array

The LEVL function enables the user to position on an operand array belonging to a structure description (see section “Format of the standardized transfer area” on page 365).

When the INIT function is called, the system positions on the operand array of the highest level, in other words on the operand array whose operand was defined with the SDF-A statement ADD-OPERAND ...,RESULT-OPERAND-LEVEL=1.

The LEVL function always refers to the current operand array. This can be the operand array of a structure description, if the LEVL function has already been called beforehand.

The structure description can also be an element in a list. In this case, the user must specify both the position of the list in the current operand array and the position of the structure description in the list.
If the structure description is not an element in a list, the user needs to specify only the position of the structure description in the operand array.

To return to the operand array of the highest level, enter 0 for the position.

Call

```
CALL SDF('LEVL',area,error,pos[,lst])
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
LEVL	char(4)	in	Function name: keyword LEVL
area	char()	in	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
pos	integer	in	– Position of the structure description in the current operand array or – position of the list containing the structure description pos=0 positions on the operand array of the highest level.
lst	integer	in	Relevant only if the structure description is an element in a list: position of the structure description in the list

i

Direct access to operand array positions greater than 2 is not possible.

Reverse positioning by the system is only possible on the operand array of the highest level. It is the responsibility of the user to trace the path to reverse-position on an intermediate level.

OPER
Read operand value from transfer area

The OPER function reads the value of an operand at a certain position from the standardized transfer area. The OPER call must be preceded by a TYPE or STRU call defining the operand type and operand length.

Call

```
CALL SDF('OPER',area,error,pos,val,lng[,lst])
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
OPER	char(4)	in	Function name: keyword OPER
area	char()	in	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
pos	integer	in	Position of the operand
val	char(lng)	inout	Address of a string in which the value is to be stored
lng	integer	in	Length of the val string must be greater than or equal to the value returned by TYPE or STRU for the lng parameter
lst	integer	in	Relevant only if the operand is an element in a list: position of the operand in the list

i If the OPER function is called for an operand of the type STRUCTURE, it returns the value of the operand introducing the structure, whose length and type were analyzed beforehand by the STRU function.

READ
Read and analyze statement

The READ function causes SDF

- to read a program statement from SYSSTMT. (The assignment for the SYSDTA system file also applies to SYSSTMT. The rules governing continuation lines, continuation characters and comments in SYSSTMT are the same as those that apply to SYSCMD.)
- analyze the statement read in and
- transfer the result of this analysis to the program.

The precondition is that an activated syntax file contains the definition of the program and its statements. For details, see the description of the RDSTMT macro (page 606ff).

Call

```
CALL SDF('READ',area,error[,msg[,allow[,prefer[,default1,...]]]])
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
READ	char(4)	in	Function name: keyword READ
area	char()	inout	Buffer for creating the standardized transfer area (see INIT)
error	integer	out	Return code
msg	V-rec@	in	Address of a variable-length record containing the text of a message to be output before the statement is read (max. 400 characters in length)
allow	ptr	in	Address of a structure containing the list of valid statements1)
prefer	char(8)	in	Only in guided dialog: internal name of the statement expected next
default1, ...	char() (standard. transfer areas)	in	List of up to 5 conversion descriptions with which default values of operands can be replaced by values generated dynamically by the program 2)

1) Format of the list of valid statements:

N	statement $_1$	statement $_2$	.....	statement$_n$

N	(2 bytes) :	number of statements in the list
statement $_x$	(8 bytes) :	internal name of a valid statement

Table 6:

2) With the aid of conversion descriptions, default values of operands defined in the syntax file with ADD-VALUE ...,VALUE=<c-string>...(OVERWRITE-POSSIBLE=*YES) or ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES can be replaced by values generated dynamically by the program. Only one conversion description can be specified per statement. The conversion description contains the internal statement name and the information defining which operand values can be replaced and how. A conversion description can be generated either by a READ call executed beforehand or by a TRNS call in which the new values are entered. The description has the format of a standardized transfer area.

The ’area’ buffer contains pointers to absolute addresses. Consequently, it cannot be copied into another memory area for evaluation.

If a zero (address=0 or blank string) is specified for the parameters msg, allow and prefer, the corresponding value is ignored.

The following are ignored:

– a statement name beginning with a blank “␣“ specified for prefer

– a record of length 0 specified for msg

– a list containing 0 elements specified for allow.

The RDSTMT macro is called with the following defaults:

```
PROT=*YES
BUFFER=*NO
INVAR=*NO
SPIN=*NO
ERRSTMT=*STEP
CALLID=*NO
CCSNAME=*NO
```

SEMA
Initiate semantic error dialog

The SEMA function causes SDF to initiate a dialog with the user, providing the opportunity to correct a semantic error in a statement. Immediately beforehand, SDF analyzes the statement and transfers it to the program as syntactically correct. For detailed information, see the description of the CORSTMT macro (page 601ff).

Call

```
CALL SDF('SEMA',area,error,msg)
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
SEMA	char(4)	in	Function name: keyword SEMA
area	char()	inout	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
msg	V-rec@	in	Address of a variable-length record containing the text of a message to be output during the error dialog (up to 400 characters long)

The CORSTMT macro is called with the following defaults:

```
DEFAULT=*NO
INVAR=*NO
CALLID=*NO
CCSNAM=*NO
```

STAT
Output information about syntax files

The STAT function outputs information concerning activated syntax files and the specifications applying to command/statement input and processing.
For a description of the transfer area for the information, see the CMDSTA macro (page 425ff).

Call

```
CALL SDF('STAT',area,error[,lng])
```

Description of the parameter list

Parameter	**Data type (length)**	**Input/ output**	**Meaning**
STAT	char(4)	in	Function name: keyword STAT
area	char()	out	Transfer area in which the information is stored by SDF. Must be aligned on halfword boundary.
error	integer	out	Return code
lng	integer	in	Length of transfer area: – If lng is specified, the length of the transfer area is lng and FORM=LONG is output (see CMDSTA) – If lng is not specified, the transfer area must be 530 bytes in length and FORM=SHORT is output (see CMDSTA)

STMT
Read statement name from transfer area

The STMT function reads the internal name of a statement read beforehand with the READ function from the standardized transfer area. It can also be used to query the number of operands in the statement.

Call

```
CALL SDF('STMT',area,error,stmt[,num])
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
STMT	char(4)	in	Function name: keyword STMT
area	char()	in	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
stmt	char(8)	out	Address of an 8-byte field into which the internal statement name will be written
num	integer	out	Number of operands in the statement read

STRU
Analyze data type and length of value introducing structure

The STRU function analyzes the operand value introducing a structure to determine its data type and length. The STRU call must be preceded by a TYPE call having the operand type structure (X'13') as its result. An operand value does not introduce a structure unless it was defined with STRUCTURE=*YES(FORM=*NORMAL) in the SDF-A statement ADD-VALUE.

Call

```
CALL SDF('STRU',area,error,pos,typ,lng[,lst])
```

Description of the parameter list

Parameter	**Data type (length)**	**Input/ output**	**Meaning**
STRU	char(4)	in	Function name: keyword STRU
area	char()	in	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
pos	integer	in	Position of the value introducing the structure
typ	integer	out	Data type of the operand value (for a description of the data types, see page 372ff)
lng	integer	out	Length of the operand value
lst	integer	in	Relevant only if the operand value is an element in a list: position of the structure in the list

TRNS Analyze statement

The TRNS function analyzes a statement in text format and translates it into the format for the standardized transfer area. The result of this analysis is stored in the standardized transfer area. For detailed information, see the description of the macro TRSTMT macro (page 614ff).

The TRNS function can also be used to create conversion descriptions for the READ function.

Call

```
CALL SDF('TRNS',area,error,stmt)
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
TRNS	char(4)	in	Function name: keyword TRNS
area	char()	inout	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
stmt	V-rec@	in	Address of a variable-length record containing the statement to be analyzed in text format

The TRSTMT macro is called with the following defaults:

```
STMT=*ALL
DIALOG=*ERROR
MESSAGE=*NO
PROT=*NO
INVAR=*NO
DEFAULT=*NO
ERROR=*NO
CALLID=*NO
EXECUTE=*NO
PROCMODE=*ANY
CCSNAME=*NO
```

TYPE
Analyze data type and length of operand

The TYPE function analyzes an operand at a given position to ascertain its data type and its length. The position corresponds to the position defined with the SDF-A statement ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...).

Call

```
CALL SDF('TYPE',area,error,pos,typ,lng[,lst])
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
TYPE	char(4)	in	Function name: keyword TYPE
area	char()	in	Buffer in which the standardized transfer area was stored (see INIT)
error	integer	out	Return code
pos	integer	in	Operand position (as defined by ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...))
typ	integer	out	Operand data type (for a description of the data types, see page 372ff)
lng	integer	out	Length of the operand
lst	integer	in	Relevant only if the operand is an element in a list: position of the operand in the list

WRRC
Set command return codes

The WRRC function enables a user program to save values it generates as command return codes; these are made available to the command processor as the official command return codes when the program is terminated and can then be used by other system functions (e.g. SDF-P). The function is based on the CMDRC macro (page 402ff).

Call

```
CALL SDF('WRRC',area,error)
```

Description of the parameter list

Parameter	Data type (length)	Input/ output	Meaning
WRRC	char(4)	in	Function name: keyword WRRC
area	char(9)	in	Area in which the 9-byte command return code is specified; must be aligned on a halfword boundary.
error	integer	out	Return code of the function

6.5.2.3 Examples

The following examples demonstrate the principles underlying SDF function calls in the programming languages COBOL and FORTRAN.

Use of the COBOL interface (program excerpt)

```
 .
  .
  DATA DIVISION.
  WORKING-STORAGE SECTION.
 *work variable
  01 SDF-DATA
      02 SDF-INIT PIC X(4) VALUE "INIT".
      02 SDF-READ PIC X(4) VALUE "READ".
      02 SDF-STMT PIC X(4) VALUE "STMT".
      02 SDF-TYPE PIC X(4) VALUE "TYPE".
      02 SDF-VAL  PIC X(4) VALUE "OPER".
      02 SDF-TYP  PIC 9(6) COMP.
      02 SDF-LNG  PIC 9(6) COMP.
      02 SDF-POS  PIC 9(6) COMP.
      02 SDF-LST  PIC 9(6) COMP.
```

```
          02 SDF-ERR  PIC S9(6) COMP.
          02 SDF-STMT PIC X(8).
          02 SDF-PROG PIC X(8).
      77 BUF PIC X(500).
      77 MAX PIC 9(6) COMP VALUE 500.
      01 VAL.
          02 FILLER PIC X OCCURS 1 TO 50 DEPENDING ON SDF-LNG.
      .
      .
     *
      PROCEDURE DIVISION.
     *
     *INIT
     *
         MOVE "TEST" TO SDF-PROG.
         CALL "SDF" USING SDF-INIT, BUF, SDF-ERR, MAX, SDF-PROG.
     *
     *READ
     *
         CALL "SDF" USING SDF-READ, BUF, SDF-ERR.
     *
     *STMT
     *
         CALL "SDF" USING SDF-STMT, BUF, SDF-ERR, SDF-STMT.
     *
     *TYPE
     *
         CALL "SDF" USING SDF-TYPE, BUF, SDF-ERR, SDF-POS,
                          SDF-TYP, SDF-LNG.
     *
     *VAL
     *
         CALL "SDF" USING SDF-VAL, BUF, SDF-ERR, SDF-POS, VAL,
                          SDF-LNG.
     .
     .
```

The program must be linked with the library SYSLIB.SDF.04x (e.g. 045 for SDF V4.5). This library includes, among other things, a COBOL copy element containing the description of the SDF declarations (type S member SDFCOPY).

Use of the FORTRAN interface (program excerpt)

```
.
.
C*****Work variable
      CHARACTER*4 SDF$INIT/'INIT'/
      CHARACTER*4 SDF$READ/'READ'/
      CHARACTER*4 SDF$STMT/'STMT'/
      CHARACTER*4 SDF$VAL/'OPER'/
      CHARACTER*4 SDF$TYPE/'TYPE'/
      INTEGER SDF$LNG,SDF$POS,SDF$LST,SDF$TYP,SDF$ERR
      CHARACTER*8 SDF$STMT,SDF$PROG
      CHARACTER*500 AREA,VAL*50
      INTEGER MTYP(2),MLNG(2)

      *
      *INIT
      *
          CALL SDF(SDF$INIT,AREA,SDF$ERR,500,'TEST␣␣␣␣')
      *
      *READ
      *
          CALL SDF(SDF$READ,AREA,SDF$ERR)
      *
      *STMT
      *
          CALL SDF(SDF$STMT,AREA,SDF$ERR,SDF$STMT)
      *
      *TYPE
      *
          CALL SDF(SDF$TYPE,AREA,SDF$ERR,SDF$POS,SDF$TYP,SDF$LNG)
      *
      *VAL
      *
          CALL SDF(SDF$VAL,AREA,SDF$ERR,SDF$POS,VAL,SDF$LNG)
      .
      .
```

The program must be linked with the library SYSLIB.SDF.04x (e.g. 045 for SDF V4.5). This library includes, among other things, a FORTRAN include element containing the description of the SDF declarations (type S member SDFINCL).

6.5.3 C interface

The conventions that apply to C for passing parameters to functions differ from those for COBOL and FORTRAN. Consequently, SDF also has a C interface which provides the same functionality as the general HLL interface. Functions in which optional parameters occur have been split into a number of C functions.

The C functions offer the functionality of the old macro calls RDSTMT, CORSTMT and TRSTMT that use the standardized transfer area in the old format (see section “Changes to the SDF program interface” on page 593ff).

If the extended functionality of the corresponding “new” macro calls CMDRST, CMDCST and CMDTST (e.g statement return code) are used for the standardized transfer area in the new format, then the Assembler interface must be used (see page 379ff).

6.5.3.1 Description of the C functions

The C functions are listed in alphabetical order in the overview and described in detail in the pages that follow (page 504ff). Those functions that support optional parameters and have therefore been split into several functions are described under a single function name. For example, the functions sdfrd, sdfrdmsg, sdfrdall, sdfrdpre and sdfrddef are all described in the section entitled ’sdfrd’ and labelled formats 1 to 5.

Results of functions

Every C function returns an integer as its result:

Result = 0:	Normal execution; no errors
Result > 0:	Error in the associated macro. The error code has been copied from register 15. The value of the error code corresponds to that of the macro error code.
Result < 0:	Error in the function call. The value is one of the following:

-1 :	Function unknown
-2 :	Not enough operands
-3 :	Last operand in structure reached
-4 :	Last operand in list reached
-5 :	Position is negative
-6 :	Operand is not of the type LIST
-7 :	Operand is not of the type STRUCTURE

Notes

In order to call a C function for SDF, the program requires the following include elements from the SYSLIB.SDF.04x library:

sdfc.h	(contains constant and type definitions)
sdfcext.h	(contains the external declarations of the functions)

Note that certain strings must be aligned on a halfword or word boundary. However, since variables of the data type ’char’ may be aligned only on the byte boundary, they must also be aligned with the aid of the malloc() function.

The variables for internal statement names must be 8 characters long and must end with “\0”. The predefined type STR8 from the include element sdfc.h can be used for this purpose.

An example of how to use the C functions begins on page 519.

6.5.3.2 Overview of C functions

sdfcbit	Set correction bit
sdfcmd	Execute a system command
sdfcor	Initiate a semantic error dialog
sdfinit	Initialize buffer
sdflev	Position on an operand array
sdfrc	Set command return code
sdfrd	Read and analyze a statement
sdfsta	Output information about syntax files
sdfstmt	Read statement name from the transfer area
sdfstv	Analyze data type and length of value-introducing structure
sdftr	Analyze a statement
sdftyp	Analyze data type and length of operands
sdfval	Read operand value from the transfer area

sdfcbit
Set correction bit

The sdfcbit function sets a correction bit for a given operand. When this correction bit is set, the operand in question is underscored in the error dialog for a statement, if it was invalid.

Format:

```
int sdfcbit (char *area, int pos);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit)
int pos	Position of the operand whose correction bit is to be set

Result

The function returns an integer as its result (see page 503).

sdfcmd
Execute system command

The sdfcmd function calls a command without exiting the program mode. The sdfcmd function is based on the CMD macro described in the “Executive Macros” manual [8].

Format:

```
int sdfcmd (char *area, int sysout, int dialog, int list, char *cmdrc, char *buff);
```

Description of the parameters

char *area	Pointer to a string containing the command to be executed. SDF internally converts the string into a variable-length record.
int sysout	Specifies whether the log is to be output to SYSOUT and/or the log buffer: 0 : SYSOUT = NO (SYSOUT only) 1 : SYSOUT = YES (SYSOUT and buffer)
int dialog	Specifies whether an error dialog is to be initiated if syntactical errors are discovered: 0 : DIALOG = NO 1 : DIALOG = YES
int list	Specifies whether the area string contains more than one command with semicolons as separators 0 : LIST = NO (only one command in the string) 1 : LIST = YES (more than one command in the string)
char *cmdrc	Pointer to a string (9 bytes) in which the command processor is to store the command return code. cmdrc must be aligned on the halfword boundary.
char *buff	Pointer to a string into which the log is output (BUFMOD=SHORT, see the CMD macro).

Result

The function returns an integer as its result (see page 503).

i A version 3 parameter list (VER=3, PARMOD=31 - see CMD macro in the „Executive Macros“ manual [6] is generated.

sdfcor
Initiate semantic error dialog

The sdfcor function causes SDF to initiate a dialog with the user, providing the opportunity to correct a semantic error in a statement. Immediately beforehand, SDF analyzes the statement and transfers it to the program as syntactically correct. For detailed information, see the description of the CORSTMT macro (page 601ff).

Format:

```
int sdfcor (char *area, char *msg);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
char *msg	Pointer to a message to be output in the error dialog. SDF internally converts the string into a variable length record no more than 400 characters long. If a NULL pointer is specified, no message is displayed.

Result

The function returns an integer as its result (see page 503).

sdfinit
Initialize buffer

The sdfinit function initializes the buffer in which the standardized transfer area is stored. The internal name of the program must be specified as it is defined in the syntax file.

The sdfinit function must be called once before all other SDF-C functions that use the standardized transfer area.

Format:

```
int sdfinit (char *area, int lng, STR8 progname);
```

Description of the parameters

char *area	Pointer to the buffer where the standardized transfer area will be placed; must be aligned on the word boundary.
int lng	Length of the buffer for the standardized transfer area.
STR8 progname	Internal program name (as defined in syntax file), 8 characters long with null ('\0') at end.

Result

The function returns an integer as its result (see page 503).

sdflev
Position on operand array

The sdflev function enables the user to position on an operand array belonging to a structure description (see page 365ff). All functions that access data in the standardized transfer area always refer to the current operand array.

When the sdfinit function is called, the system positions on the operand array of the highest level, in other words on the operand array whose operand was defined in the syntax file with the SDF-A statement ADD-OPERAND ...,RESULT-OPERAND-LEVEL=1.

The sdflev function always refers to the current operand array. This can be the operand array of a structure description, if the sdflev function has already been called beforehand.

The structure description can also be an element in a list (the operand type check with sdftyp returns 'list' as the operand type). In this case, the user must specify both the position of the list in the current operand array and the position of the structure description in the list. If the structure description is not an element in a list, the user needs to specify only the position of the structure description in the operand array.

To return to the operand array of the highest level, enter 0 for the position.

Since the function has an optional parameter, there are 2 different formats.

Format 1:

```
int sdflev (char *area, int pos);
```

Format 2:

```
int sdflevls (char *area, int pos, int lst);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
int pos	Position of the structure description in the current operand array or position of the list containing the structure description. pos=0 positions on the operand array of the highest level.
int lst	Relevant only if the structure description is an element in a list; specifies the position of the structure description in the list.

Result

The function returns an integer as its result (see page 503).

sdfrc
Set command return code

The sdfrc function enables a user program to save values it generates as command return codes; these are made available to the command processor as the official command return codes when the program is terminated and can then be used by other system functions such as SDF-P (see the “Commands” manual [4]). The sdfrc function is based on the CMDRC macro (see page 402ff).

Format:

```
int sdfrc (char *area);
```

Description of the parameters

char *area	Pointer to a string containing the command return code, must be aligned on a halfword boundary. The predefined structure type SDF_COMMAND_RC should be used (see example on page 519ff).

Result

The function returns an integer as its result (see page 503).

sdfrd
Read and analyze statement

The sdfrd function causes SDF

- to read a program statement from SYSSTMT. (The assignment for the SYSDTA system file also applies to SYSSTMT. The rules governing continuation lines, continuation characters, logging and comments in SYSSTMT are the same as those that apply to SYSCMD.)
- analyze the statement read in and
- transfer the result of this analysis to the program.

This presupposes that an activated syntax file contains the definition of the program and its statements. For details, see the description of the RDSTMT macro (page 606ff).

Since the function has a number of optional parameters, there are 5 different formats.

Format 1:

```
int sdfrd (char *area);
```

Format 2:

```
int sdfrdmsg (char *area, char *msg);
```

Format 3:

```
int sdfrdall (char *area, char *msg, char *allow);
```

Format 4:

```
int sdfrdpre (char *area, char *msg, char *allow, STR8 prefer);
```

Format 5:

```
int sdfrddef  (char *area, char *msg, char *allow, STR8 prefer, char *def1,
               char *def2, char *def3, char *def4, char *def5);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
char *msg	Pointer to a message to be output before the statement is read. SDF internally converts the string into a variable-length record. The string must be no more than 400 characters long. If a NULL pointer is specified, no message is displayed.
char *allow	Pointer to the list of valid statements (must be aligned on word boundary).

Format of list of valid statements:

N	statement $_1$	statement $_2$	.....	statement $_n$

N	(2 bytes) :	number of statements in the list
statement$_x$	(8 bytes) :	internal name of a valid statement

STR8 prefe — Only in guided dialog: internal name of the statement expected next, 8 characters long, with zero ('\0') as the last character.

char *def1,*def2,*def3,*def4,*def5
Pointers to up to 5 conversion descriptions with which default values of operands can be replaced by values generated dynamically by the program. A value must be specified for each of the 5 pointers. Pointers not needed must be specified as NULL pointers. All pointers after the first NULL pointer are ignored.

Result

The function returns an integer as its result (see page 503).

Read and convert default values

With the aid of conversion descriptions, default values of operands defined in the syntax file with ADD-VALUE ...,VALUE=<c-string>...(OVERWRITE-POSSIBLE=*YES) or ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES can be replaced by values generated dynamically by the program. Only one conversion description can be specified per statement. The conversion description contains the internal statement name and the information defining which operand values can be replaced and how. A conversion description can be generated either by an sdfrd call executed beforehand or by an sdftr call in which the new values are entered. This description has the format of a standardized transfer area.

sdfsta
Output information about syntax files

The sdfsta function outputs information concerning activated syntax files and the specifications applying to command/statement input and processing.
For a description of the transfer area for the information, see the CMDSTA macro (page 425ff).

Since the function has an optional parameter, there are 2 different formats.

If format 1 is called, the short form is always output, i.e.:

activated syntax files, but no subsystem syntax files, plus the current options for the input and processing of commands and statements. The predefined structure type SDF_STATUS_SHORT should be used for area (see example on page 519ff).

If format 2 is called, the long form is output, i.e.:

activated syntax files (including subsystem syntax files) and the current options for the input and processing of commands and statements. The scope of information about subsystem syntax files depends on the amount of space available in the transfer area. If the subsystem information overflows the transfer area, an error code to the appropriate effect is output.

Format 1:

```
int sdfsta (char *area);
```

Format 2:

```
int sdfstal (char *area, int lng);
```

Description of the parameters

char *area	Pointer to the transfer area in which the information is to be stored by SDF; must be aligned on a halfword boundary.
int lng	Length of transfer area for FORM=LONG (see CMDSTA macro). The length must be greater than SDF_STATUS_SIZE_SHORT (SDF_STATUS_SIZE_SHORT=530)

Result

The function returns an integer as its result (see page 503).

sdfstmt
Read statement name from transfer area

The sdfstmt function reads the internal name of a statement read beforehand with the sdfrd function from the standardized transfer area. It can also be used to query the number of operands in the statement.

Since the function has an optional parameter, there are 2 different formats.

Format 1:

```
int sdfstmt (char *area, STR8 stmt);
```

Format 2:

```
int sdfstmtn (char *area, STR8 stmt, int *num);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was stored (see sdfinit).
STR8 stmt	String for the internal name of the statement, 8 characters long, with zero ('\0') at the end.
int *num	Pointer to the number of operands in the statement read.

Result

The function returns an integer as its result (see page 503).

sdfstv
Analyze data type and length of value introducing structure

The sdfstv function analyzes the operand value introducing a structure to determine its data type and length. The sdfstv function must be preceded by a sdftyp call returning the operand type structure (19) as its result. An operand value does not introduce a structure unless it was defined with STRUCTURE=*YES(FORM=*NORMAL) in the SDF-A statement ADD-VALUE.

Since the function has an optional parameter, there are 2 different formats.

Format 1:

```
int sdfstv (char *area, int pos, int *typ, int *lng);
```

Format 2:

```
int sdfstvls (char *area, int pos, int *typ, int *lng, int lst);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
int pos	Position of the value of the operand introducing the structure.
int *typ	Pointer to a value containing the type of the operand value (for description of types, see page 372ff).
int *lng	Pointer to the length of the operand value.
int lst	Relevant only if the operand value is an element in a list and specifies the position of the structure in the list.

Result

The function returns an integer as its result (see page 503).

sdftr
Analyze statement

The sdftr function analyzes a statement in text format and translates it into the format for the standardized transfer area. The result of this analysis is stored in the standardized transfer area. For detailed information, see the description of the TRSTMT macro (page 614ff).

The sdftr function can also be used to create conversion descriptions for the sdfrddef function.

Since the function has an optional parameter, there are 2 different formats.

Format 1:

```
int sdftr (char *area, char *stmt);
```

Format 2:

```
int sdftrall (char *area, char *stmt, char *allow);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was placed (see sdfinit).
char *stmt	Pointer to a string containing the statement to be analyzed in text format. SDF internally converts the string into a variable-length record.
char *allow	Pointer to a list of statements permissible as inputs (must be aligned on a word boundary).

Format of the list of permissible statements:

N	statement $_1$	statement $_2$	. . . . .	statement $_n$

N	(2 bytes):	number of statements in the list
statement$_x$	(8 bytes) :	internal name of a valid statement

Result

The function returns an integer as its result (see page 503).

sdftyp
Analyze data type and length of operands

The sdftyp function analyzes an operand at a given position to ascertain its data type and its length. The position corresponds to the position defined with the SDF-A statement ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...).

Format 1:

```
int sdftyp (char *area, int pos, int *typ, int *lng);
```

Format 2:

```
int sdftypls (char *area, int pos, int *typ, int *lng, int lst);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
int pos	Position of the operand (as specified in ADD-OPERAND ..., RESULT-OPERAND-NAME=*POS(...)).
int *typ	Pointer to a value containing the type of the operand (see page 372ff).
int *lng	Pointer to the length of the operand.
int lst	Relevant only if the operand is an element in a list and specifies the position of the operand in the list.

Result

The function returns an integer as its result (see page 503).

sdfval
Read operand value from transfer area

The sdfval function reads the value of an operand at a certain position from the standardized transfer area. The sdfval call must be preceded by an sdftyp or sdfstv call defining the operand type and the operand length.

Since the function has an optional parameter, there are 2 different formats.

Format 1:

```
int sdfval (char *area, int pos, char *val, int lng);
```

Format 2:

```
int sdfvalls (char *area, int pos, char *val, int lng, int lst);
```

Description of the parameters

char *area	Pointer to the buffer in which the standardized transfer area was created (see sdfinit).
int pos	Position of the operand in the current operand array.
char *val	Pointer to a string in which the value is to be placed.
int lng	Length of the val string, must be greater than or equal to the value returned beforehand for the lng parameter by sdfstv or sdftyp.
int lst	Relevant only if the operand is an element in a list and specifies the position of the operand in the list.

Result

The function returns an integer as its result (see page 503).

6.5.3.3 Example of the use of the C functions

In section “Example: Program for copying files” on page 104 an Assembler program KOP for copying SAM and ISAM files was created. The following example is intended to show how to achieve similar results with a C program. In addition to the SDF standard statements, the program KOPC has the following statement:

COPY-FILE

FROM-FILE = <filename 1..54>
,TO-FILE = <filename 1..54 without-gen-vers>(...)
    <filename 1..54 without-gen-vers>(...)
        | **ACCESS-METHOD** = **_*SAME_** / ***ISAM**(...) / ***SAM**
        | ***ISAM**(...)
        | | **KEY-LENGTH** = **_*STD_** / <integer 1..50>
        | **,RECORD-SIZE** = **_*SAME_** / ***VARIABLE** / <integer 1..2048>
,PASSWORD = **_*NONE_** / <c-string 1..4> / ***SECRET-PROMPT**

Defining the program in the user syntax file

The KOPC program is defined in the syntax file SDF.USER.SYNTAX. This is done by passing the same statements to SDF-A as are given on page 105ff, except for statements 4 and 5, which are entered as follows:

```
//add-prog kopc ---------------------------------------------------- (4)
//add-stmt name=copy-file,prog=kopc,intern-name=copyfi,stmt-vers=1 ------ (5)
```

Generating the program

Excerpts from the KOPC program are reproduced on the following pages.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*******************************/
/*   SDF includes              */
/*******************************/
#include "sdfc.h"
#include "sdfcext.h"

/*******************************/
/*   Constants & types         */
/*******************************/
#define OUTPUTL 200
typedef char STR8[8+1];         /* string of 8 chars, null terminated */
typedef char FILENAME[54+1];
typedef char STR4[4+1];

main () {

/*******************************/
/*   data                      */
/*******************************/

/* SDF interface data */

char *output;
SDF_STATUS_SHORT *status;
char *usfname;
STR8 name;
int sdf_err;
int sdf_op_l,sdf_op_t;
char val_char[16];

/* program data */

FILENAME from_file,to_file;
int  access;
int  keyl;
int  recs_var;
int  recs;
int  passwd_given;
STR4 passwd;
int nocorr;
```

```
/* SDF interface data  allocation */

if ( (output=malloc(OUTPUTL)) == NULL ) exit(2);
if ( (status=malloc(SDF_STATUS_SIZE_SHORT)) == NULL )exit(2);

/*******************************/
/*   check syntax file         */
/*******************************/

sdf_err = sdfsta((char *)status);
if (sdf_err) exit(3);
usfname = strchr((status)->sdf_user_sf_name,'.') + 1;
if (strncmp( usfname,
         "SDF.KOP.SYNTAX",14))
                     exit(3);

/*******************************/
/*   initialization            */
/*******************************/

sdf_err = sdfinit (output,OUTPUTL, "KOP     ");
if (sdf_err) exit(4);

/*******************************/
/*   read first statement      */
/*******************************/

sdf_err = sdfrd (output);
switch (sdf_err) {
    case SDF_END:
    case SDF_EOF: return(0);
    case SDF_OK:  break;
    default:      exit(5);
    }

/*******************************/
/*   analyse loop              */
/*******************************/

do {
    /* * * * * * * * * * * * * * * * */
    /*   transfer area analysis      */
    /* * * * * * * * * * * * * * * * */

    /* statement name */

    sdf_err = sdfstmt (output,name);
```

```
if (sdf_err) exit(6);
if (strncmp (name,"COPYFI  ",8)) exit(7);

/* operand from file */

sdf_err = sdftyp (output,1,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(8);
sdf_err = sdfval (output,1,from_file,sdf_op_l);
if (sdf_err) exit(9);
from_file[sdf_op_l] = '\0';

/* operand password */

passwd_given = 0;
sdf_err = sdftyp (output,6,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(10);
if (sdf_op_t == SDF_CSTR) {
    passwd_given = 1;
    sdf_err = sdfval (output,6,passwd,sdf_op_l);
    if (sdf_err) exit(11);
    passwd[sdf_op_l] = '\0';
}

/* operand to file */

sdf_err = sdftyp (output,2,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(13);
sdf_err = sdfval (output,2,to_file,sdf_op_l);
if (sdf_err) exit(14);
to_file[sdf_op_l] = '\0';

/* sub operand access-method */

sdf_err = sdftyp (output,3,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(15);
sdf_err = sdfval (output,3,val_char,sdf_op_l);
if (sdf_err) exit(16);
if (sdf_op_l == 3)
    access = 1;      /* assume SAM is 1 */
else if ( val_char[0] == 'I' )
    access = 2;      /* assume ISAM is 2 */
else
    access = 0;      /* assume SAME is 0 */


/* sub sub operand key length */
```

```
if (access == 2) {   /* only for ACCESS-METHOD=ISAM */
    sdf_err = sdftyp (output,4,&sdf_op_t,&sdf_op_l);
    if (sdf_err) exit(17);
    switch (sdf_op_t) {
        case (SDF_NOTY): return(17); /* operand not occupied */
        case (SDF_INTG):
          sdf_err = sdfval (output,4,(char *)&keyl,sdf_op_l);
          if (sdf_err) exit(18);
          break;
        default: keyl = 8;
                 break;
    }
}

/* sub operand record-size */

recs = 0;
recs_var = 0;
sdf_err = sdftyp (output,5,&sdf_op_t,&sdf_op_l);
if (sdf_err) exit(21);
if (sdf_op_t == SDF_INTG) {
    sdf_err = sdfval (output,5,(char *)&recs,sdf_op_l);
    if (sdf_err) exit(22);
}
else {
    sdf_err = sdfval (output,5,val_char,sdf_op_l);
    val_char[sdf_op_l] = '\0';
    if (sdf_err) exit(23);
    if (!strcmp (val_char,"SAME"))
        recs = 0;    /* assume 0 is same */
    else if (!strcmp (val_char,"VARIABLE"))
        recs_var = 1;
    else exit(24);
}

/* * * * * * * * * * * * * * * * * */
/*      actual processing        */
/* * * * * * * * * * * * * * * * * */

/* semantic validation of input parameter and
   correction if necessary */
/* example : open of input file
   call DMS
   IF error
      sdf_err = sdfcbit (output,1);
      sdf_err = sdfcor (output,"Input file can not be opened");
      if (!sdf_err) continue; restart analysis       */
      /* example : assume password 'AAAA' wrong : prompt for
         correction */
        nocorr = 0;
        if (!strcmp (passwd,"AAAA")) {
           sdf_err = sdfcbit (output,6);
           sdf_err = sdfcor (output,"Invalid password");
           if (!sdf_err) continue; /* restart analysis */
           else nocorr = 1;
        }


      if (!nocorr) {
        /* copy processing .... */
        printf ("from file %s\n",from_file);
        printf ("to file %s\n",to_file);
        printf ("access %d \n",access);
        if (access == 2) {
           printf ("key 1 %d\n",key1);
        }
        if (!recs_var)
           printf ("recs %d \n",recs);
        else
           printf ("rec var\n");
        if (passwd_given)
           printf ("password %s\n",passwd);
      }

     /* * * * * * * * * * * * * * * * * */
     /*      read next statement      */
     /* * * * * * * * * * * * * * * * * */

     sdf_err = sdfrd (output);
     switch (sdf_err) {
         case SDF_END:
         case SDF_EOF: return(0);
         case SDF_OK:  break;
         default:      exit(3);
         }
 } while (1);   /* endless loop */

 }
```

Testing the program

The source program KOPC shown above has been compiled and linked and can now be tested.

```
/set-logon-parameters sdfusr,... ──────────────────────────────────── (1)
 .
 .
/modify-sdf-options syntax-file=*add(sdf.kop.syntax) ──────────────── (2)
/start-exe *lib-elem(lib=kop.lib,elem=kopc) ───────────────────────── (3)
%  BLS0524 LLM 'KOPC', VERSION ' ' OF '2001-10-10 14:00:22' LOADED
%  BLS0551 COPYRIGHT (C) FUJITSU SIEMENS COMPUTERS GMBH 2001. ALL RIGHTS
RESERVED
%// ... ───────────────────────────────────────────────────────────── (4)
 .
 .
```

1. A task is started under the user ID SDFUSR.
2. The user syntax file SDF.KOP.SYNTAX, in which statements for the program KOPC are defined, is activated.
3. The program KOPC is started. The command used here, START-EXECUTABLE-PROGRAM, is available in BLSSERV V2.3 and higher (the START-PROGRAM command with RUN-MODE=*ADVANCED is to be used if necessary).
4. The program KOPC expects the entry of a statement. This can be one of the SDF standard statements or the COPY-FILE statement. The KOPC program can be tested in the same way as the KOP program (cf. page 120ff, from step 8 onwards).

7 SDF-SIM

When defining commands and statements with SDF-A, there is no way of getting by without testing their syntax. SDF-SIM (System Dialog Facility-Simulator) provides you with a tool that enables you to carry out this essential syntax test efficiently and without risk in a test environment of your own choice.

SDF-SIM has a simple interface which makes it easy to use. It supports a variety of useful auxiliary features, such as:

– flexible test environments

– simulation of procedure or batch mode with realistic spin-off behavior

– entry of job variables

– support of privileges

– differentiated output options for the simulation result.

Defining the simulation environment

SDF-SIM can be used to simulate an environment in which the syntax of commands and statements is analyzed. This environment is not dependent on the BS2000 system environment and a user who is not equipped with any special privileges can exert a considerably greater influence on it than is normally possible (e.g. as regards assigning system and group syntax files).

The following SDF options can be defined for the environment:

– batch or interactive mode

– procedure mode

– guided dialog or unguided dialog

– continuation mode (CONTINUATION)

– logging (INPUT-FORM, ACCEPTED-FORM, INVARIANT-FORM)

– privileges belonging to the task.

Testing command and statement syntax

The syntax of the commands or statements, which are described in a specific SDF syntax file hierarchy (system, group and user syntax file), is tested on TU level. This enables the user to check how SDF edits the syntax of an entered command or statement.

The procedures, programs or command servers are not required for the syntax test as SDF-SIM only accesses the syntax definitions in the defined syntax file hierarchy. This can be seen in Figure 16.

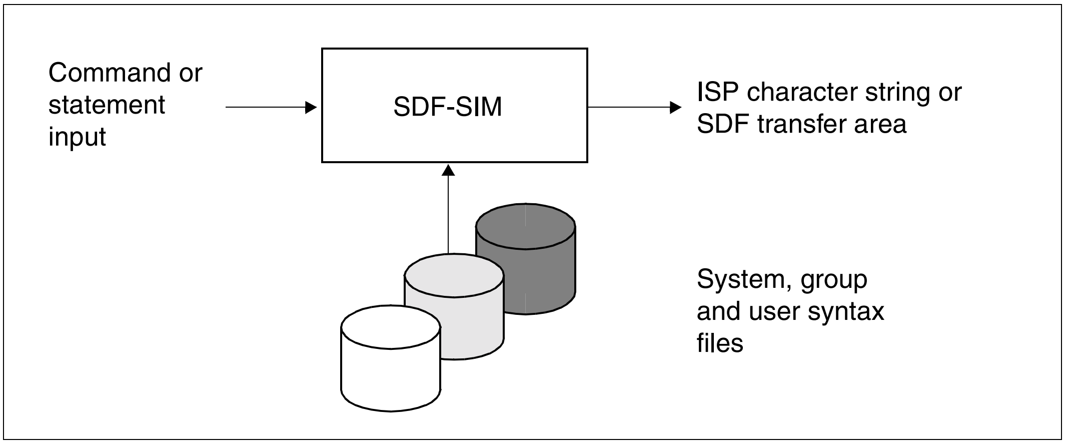

Command or statement input
SDF-SIM
ISP character string or SDF transfer area
System, group and user syntax files

Figure 16: Testing the command and statement syntax

Outputting information on the command or statement

The following information is output for each simulated command or statement:

- log of the command or statement
- interface type for the commands (ASS/ISL/SPL/PROCEDURE)
- name of the entry point for the routine executing the command (for TPR commands only)
- output format:
  - character string if a TPR command is mapped onto an old ISP command or if the command is implemented by a procedure
  - SDF transfer area in the case of new SDF commands or statements
  - in the case of commands implemented by a procedure, the associated procedure call is also output. This enables you to check how the procedure parameters are passed to the procedure (see the example on page 566).

7.1 Working with SDF-SIM

Components for installing SDF-SIM

The statements that can be passed to SDF-SIM are defined in the syntax file SYSSDF.SDF-SIM.045.

The following files are supplied as part of SDF-SIM:

SYSLNK.SDF-SIM.045	Library containing the module SDF-SIM.
SYSSDF.SDF-SIM.045	Syntax file containing the SDF-SIM statements and the START-SDF-SIM command.
SYSMES.SDF-SIM.045	SDF-SIM message file

Coexistence between different SDF-SIM versions

The START-SDF-SIM command (see page 530) invokes the SDF-SIM version that was specified for the user´s system by the system administrator. This will usually be the latest SDF-SIM version. It is, however, also possible to call an older SDF-SIM version if desired. The following steps are required for this purpose:

1. Create a user syntax file with SDF-A and copy all SDF-SIM statements of the desired SDF-SIM version into it (using the COPY statement). The required SDF-SIM statements can be found in the system syntax file that was supplied with the desired SDF-SIM version (SYSSDF.SDF-SIM.<version>).
2. Activate the created user syntax file with the MODIFY-SDF-OPTIONS command (see the “Commands” manual [4]).
3. Call the desired SDF-SIM version with the following command:

   ```
   /START-EXE *LIB-ELEM(LIBRARY=$.SYSLNK.SDF-SIM.vvv,ELEMENT=SDF-SIM) 1
   ```

   <version> is the desired version number, e.g. 041 for SDF-SIM V4.1.

   or

   ```
   /START-SDF-SIM VERSION=<version>
   ```

   <version> is the desired version number, e.g. 041 for SDF-SIM V4.1.

[1] The command used here, START-EXECUTABLE-PROGRAM, is available in BLSSERV V2.3 and higher (the command START-PROGRAM is to be used if necessary).

Starting SDF-SIM

SDF-SIM is started by means of the following command:

START-SDF-SIM	Abbreviation: **SDF-SIM**
VERSION = ***STD** / <product-version>	
,MONJV = ***NONE** / <filename 1..54 without-gen-vers>	
,CPU-LIMIT = ***JOB-REST** / <integer 1..32767 *seconds*>	

VERSION=
Allows the user to select the desired SDF-SIM version if multiple versions of SDF-SIM were installed with IMON. If the version is specified within single quotes, it may be preceded by the letter C (C-STRING syntax).
If the product was not installed using IMON or if the specified version does not exist, VERSION=*STD applies (see also "Coexistence between different SDF-SIM versions" on page 529).

VERSION=*STD
Calls the SDF-SIM version with the highest version number.

VERSION=<product-version>
Specifies the SDF-SIM version in the format m.n[a[so]] (see also "product-version" on page 15).

MONJV =
Specifies a monitoring job variable to monitor the SDF-SIM run.

MONJV = *NONE
No monitoring job variable is used.

MONJV = <filename 1..54 without-gen-vers>
Name of the job variable to be used.

CPU-LIMIT =
Maximum CPU time in seconds which the program may consume for execution.

CPU-LIMIT = *JOB-REST
The remaining CPU time available is to be used for the job.

CPU-LIMIT = <integer 1..32767 *seconds*>
Only as much time as is specified is to be used.

You can then enter the statements for preparing the simulation (DEFINE-TEST-OBJECT, DEFINE-ENVIRONMENT).

The simulation is started after entry of the START-SIMULATION statement. SDF-SIM first of all displays the current syntax file hierarchy, then it displays an asterisk (*) prompting you to enter the command or statement to be simulated.

Terminating SDF-SIM

During simulation ('*' prompting), SDF-SIM can be terminated by entering the string '/*'. During the preparations for simulation ('%//' prompting), this can also be done with the aid of the SDF standard statement END.

Features of a simulation run

The following example shows the basic features of a simulation run.

```
/START-SDF-SIM
%// ...           }
%// ...           }—————  Statements for preparing simulation
%// ...           }
%//START-SIMULATION —————  Starting simulation
*  ...            }
*  ...            }—————  Commands / statements to be simulated
*  ...            }
*/*
```

The statements for preparing simulation can also be specified more than once. In such cases, the settings made with the last statement of the same sort apply.

7.1.1 Command environment

SDF-SIM checks the command defined in the active syntax file hierarchy for syntax errors. This check is conducted within an environment defined by means of the DEFINE-ENVIRONMENT statement.

Simulation preparation

If a parameter file (*STD or <filename>) is specified in the PARAMETER-FILE operand, the names of the system syntax file and the group syntax file assigned to the profile ID of TSOS (SYS-TSOS) are read from the parameter file and activated for the simulation.
If the parameter file contains an error or the syntax files are invalid, the system syntax file and the TSOS group syntax file with the standard names are used.

If the name of the system, group or user syntax file is entered and an error occurs during activation of the syntax file, an error message is output. In the case of an error in the system or group syntax file, the syntax files with the standard names are activated. If there is an error in the user syntax file, no user syntax file is activated. At least one system or group syntax file must be activated, otherwise simulation is impossible as there is no global information available. If *STD is entered for the system syntax file, only the current basic system syntax file is activated. If *CURRENT is entered as the system syntax file, then the current basic system syntax file and the current subsystem syntax file are activated.

Simulation

Once simulation has been started, the command to be simulated can be entered in guided or unguided dialog. This is also contingent upon the specification in the MODIFY-SDF-OPTIONS command (see the example on page 556). Both in simulated procedure mode and in batch mode (statement DEFINE- ENVIRONMENT PROC-MODE=*YES or TASK-TYPE=*BATCH), the command entered must be preceded by a slash (/). The commands may also be entered in lowercase letters (see the example on page 561).

Output

A log is output after the syntax analysis. This log contains the name of the entry point, the type of the programming language interface, the generated character string (ISP format) or the standardized transfer area for commands defined by way of the SDF-A statement ADD-CMD ...,IMPLEMENTOR=*TPR(...,CMD-INTERFACE=*NEW/*TRANSFER-AREA...).

Replacing job variables

Job variables are replaced in SDF-SIM in exactly the same way as when simulation is not being performed (see the example on page 560). Procedure parameters, however, cannot be replaced.

Special SDF commands

The following SDF commands can be used if they are contained in at least one of the syntax files from the syntax file hierarchy. These commands are not simulated, they are really executed by SDF-SIM.

These commands include:

- MODIFY-SDF-OPTIONS:
  This activates and deactivates a user syntax file and modifies the SDF options for simulation (see page 551ff).
- SHOW-SDF-OPTIONS:
  This displays the active syntax files and the SDF options set for simulation.
- SHOW-INPUT-HISTORY:
  This shows a list of preceding inputs.
- RESTORE-SDF-INPUT:
  This restores the input.
- SHOW-INPUT-DEFAULTS:
  This displays task-specific default values.
- RESET-INPUT-DEFAULTS:
  This resets task-specific default values.
- WRITE-TEXT:
  This displays text that has been entered.
- SHOW-SYNTAX-VERSIONS:
  This displays the names and versions of the syntax files of individual products contained in currently active group and system syntax files.
- MODIFY-SDF-PARAMETERS:
  This modifies or creates a parameter file for SDF.
  This command can only be executed if the privilege *TSOS or *ALL has been specified in the DEFINE-ENVIRONMENT statement. In contexts other than simulation, it is reserved for system administration.
- SHOW-SDF-PARAMETERS:
  This displays the contents of the SDF parameter file.
  This command can only be executed if the privilege *TSOS or *ALL has been specified in the DEFINE-ENVIRONMENT statement. In contexts other than simulation, it is reserved for system administration.

Information output by SDF-SIM

SDF-SIM outputs the following information:

- the log; SDF supplies this in the form defined in the LOGGING operand of the MODIFY-SDF-OPTIONS command
- the name of the entry point into the TPR routine (not if the command was defined with ADD-CMD ...,IMPLEMENTOR=*PROCEDURE (see SDF-A))
- the interface type: ASS/SPL/ISL/PROCEDURE
- the output supplied by the command server, namely either:
  - a character string, if a command is mapped onto an old ISP command or if the command is implemented by way of a procedure, or
  - a standardized transfer area of not longer than 2042 bytes.

In the case of commands implemented by way of a procedure, the generated procedure call is also output. For commands which use the new format of the standardized transfer area (as of SDF V4.1), the version of the transfer area and command area are also displayed.

Example

Command implemented by way of a procedure (defined with ADD-CMD ...,IMPL=*PROCEDURE..., see SDF-A page 133ff)

```
/x-write 'date 2001-12-31'
(IN)      /x-write 'date 2001-12-31'

ENTRY : CLICOLD
INTERFACE : ISL
STRUCTURE-FORM
GENERATED CALL COMMAND :
/CALL-PROCEDURE $USER1.PROC.TEXT.1,(TEXT='date 2001-12-31')
```

7.1.2 Statement environment

SDF-SIM checks statements defined in the active syntax file hierarchy for syntax errors. The internal name of the program to which the statements to be simulated belong as well as the format of the standardized transfer area used for the statements must be specified in the preparatory phase for simulation (see DEFINE-TEST-OBJECT TEST-MODE=*STMT(PROGRAM-NAME=...,LAYOUT=<u>*OLD</u>/*NEW).

The statements are simulated in the environment defined by means of DEFINE-ENVIRONMENT. The MODIFY-SDF-OPTIONS statement can be used to influence simulation in the same way as in the command environment (guided or unguided dialog, form of logging, etc.).

On completion of the syntax analysis, a log and the standardized transfer area are output, if so requested by the user.

Information output by SDF-SIM

SDF-SIM outputs the following information:

- the log of the statement in the form defined in the LOGGING operand of the MODIFY-SDF-OPTIONS statement
- the standardized transfer area (a detailed example can be found on page 567).

  The standardized transfer area is displayed as follows if:

  – the operand DISPLAY=*LONG is set in the DEFINE-ENVIRONMENT statement, or

  – the message
  ```
  %  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
  ```
  is answered with “LONG”.

  If DISPLAY=*SHORT is specified, only the first part (the contents of the transfer area in hexadecimal form) is output. The structure description is only output if DISPLAY=*LONG is specified. The internal name of the statement is displayed. The structure description of the statement must not be longer than 15000 bytes.

Example

```
 .
 .
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
*long

     *** TRANSFER AREA ***
     ~~~~~~~~~~~~~~~~~~~~~
                  x x  x x x x
     (0FD540) 07FAC3D9 C6C9D3C5 40400000 00000000 00000020 ...
     (0FD5C0) 00000000 00000000 00000000 00000000 00008014 ...
     (0FD600) 00000000 00000000 00000000 00000000 ...
     (0FD6E0) 00000000 00000000 80160005 ...
      .
      .
     *** STRUCTURED DESCRIPTION ***
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     INTERNAL NAME OF THE CMD/STMT : xxxxxxxx
     VERSION OF THE TRANSFER AREA  : x   }  (only for new format of standardized
     VERSION OF THE CMD/STMT       : x   }  transfer area (as of SDF V4.1))
     MAXIMUM NUMBER OF OPERANDS    : xx
     OP( 1) :
      - TYPE   : KEYWORD
      - LENGTH :   xx
      - VALUE  : xxxxxx

     OP( 2) :
      - TYPE   : STRUCTURE
            VALUE INTRODUCING THE STRUCTURE:
           - TYPE   : xxxx
           - LENGTH :   xx
           - VALUE  : xxxxxxx
           MAXIMUM NUMBER OF OPERANDS   :    xx
           OP( 1) :
            - TYPE   : xxxxxx
            - LENGTH :  ....
            - VALUE  :  ...
           OP( 2) :     ....
       .
       .

     OP( 3) :
      - TYPE   : LIST
      - VAL(1) : - TYPE   : xxxx
                 - LENGTH : xxxxxxx
                 - VALUE  : xxxxxxx
      - VAL(2) : .........
```

```
OP( 4) :
 - TYPE   : OR_LIST
 - LENGTH : xxxx
 - VALUE  : xxxxx
 .
 .
```

The following values are possible for TYPE:

STRUCTURE	: Structure
LIST	: List
OR_LIST	: OR list
X_STR	: Hexadecimal string (x-string)
ALPHA_NAME	: Alphanumeric name (alphanum-name)
NAME	: Name (name)
STRUCT_NAME	: Structured name (structured-name)
COM_R	: Command rest (command-rest)
C_STR	: Character string (c-string)
TEXT	: Text (text)
INT	: Integer (integer)
KEYW	: Keyword (KEYWORD)
KEYW_NUM	: Keyword number (reserved for internal use)
F_FILENAME	: Fully qualified file name (filename)
P_FILENAME	: Partially qualified file name (partial-filename)
TIME	: Time (time)
DATE	: Date (date)
CAT_ID	: Catalog identifier (cat-id)
REAL	: real
LABEL	: Label (label)
STAR-ALPHA	: star-alphanumeric
COMPOSED_N	: Composed name (composed-name)
INPUT_TEXT	: Input text (input-text)
VSN	: Volume serial number (vsn)
X_TEXT	: X text (x-text)
FIXED	: Fixed-point number (fixed)
DEVICE	: Device (device)
POSIX_PATHN	: POSIX pathname (posix-pathname)
POSIX_FILEN	: POSIX file name (posix-filename)
PRODUCT_V	: Product version (product-version)

Special SDF statements

The following statements are SDF standard statements and are therefore defined for all programs if the activated syntax file was created correctly. These statements are not simulated by SDF-SIM; they are actually executed. As of SDF V4.1 the SDF standard statements in the syntax file are defined by SDF. They are no longer assigned to the SDF-U syntax file and cannot be copied into the syntax file of the application program. Therefore, when standard statements are used in the simulation, an SDF parameter file must be specified which, as basic system syntax file, contains the syntax file (see the example on page 549).

These statements include:

– MODIFY-SDF-OPTIONS:
  This activates or deactivates a user syntax file and modifies the SDF options for simulation.

– SHOW-SDF-OPTIONS:
  This displays the active syntax files and the SDF options set for simulation.

– SHOW-INPUT-HISTORY:
  This shows a list of preceding inputs.

– RESTORE-SDF-INPUT:
  This restores the input.

– SHOW-INPUT-DEFAULTS:
  This shows task-specific default values.

– SHOW-STMT:
  Show the syntax of a statement

– RESET-INPUT-DEFAULTS
  This resets task-specific default values.

– WRITE-TEXT:
  This displays text that has been entered.

– REMARK:
  This provides comments on statement sequences.

– STEP:
  This interrupts the spin-off mechanism.

– END:
  This terminates the simulated program.

The SDF standard statements EXECUTE-SYSTEM-CMD and HOLD-PROGRAM (as of SDF V4.0 and BS2000/OSD-BC V2.0) are no longer supported in SDF-SIM.

Program notes

– If the program name specified in the DEFINE-TEST-OBJECT statement is not defined in the activated syntax file hierarchy, execution of START-SIMULATION is followed by the display of the syntax file hierarchy, output of the message
```
% SDS0006 PROGRAM '...' NOT DEFINED IN ACTIVATED SYNTAX FILES
```
and the termination of SDF-SIM.

– The SDF-A statement ADD-PROGRAM ... COMMENT-LINE=... can be used to add comment lines to programs. If COMMENT-LINE=*STD was specified, no comment line will appear in the simulation, since SDF-SIM cannot access this internal program information.

7.1.3 SDF-SIM execution within a procedure or a batch task

Like other utilities, SDF-SIM can also be called in procedures or batch tasks. However, this does not implicitly mean that commands or statements are automatically tested in simulated procedure or batch mode. This is only possible if the DEFINE-ENVIRONMENT statement with the operand PROC-MODE=*YES or TASK-TYPE=*BATCH is entered.

If the value QUESTION is specified explicitly or implicitly (as the default value) for the DISPLAY operand of the DEFINE-ENVIRONMENT statement, this has the same effect in procedures and batch tasks as DISPLAY=*NO.

In the case of commands or statements which are tested within a procedure or a batch task in procedure or batch mode, an asterisk (*) must be entered before the slash (/) or double slash (//). If this is not done, the entries will be interpreted as real commands or statements and will actually be processed.

7.2 SDF-SIM statements

SDF-SIM offers the user the following statements:

– DEFINE-TEST-OBJECT defines the type of the test object. The test object can be either a command or a statement.

– DEFINE-ENVIRONMENT defines a test environment.

– START-SIMULATION terminates simulation preparation and starts simulation.

The SDF standard statements (MODIFY-SDF-OPTIONS, REMARK, RESTORE-SDF-INPUT, SHOW-INPUT-HISTORY, SHOW-SDF-OPTIONS, SHOW-STMT, STEP, SHOW-INPUT-DEFAULTS, RESET-INPUT-DEFAULTS, WRITE-TEXT) can likewise be used in SDF-SIM. These standard statements will be actually executed at any point in the SDF-SIM run, i.e. even after START-SIMULATION (during * prompting), and therefore have a direct effect on the simulation run. For example, you can enter statements in guided dialog during simulation or conduct an error dialog.

The metasyntax used for SDF-SIM statements can be found in section “Metasyntax” on page 7. The SDF standard statements are described in the “Introductory Guide to the SDF Dialog Interface” [1].

DEFINE-ENVIRONMENT
Define test environment

The DEFINE-ENVIRONMENT statement defines the test environment for simulation.

The following options can be defined for the environment:

– syntax file hierarchy (PARAMETER-FILE operand)

– procedure mode (PROC-MODE operand)

– batch or dialog mode (TASK-TYPE operand)

– behavior in the event of an error (SPINOFF operand)

– display of the standardized transfer area (DISPLAY operand)

– privileges of the task (PRIVILEGES operand).

– display of call information

(part 1 of 2)

DEFINE-**ENV**IRONMENT
PARAMETER-**FI**LE = ***STD** (...) / ***NO**(...) / <filename 1..54 without-gen-vers>(...)
***STD**(...)
| **USER** = ***STD** / ***NO** / <filename 1..54>
***NO**(...)
| **SYST**EM = ***STD** / ***NO** / ***CURR**ENT / <filename 1..54>
| ,**GR**OUP = ***STD** / ***NO** / <filename 1..54>
| ,**USER** = ***STD** / ***NO** / <filename 1..54>
<filename 1..54 without-gen-vers>(...)
| **USER** = ***STD** / ***NO** / <filename 1..54>
,**PROC-MODE** = ***NO** / ***YES**
,**TASK-TYPE** = ***DIALOG** / ***BATCH**
,**SPINOFF** = ***NO** / ***YES**
,**DISPL**AY = ***QUESTION** / ***NO** / ***SHORT** / ***LONG**

continued ➠

(part 2 of 2)

,**PRIV**ILEGES = ***ALL** / list-poss(2000): ***TSOS** / ***SEC**URITY-**ADM**INISTRATION / ***USER-ADM**INISTRATION / ***HSMS-ADM**INISTRATION / ***SEC**URE-**OLTP** / ***TAPE-ADM**INISTRATION / ***SAT-FILE-MANAG**EMENT / ***NET-ADM**INISTRATION / ***FT-ADM**INISTRATION / ***FTAC-ADM**INISTRATION / ***HARDWARE-MAINT**ENANCE / ***SAT-FILE-EVAL**UATION / ***SUBSYS**TEM-**MANAG**EMENT / ***SW-MONITOR-ADM**INISTRATION / ***ACS-ADM**INISTRATION / ***VM2000-ADM**INISTRATION / ***VIRT**UAL-**MACH**INE-**ADM**INISTRATION / ***SEC**URE-**UTM** / ***PROP-ADM**INISTRATION / ***OPERA**TING / ***STD-PROC**ESSING / ***POSIX-ADM**INISTRATION / ***PRINT-SERVICE-ADM**INISTRATION

,**CALL-INF**ORMATION = ***YES** / ***NO**

PARAMETER-FILE =
Determines the syntax file hierarchy in which the commands or statements are simulated.

PARAMETER-FILE = *STD(...)
The SDF standard parameter file is used.

USER =
Determines the user syntax file.

USER = *STD
The current user syntax file is used.

USER = *NO
No user syntax file is activated.

USER = <filename 1..54>
Name of the user syntax file that is to be activated.

PARAMETER-FILE = *NO
No parameter file is used. The user defines the syntax file hierarchy himself.

SYSTEM =
Determines the system syntax file.

SYSTEM = *STD
The current basic system syntax file is used.

SYSTEM = *NO
No system syntax file is used.

SYSTEM = *CURRENT
The current basic system syntax file and the current subsystem syntax file are used.

SYSTEM = <filename 1..54>
Name of the system syntax file that is to be activated.

GROUP =
Determines the group syntax file.

GROUP = *STD
The current group syntax file is used.

GROUP = *NO
No group syntax file is activated.

GROUP = <filename 1..54>
Name of the group syntax file that is to be activated.

USER =
Determines the user syntax file.

USER = *STD
The current user syntax file is used.

USER = *NO
No user syntax file is activated.

USER = <filename 1..54>
Name of the user syntax file that is to be activated.

PARAMETER-FILE = <filename 1..54 without-gen-vers>(...)
Name of the parameter file that is used.

USER =
Determines the user syntax file.

USER = *STD
The current user syntax file is used.

USER = *NO
No user syntax file is activated.

USER = <filename 1..54>

Name of the user syntax file that is to be activated.

i If a parameter file is used for input (*STD or <filename>), the names of the system syntax file and the group syntax file assigned to the profile ID of TSOS (SYS-TSOS) are read from the parameter file and activated for simulation.
If the parameter file contains errors or if the syntax files are invalid, the system syntax file and the TSOS group syntax file with the standard names are used and messages CMD0685 and CMD0686 are output.

If the name of the system, group or user syntax file is entered and an error occurs while activating the syntax file, an error message is output. If the system syntax file or the group syntax file contains errors, the syntax files with the standard names are

used. If the user syntax file is errored, no user syntax file is activated.
At least one system or group syntax file must be activated, otherwise simulation is not possible as there is no global information available.

If *STD is entered as the system syntax file, only the current basic system syntax file is activated.

PROC-MODE =
Determines whether procedure mode is to be simulated.

PROC-MODE = <u>*NO</u>
Procedure mode is not to be simulated in the test environment.

PROC-MODE = *YES
Procedure mode is to be simulated in the test environment.

In simulated procedure mode, commands must be entered preceded by a slash (/) and statements must be entered preceded by a double slash (//). Commands and statements can also be entered in lowercase letters.
The spin-off mechanism can be activated following an error (see the SPINOFF operand, page 545). The spin-off mechanism is stopped by entering the SET-JOB-STEP command or the STEP statement.
Procedure parameters cannot be replaced.
This environment makes it possible to test commands and statements that are only permitted in batch mode.

TASK-TYPE =
Determines the type of the task.

TASK-TYPE = <u>*DIALOG</u>
Dialog mode is used in the environment.

TASK-TYPE = *BATCH
Batch mode is used in the environment.

In simulated batch mode, commands must be entered preceded by a slash (/) and statements must be entered preceded by a double slash (//). Commands and statements can also be entered in lowercase letters. The spin-off mechanism can be activated following an error (see the SPINOFF operand, page 545). The spin-off mechanism is stopped by entering the SET-JOB-STEP command or the STEP statement.
This environment makes it possible to test commands and statements that are only permitted in batch mode.

SPINOFF =
Determines whether the spin-off mechanism is to be activated for errored commands or statements. This operand is used only if PROC-MODE=*YES or TASK-TYPE=*BATCH has been set.

SPINOFF = <u>*NO</u>
The spin-off mechanism is not activated in the event of an error.

SPINOFF = *YES
The spin-off mechanism is activated in the event of an error. To stop this mechanism, the SET-JOB-STEP command or the STEP statement must be processed.

DISPLAY =
Determines whether the contents of the standardized transfer area are displayed. This operand does *not* apply to commands defined with ADD-CMD...,IMPLEMENTOR= *TPR(...,CMD-INTERFACE=*STRING...) (see SDF-A).

DISPLAY = <u>*QUESTION</u>
Following the syntax analysis, the following query is issued for each command or statement:

```
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT
```

In procedures and batch tasks, DISPLAY=*QUESTION has the same effect as DISPLAY=*NO.

DISPLAY = *NO
The contents of the standardized transfer area are not displayed.

DISPLAY = *SHORT
The contents of the standardized transfer area are displayed in character format and in hexadecimal format with a line length of 80 characters per line.

DISPLAY = *LONG
In addition to the information output for DISPLAY=*SHORT, a structure description (STRUCTURED DESCRIPTION) is displayed. This contains the type, length and value of every operand and can be up to 15000 bytes long.
For a correct output to be received in the case of statements which use the normal format as of SDF V4.1 for the standardized transfer area, *STMT(...LAYOUT=*NEW) must be specified for DEFINE-TEST-OBJECT.

PRIVILEGES =
Determines the privileges which the task has in the test environment.

PRIVILEGES = <u>*ALL</u>
The task has all privileges.

PRIVILEGES = list-poss(2000): *TSOS / *SECURITY-ADMINISTRATION / *USER-ADMINISTRATION / *HSMS-ADMINISTRATION / *SECURE-OLTP / *TAPE-ADMINISTRATION / *SAT-FILE-MANAGEMENT / *NET-ADMINISTRATION / *FT-ADMINISTRATION / *FTAC-ADMINISTRATION / *HARDWARE-MAINTENANCE / *SAT-FILE-EVALUATION / *SUBSYSTEM-MANAGEMENT / *SW-MONITOR-ADMINISTRATION / *ACS-ADMINISTRATION / *VM2000-ADMINISTRATION / *VIRTUAL-MACHINE-ADMINISTRATION / *SECURE-UTM / *PROP-ADMINISTRATION / *OPERATING / *STD-PROCESSING *POSIX-ADMINISTRATION / *PRINT-SERVICE-ADMINISTRATION
The task has precisely those privileges included in this list.

CALL-INFORMATION = <u>*YES</u> / *NO
Defines whether call information is to be shown. The call information includes:
– a log of the command or statement
– the interface type of the command (ASS/ISL/SPL/PROCEDURE)
– the name of the entry point for TPR commands.

DEFINE-TEST-OBJECT
Define test object

The DEFINE-TEST-OBJECT statement defines whether the syntax of commands or statements is to be tested.

```
DEFINE-TEST-OBJECT

TEST-MODE = *CMD / *STMT(...)

  *STMT(...)
    |  PROGRAM-NAME = <name 1..8>
    | ,LAYOUT = *OLD / *NEW
```

TEST-MODE =
Defines the type of the test objects.

TEST-MODE = *CMD
The test objects are commands.

TEST-MODE = *STMT(...)
The test objects are statements.

PROGRAM-NAME = <name 1..8>
Internal name of the program to which the statements to be simulated belong.

LAYOUT = *OLD / *NEW
Format of the standardized transfer area. If statements use the new format (as of SDF V4.1), *NEW must be specified for correct outputs to be received from SDF-SIM.

START-SIMULATION
Start simulation

The START-SIMULATION statement starts simulation in the environment defined by DEFINE-ENVIRONMENT for the test object defined by DEFINE-TEST-OBJECT.

Once simulation has started, SDF-SIM displays the current syntax file hierarchy. It then displays an asterisk (*) prompting you to enter the command or statement to be simulated.

START-SIMULATION

This statement has no operands.

7.3 Examples of the application of SDF-SIM

7.3.1 Providing SDF standard statements in the simulation

As of SDF V4.1 the SDF standard statements in the syntax file are defined by SDF. They are no longer assigned to the SDF-U syntax file and cannot be copied into the syntax file of the application program. Therefore, when standard statements are used in the simulation, a SDF parameter file must be specified which, as basic system syntax file, contains the syntax file.

In order to use the standard statements in SDF-SIM, it is necessary to carry out the following steps:

- Create a parameter file which contains the syntax file of SDF as basic system syntax file. If the syntax file of the application program is a subsystem syntax file, it must be entered in the SDF parameter file (see “Creating a SDF parameter file with the help of SDF-SIM” on page 550).
- Start SDF-SIM (START-SDF-SIM command)
- Specify simulation environment and text object,
  - if the syntax file of the application program is a system syntax file:

    ```
    //DEFINE-ENVIRONMENT PARAMETER-FILE=MY-PARAMETER-FILE
    //DEFINE-TEST-OBJECT *STMT(PROGRAM-NAME=<internal-program-name>)
    ```

  - if the syntax file of the application program is a user syntax file:

    ```
    //DEFINE-ENVIRONMENT PARAMETER-FILE=MY-PARAMETER-FILE,
                         USER=<program-syntax-file>
    //DEFINE-TEST-OBJECT *STMT(PROGRAM-NAME=<name>,LAYOUT=<layout>)
    ```

- Start simulation (`//START-SIMULATION`).
  Then the SDF standard statements are available in addition to the statements of the application program.

Creating a SDF parameter file with the help of SDF-SIM

Nonprivileged users can also use the MODIFY-SDF-PARAMETERS command in SDF-SIM to create a SDF parameter file for the simulation:

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//define-environment parameter-file=*no(system=$.syssdf.sdf.045)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$.SYSSDF.SDF.045' ACTIVATED
%  SDS0005 'USER' SYNTAX FILE 'SDF.USER.SYNTAX' ACTIVATED
*modify-sdf-parameters scope=*next-session(parameter-file-name=my-parameter-
file),syntax-file-type=*system(name=$.syssdf.sdf.045)
(IN)       modify-sdf-parameters scope=*next-session(parameter-file-name=my-
parameter-file),syntax-file-type=*system(name=$.syssdf.sdf.045)
%  CMD0681 SYNTAX FILE '$.SYSSDF.SDF.045' INSERTED IN PARAMETER FILE 'MY-
PARAMETER-FILE'
*modify-sdf-parameters scope=*next-session(parameter-file-name=my-parameter-
file),syntax-file-type=*subsystem(name=$.syssdf.sdf-a.041,sub-name=sdf-a)
(IN)       modify-sdf-parameters scope=*next-session(parameter-file-name=my-
parameter-file),syntax-file-type=*subsystem(name=$.syssdf.sdf-a.041,sub-
name=sdf-a)
%  CMD0709 SYSTEM SYNTAX FILE '$.SYSSDF.SDF-A.041' INSERTED IN PARAMETER FILE
'MY-PARAMETER-FILE'
*/*
```

7.3.2 Using MODIFY-SDF-OPTIONS

The MODIFY-SDF-OPTIONS command or statement can be used to modify the SDF options during simulation (* prompting). MODIFY-SDF-OPTIONS is not simulated, it is actually executed.

The following options are useful for simulation:

GUIDANCE = *EXPERT / *NO / *MINIMUM / *MEDIUM / *MAXIMUM

for specifying the type of dialog guidance to be used for the syntax test

LOGGING = *INPUT-FORM / *ACCEPTED-FORM / *INVARIANT-FORM

for specifying the form of the log output by SDF-SIM

PROCEDURE-DIALOGUE = *YES / *NO

for simulating a procedure dialog in the event of an error (valid only if PROC-MODE=*YES has been set in the DEFINE-ENVIRONMENT statement)

CONTINUATION = *OLD-MODE / *NEW-MODE

only for the simulated procedure or batch mode.

Test with interactive corrections in simulated procedure mode

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*std(user=*no),proc-mode=*yes,display=*no
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'GROUP' SYNTAX FILE '$TSOS.SYS.SDF.GROUP.SYNTAX.TSOS' ACTIVATED
*/mod-sdf-options guid=*no,proc-dial=*yes ―――――――――――――――――――――――――  (1)
(IN)       /mod-sdf-options guid=*no,proc-dial=*yes
CMD:
*/sh-fi-att file-ne=aaaa
(IN)       /sh-fi-att file-ne=aaaa
%  CMD0185 OPERAND NAME 'FILE-NE' COULD NOT BE IDENTIFIED.
ENTER OPERANDS: ――――――――――――――――――――――――――――――――――――――――――――――――――  (2)
file-ne=aaaa
*file-name=aaaa
(IN)      /show-file-attributes file-name=aaaa

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS AAAA,LIST=(SYSOUT)
CMD:
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. The MODIFY-SDF-OPTIONS command is executed in the simulation context. Since simulation is executed in procedure mode (DEFINE-ENV ...,PROC-MODE=*YES), an error dialog is conducted if an error is detected (GUIDANCE=*NO).
2. The entry of an invalid operand name initiates an error dialog.

Logging during simulation

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env *std(user=*no),display=*no
%//start-simulation
  .
  .
*mod-sdf-opt logging=*inv ————————————————————————————————————————————— (1)
(IN)      mod-sdf-opt logging=*inv
*sh-f-att aaa.
(IN)      SHOW-FILE-ATTRIBUTES FILE-NAME=AAA.,INFORMATION=*NAME-AND-
SPACE,SELECT=*ALL,OUTPUT=*SYSOUT,OUTPUT-OPTIONS=*PARAMETERS(SORT-LIST=
*BY-FILENAME)      ————————————————————————————————————————————————————— (2)

ENTRY : DCOFSDF
INTERFACE : ISL
STRUCTURE-FORM
*mod-fi-att aaa,bbb,wr-pass=c'111'
(IN)      MODIFY-FILE-ATTRIBUTES FILE-NAME=AAA,NEW-NAME=BBB,SUPPORT=
*UNCHANGED,PROTECTION=*PARAMETERS(PROTECTION-ATTR=*UNCHANGED,ACCESS=*BY-
PROTECTION-ATTR,USER-ACCESS=*BY-PROTECTION-ATTR,BASIC-ACL=*BY-PROTECTION-
ATTR,GUARDS=*BY-PROTECTION-ATTR,WRITE-PASSWORD=P,READ-PASSWORD=P,EXEC-
PASSWORD=P,DESTROY-BY-DELETE=*BY-PROTECTION-ATTR,AUDIT=*UNCHANGED,SPACE-
RELEASE-LOCK=*BY-PROTECTION-ATTR,EXPIRATION-DATE=*BY-PROTECTION-ATTR,FREE-
FOR-DELETION=*BY-PROT-ATTR-OR-UNCH),SAVE=*UNCHANGED,MIGRATE=*STD,CODED-
CHARACTER-SET=*UNCHANGED,DIALOG-CONTROL=*STD,OUTPUT=*NO

ENTRY : CMDFCAT
INTERFACE : ISL
STRUCTURE-FORM
  .
  .
```

1. Logging is to be as detailed as possible (INVARIANT-FORM).
2. The simulated commands are logged in their most detailed form. For this reason, the operands for which default values are set are also displayed.

7.3.3 Testing in temporary guided dialog

In temporary guided dialog application domains and command/statement structures (operands and default values) can be tested.

The following example shows the test of the COPY-FILE command contained in the system syntax file $TSOS.SYS.SDF.SYSTEM.SYNTAX.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(system=$.sys.sdf.system.syntax,group=*no,user=*no)
%//start-simulation
 .
 .
*copy-file? ——————————————————————————————————————————————————————————  (1)
```

```
COMMAND  : COPY-FILE


--------------------------------------------------------------------------------
FROM-FILE             = a
TO-FILE               = b                                                   (2)
PROTECTION            = *STD
REPLACE-OLD-FILES     = *YES
BLOCK-CONTROL-INFO    = *KEEP-ATTRIBUTE
IGNORE-PROTECTION     = *NO
DIALOG-CONTROL        = *STD
OUTPUT                = *NO







--------------------------------------------------------------------------------
NEXT = *EXECUTE
       *EXECUTE"F3" / Next-cmd / *CONTINUE / *EXIT"K1" / *EXIT-ALL"F1" /
       *TEST"F2"



________________________________________________________________________________
LTG                                                            TAST
```

```
(IN)       /COPY-FILE FROM-FILE=A,TO-FILE=B

ENTRY : DCOCOPF
INTERFACE : ISL
STRUCTURE-FORM
%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT?short
```

```
          *** TRANSFER AREA ***
          ßßßßßßßßßßßßßßßßßßßßß
                       C O  P F I L
          (2250E0) 0052C3D6 D7C6C9D3 40400000 00000000

          (2250F0) 00000009 800B0022 512A800B 0022512E
                                                  A
          (225120) 00000000 00000000 00000001 C1000001
                    B
          (225130) C200
     */*
     %  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. Testing is not possible in temporary guided dialog unless the question mark (?) required for normal operation with SDF is entered here too.
2. When displayed on the screen, the command or statement operands have their preset values. The values that are to be checked (in this case a and b) are entered.

i In order to test with permanent guided dialog or with an error dialog, MODIFY-SDF-OPTIONS must be used in the simulation (GUIDANCE=*MINIMUM/*MEDIUM/*MAXIMUM/*NO).

7.3.4 Testing with the maximum guidance level

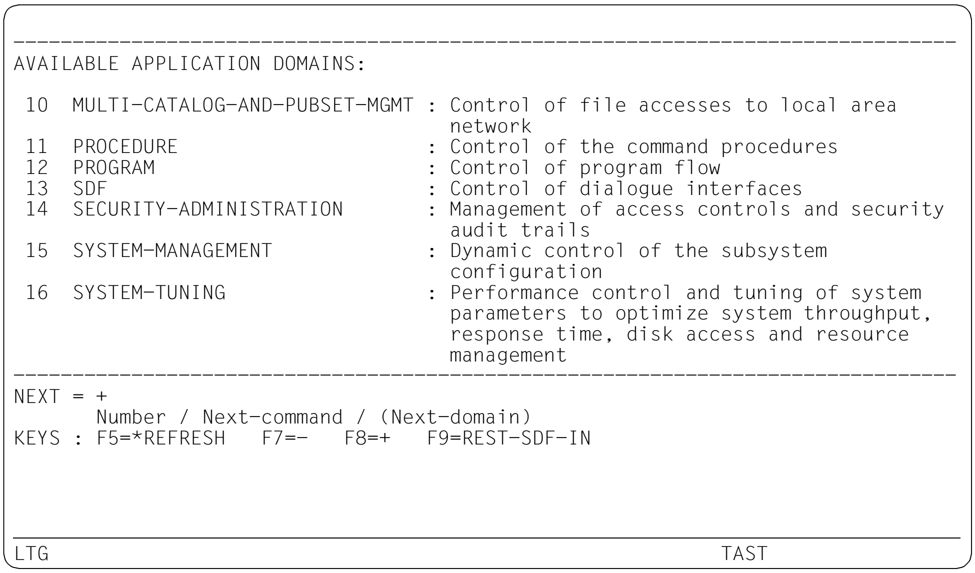
When working with the maximum guidance level you can test and check not only the commands/statements, operands and values, but also the application domains and the help texts for the commands/statements and operands.

The following example shows how the MODIFY-USER-SWITCHES command defined in the current system syntax file of the task is tested.

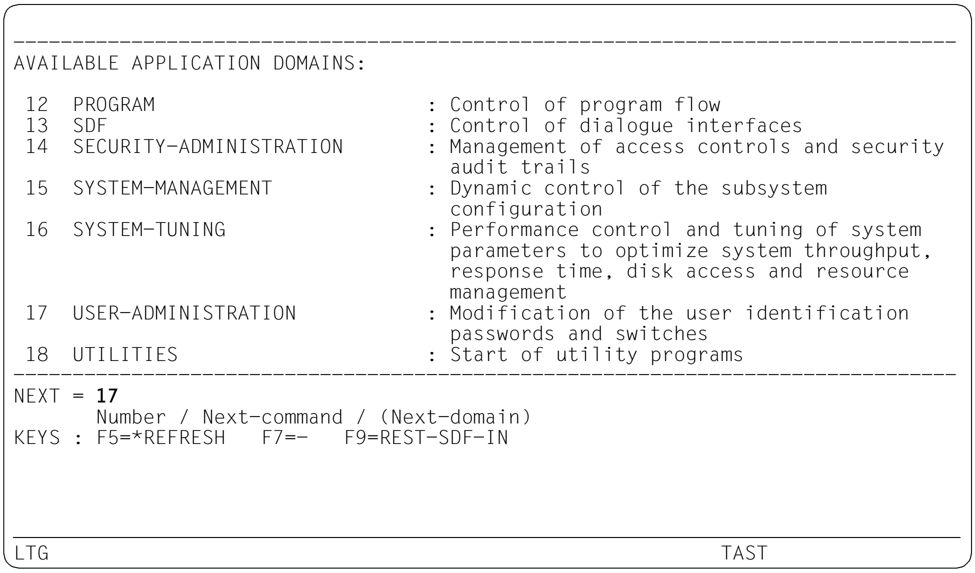
```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(system=*std)
%//start-simulation
 .
 .
*mod-sdf-opt guidance=*max ———————————————————————————————————————————  (1)
(IN)      mod-sdf-opt guidance=*max
```

```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

  1  ACCOUNTING                    : Output of informations about the user
                                     identification and introduction of data
                                     into the accounting record
  2  ALL-COMMANDS                  : Output of all command names in alphabetic
                                     order
  3  CONSOLE-MANAGEMENT            : Control of operator console/terminal
  4  DATABASE                      : Management and administration of databases
  5  DCAM                          : Control of transaction-driven system (DCAM)
  6  DCE                           : Management of DCE (Distributed Computing
                                     Environment)
  7  DEVICE                        : Information about devices and volumes
  8  FILE                          : Management of files
  9  FILE-GENERATION-GROUP         : Management of file generation groups
--------------------------------------------------------------------------------
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F8=+   F9=REST-SDF-IN




________________________________________________________________________________
LTG                                                            TAST
```

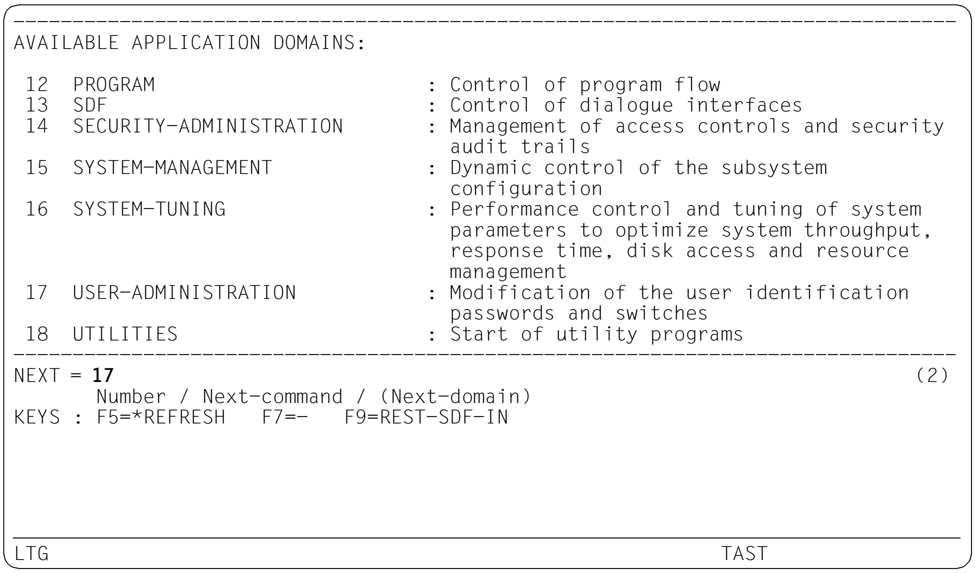
```
 .
 .
```



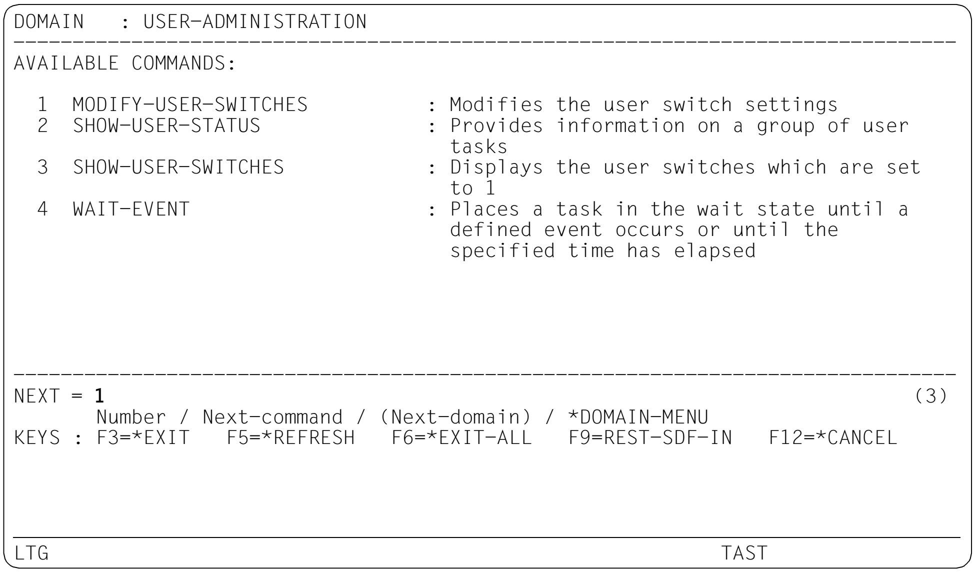
```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 10  MULTI-CATALOG-AND-PUBSET-MGMT : Control of file accesses to local area
                                     network
 11  PROCEDURE                     : Control of the command procedures
 12  PROGRAM                       : Control of program flow
 13  SDF                           : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION       : Management of access controls and security
                                     audit trails
 15  SYSTEM-MANAGEMENT             : Dynamic control of the subsystem
                                     configuration
 16  SYSTEM-TUNING                 : Performance control and tuning of system
                                     parameters to optimize system throughput,
                                     response time, disk access and resource
                                     management
--------------------------------------------------------------------------------
NEXT = +
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F8=+   F9=REST-SDF-IN




________________________________________________________________________________
LTG                                                          TAST
```



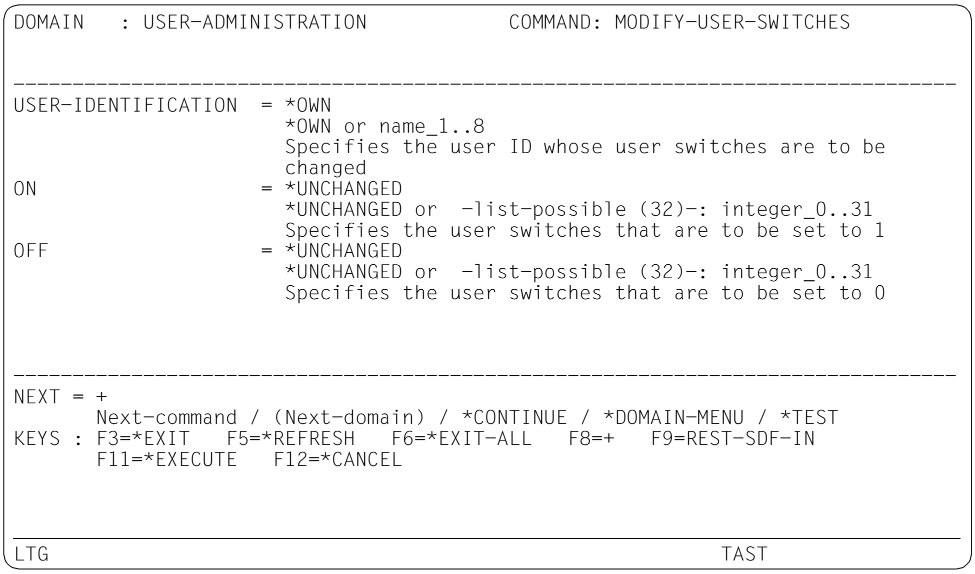
```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 12  PROGRAM                       : Control of program flow
 13  SDF                           : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION       : Management of access controls and security
                                     audit trails
 15  SYSTEM-MANAGEMENT             : Dynamic control of the subsystem
                                     configuration
 16  SYSTEM-TUNING                 : Performance control and tuning of system
                                     parameters to optimize system throughput,
                                     response time, disk access and resource
                                     management
 17  USER-ADMINISTRATION           : Modification of the user identification
                                     passwords and switches
 18  UTILITIES                     : Start of utility programs
--------------------------------------------------------------------------------
NEXT = 17
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F9=REST-SDF-IN




________________________________________________________________________________
LTG                                                          TAST
```



```
--------------------------------------------------------------------------------
AVAILABLE APPLICATION DOMAINS:

 12  PROGRAM                          : Control of program flow
 13  SDF                              : Control of dialogue interfaces
 14  SECURITY-ADMINISTRATION          : Management of access controls and security
                                        audit trails
 15  SYSTEM-MANAGEMENT                : Dynamic control of the subsystem
                                        configuration
 16  SYSTEM-TUNING                    : Performance control and tuning of system
                                        parameters to optimize system throughput,
                                        response time, disk access and resource
                                        management
 17  USER-ADMINISTRATION              : Modification of the user identification
                                        passwords and switches
 18  UTILITIES                        : Start of utility programs
--------------------------------------------------------------------------------
NEXT = 17                                                                    (2)
       Number / Next-command / (Next-domain)
KEYS : F5=*REFRESH   F7=-   F9=REST-SDF-IN



________________________________________________________________________________
LTG                                                         TAST
```



```
DOMAIN   : USER-ADMINISTRATION
--------------------------------------------------------------------------------
AVAILABLE COMMANDS:

  1  MODIFY-USER-SWITCHES             : Modifies the user switch settings
  2  SHOW-USER-STATUS                 : Provides information on a group of user
                                        tasks
  3  SHOW-USER-SWITCHES               : Displays the user switches which are set
                                        to 1
  4  WAIT-EVENT                       : Places a task in the wait state until a
                                        defined event occurs or until the
                                        specified time has elapsed



--------------------------------------------------------------------------------
NEXT = 1                                                                     (3)
       Number / Next-command / (Next-domain) / *DOMAIN-MENU
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F9=REST-SDF-IN   F12=*CANCEL



________________________________________________________________________________
LTG                                                         TAST
```



```
DOMAIN   : USER-ADMINISTRATION           COMMAND: MODIFY-USER-SWITCHES


--------------------------------------------------------------------------------
USER-IDENTIFICATION  = *OWN
                       *OWN or name_1..8
                       Specifies the user ID whose user switches are to be
                       changed
ON                   = *UNCHANGED
                       *UNCHANGED or  -list-possible (32)-: integer_0..31
                       Specifies the user switches that are to be set to 1
OFF                  = *UNCHANGED
                       *UNCHANGED or  -list-possible (32)-: integer_0..31
                       Specifies the user switches that are to be set to 0



--------------------------------------------------------------------------------
NEXT = +
       Next-command / (Next-domain) / *CONTINUE / *DOMAIN-MENU / *TEST
KEYS : F3=*EXIT   F5=*REFRESH   F6=*EXIT-ALL   F8=+   F9=REST-SDF-IN
       F11=*EXECUTE   F12=*CANCEL



________________________________________________________________________________
LTG                                                           TAST
```

.
.

1. The MODIFY-SDF-OPTIONS command is executed in the context of the simulation. The possible domains are displayed on the screen, together with their help texts. The user can select one of the domains.
2. The user selects the domain USER-ADMINISTRATION by entering the number 17.
3. The MODIFY-USER-SWITCHES command is selected by entering the number 1. The operand form for the command is displayed, together with its help texts and default values. The user can now enter the values to be tested.

7.3.5 Replacing job variables

Job variables are replaced in SDF-SIM in exactly the same way as outside the context of simulation.

```
 .
 .
/create-jv jv-name=jv1
/modify-jv jv-contents=jv1,set-value='bbb' ────────────────────────────  (1)
 .
 .
/start-sdf-sim
 .
 .
*sh-f-att &(jv1)
(IN)      sh-f-att bbb ─────────────────────────────────────────────────  (2)

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS BBB,LIST=(SYSOUT)
*enter-job &(jv1),j-class=jcb00200
(IN)      enter-job bbb,j-class=jcb00200

ENTRY : JMGENTR
INTERFACE : ISL
STRING FORM
/ENTER BBB,ERASE=NO,JOB-CLASS=JCB00200,JOB-PRIO=STD,RERUN=NO,FLUSH=NO,
START=STD,REPEAT=STD,RUN-PRIO=STD,TIME=STD,PRINT=STD,PUNCH=STD,
LOG=(LISTING=NO)
*
 .
 .
```

1. The user creates a job variable with the name jv1 and assigns it the value ’bbb’.
2. SDF-SIM replaces the job variable by its value.

i Users should bear in mind that it is not possible to replace procedure parameters.

7.3.6 Simulation of procedure or batch mode

SDF-SIM permits testing in a simulated procedure mode if PROC-MODE=*YES is specified in the DEFINE-ENVIRONMENT statement. If TASK-TYPE=*BATCH is specified in this statement, testing can be carried out in simulated batch mode. Commands/statements must begin with a slash (/) or a double slash (//) and may be entered in lowercase form. If SPINOFF=*YES is specified in the DEFINE-ENVIRONMENT statement, the spin-off mechanism is triggered in the event of an error. If SDF-SIM is called in procedures or in batch files, the first position in the line must be occupied by an asterisk (*) for commands/statements to be simulated in procedure or batch mode. If this is not the case, the commands or statements will be interpreted as being real and will actually be processed.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*std(user=*no),proc-mode=*yes,spinoff=*yes,display=*no  (1)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'GROUP' SYNTAX FILE '$TSOS.SYS.SDF.GROUP.SYNTAX.TSOS' ACTIVATED
*fstat aaa.                                                  }
(IN)       FSTAT AAA.                                        } ———————————  (2)
%  CMD0661 DATA RECORD WAS READ INSTEAD OF COMMAND           }
*/fstat aaa.
(IN)       /FSTAT AAA.

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTAT AAA.
*/shh-file-att aaa.                                                     }
(IN)       /SHH-FILE-ATT                                                }
%  CMD0186 OPERATION NAME 'SHH-FILE-ATT' UNKNOWN                        }
%  CMD0205 ERROR IN PRECEDING COMMAND OR PROGRAM AND PROCEDURE STEP     } (3)
TERMINATION: COMMANDS WILL BE IGNORED UNTIL /SET-JOB-STEP OR /LOGOFF    }
OR /ABEND IS RECOGNIZED                                                 }
*/sh-file-att aaa.                                                      }
*/set-job-step
(IN)       /SET-JOB-STEP
ENTRY : SSMSTEP
INTERFACE : ISL
STRING FORM
/STEP
```

```
*/sh-file-att aaa.
(IN)       /SH-FILE-ATT AAA.

ENTRY : DCOFSTAT
INTERFACE : ISL
STRING FORM
/FSTATUS AAA.,LIST=(SYSOUT)
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. Procedure mode (PROC-MODE=*YES) is simulated. In addition, the spin-off mechanism (SPINOFF=*YES) is to be triggered in the event of an error.
2. In procedure mode, an asterisk here must be followed by a slash; as the slash is missing, an error message is issued.
3. An invalid command name is entered, thereby activating the spin-off mechanism. The subsequent correction attempt is ignored and cannot be implemented until /SET-JOB-STEP has been processed.

7.3.7 SDF-SIM execution within a procedure or a batch task

Like other utilities, SDF-SIM can also be called in procedures or batch tasks. However, this does not implicitly mean that commands or statements are automatically tested in simulated procedure or batch mode. This is only possible if the DEFINE-ENVIRONMENT statement with the operand PROC-MODE=*YES or TASK-TYPE=*BATCH is entered.

In the case of commands or statements which are tested within a procedure or a batch task in procedure or batch mode, an asterisk (*) must be entered before the slash (/) or double slash (//). If this is not done, the entries will be interpreted as real commands or statements and will actually be processed.

SDF-SIM in a procedure

The following example illustrates the use of SDF-SIM in the procedure SIM.PROC.

Procedure file SIM.PROC

```
/BEGIN-PROCEDURE LOGGING=A
/ASSIGN-SYSDTA *SYSCMD
/START-SDF-SIM
//DEFINE-TEST-OBJECT *CMD                  }
//DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=*NO) }  ------------------------  (1)
//START-SIMULATION                         }
*CR-FILE AAA,SUPP=LLMLM
*/SH-FI-ATT BBBB
*/*
/ASSIGN-SYSDTA *PRIMARY
/END-PROCEDURE
```

Tracer listing for the procedure SIM.PROC

```
(IN)     CALL-PROC SIM.PROC
(IN)     /BEGIN-PROCEDURE LOGGING=A
(IN)     /ASSIGN-SYSDTA *SYSCMD
(IN)     /START-SDF-SIM
(OUT)    %  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY
(   )      ':2OSH:$TSOS.SYSLNK.SDF-SIM.045' IN PROCESS
(OUT)    %  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57'
(   )      'LOADED
(OUT)   %  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
(OUT)    %  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
(IN)     //DEFINE-TEST-OBJECT *CMD
(IN)     //DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=*NO)
(IN)     //START-SIMULATION
```

```
(OUT)    %  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX'
(   )     ACTIVATED
(IN)     *CR-FILE AAA,SUPP=LLMLM ————————————————————————————————————————  (2)
(OUT)    (IN)       CR-FILE
(OUT)    %  CMD0051 INVALID OPERAND 'SUPPORT'
(OUT)    %  CMD0081 VALUE 'P' NOT CONTAINED IN KEYWORD LIST OF VALUE RANGE
(   )    '*PUBLIC-DISK() OR *PRIVATE-DISK() OR *TAPE() OR *NONE'
(IN)     */SH-FI-ATT BBBB ———————————————————————————————————————————————  (3)
(OUT)    (IN)       /SH-FI-ATT BBBB
(OUT)
(OUT)    ENTRY : DCOFSTAT
(OUT)    INTERFACE : ISL
(OUT)    STRING FORM
(OUT)    /FSTATUS BBBB,LIST=(SYSOUT)
(IN)     */*
(OUT)    %  SDS0002 SDF-SIM TERMINATED NORMALLY
(IN)     /ASSIGN-SYSDTA *PRIMARY
(IN)     /END-PROCEDURE
```

1. The statements required for simulation preparation are entered and simulation is started.
2. The CREATE-FILE command is simulated. As an invalid value has been specified for the SUPPORT operand, SDF-SIM reports an error.
3. The spin-off mechanism was not triggered, even though there was an input error. As a result, the next command can be simulated immediately. If the command is preceded by a slash (/), there must be an asterisk (*) at the start of the line. Here, however, the slash is not necessary as procedure mode is not being simulated.

SDF-SIM in a batch task

The following example illustrates the simulation of the statements for the test program TESTPRO in a batch task. The prerequisites outlined above for procedures apply here too, so there is no need to reproduce the tracer listing here.

Batch file SIM.ENTER

```
/LOGON
/START-SDF-SIM
//DEFINE-TEST-OBJECT *STMT(PROG-NAME=TESTPRO)
//DEFINE-ENVIRONMENT PAR-FILE=*STD(USER=TEST.USER),DISPLAY=*SHORT
//START-SIMULATION
*OPEN AAAA,TYPE=*USER, ...  ——————————————————————————————————————————  (1)
 .
 .
*END  ————————————————————————————————————————————————————————————————  (2)
/LOGOFF
```

1. OPEN is a statement defined in the TESTPRO program.
2. If *STMT was specified for DEFINE-TEST-OBJECT, END terminates the SDF-SIM run and issues the message `END OF PROGRAM`. The END statement is not simulated.

7.3.8 Command implemented by a procedure

In the following example the syntax of a user-defined command is tested. The command is defined in the user syntax file SF.TEST.

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test-obj *cmd
%//def-env par-fi=*no(sys=*std,group=*n,user=sf.test),displ=*long ——— (1)
%//start-simulation
%  SDS0005 'SYSTEM' SYNTAX FILE '$TSOS.SYS.SDF.SYSTEM.SYNTAX' ACTIVATED
%  SDS0005 'USER' SYNTAX FILE 'SF.TEST' ACTIVATED
*my-com first-op=aaa,second-op=bbbb ————————————————————————————————— (2)
(IN)      my-com first-op=aaa,second-op=bbbb

INTERFACE : PROCEDURE                            }
STRING FORM                                      } —————————————————— (3)
/CALL MY-PROC,(FIRST-OP='AAA',SECOND-OP='BBBB')  }
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

1. The user syntax file SF.TEST containing the syntax definition of the command MY-COMMAND must be activated in the syntax file hierarchy of the test environment.
2. Once simulation has been started, the user enters the command to be simulated, together with two operands.
3. SDF-SIM outputs a log containing the following information:
   - interface type PROCEDURE
   - transfer in the form of a character string
   - generated procedure call with the operands as procedure parameters.

7.3.9 Displaying the standardized transfer area

The following example illustrates the display of the standardized transfer area for a simulated SDF-A statement. In the case of commands, the standardized transfer area is displayed only if it was defined with ADD-CMD ...,IMPLEMENTOR= *TPR(...,CALL=*NEW...) (see SDF-A). The standardized transfer area is displayed as shown in the example:

- if the operand DISPLAY=*LONG has been set in the DEFINE-ENVIRONMENT statement, or
- if “LONG” is entered in response to the message
  `%  SDS0008 DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT`

```
/start-sdf-sim
%  BLS0523 ELEMENT 'SDF-SIM', VERSION 'V04.5A10' FROM LIBRARY ':2OSH:$TSOS.SY
SLNK.SDF-SIM.045' IN PROCESS
%  BLS0524 LLM 'SDF-SIM', VERSION ' ' OF '2001-04-19 16:11:57' LOADED
%  BLS0551 COPYRIGHT (C) SIEMENS AG 2001. ALL RIGHTS RESERVED
%  SDS0001 SDF-SIM VERSION 'V04.5A10' STARTED
%//def-test *stmt(prog=$sdaed41) ———————————————————————————————————  (1)
%//def-env par-fi=*no(sys=$tsos.syssdf.sdf-a.041,group=*n,user=*n),-
%//display=*long ————————————————————————————————————————————————————  (2)
%//start-simulation
%  CMD0692 THE SYSTEM SYNTAX FILE $TSOS.SYSSDF.SDF-A.041 DOESN‘T CONTAIN THE
COMMAND MODIFY-SDF-PARAMETERS
%  SDS0005 ’SYSTEM’ SYNTAX FILE ’$TSOS.SYSSDF.SDF-A.041’ ACTIVATED
//
*add-cmd name=my-command,int-name=mycom,batch-allowed=*no,-
//
*impl=*proc(name='my-proc')
(IN)       add-command name=my-command,int-name=mycom,batch-allowed=*no,impl=
*proc(name=’my-proc’)
```

1. The statements of the program SDF-A are defined as test objects. The internal program name for SDF V4.1 is $SDAED41.
2. It is defined for the environment that only the system syntax file of SDF-A be activated. DISPLAY=*LONG specifies that the standardized transfer area and the structure description will be displayed at once and without having to request them. The output is reproduced on the following pages.

```
*** TRANSFER AREA ***
~~~~~~~~~~~~~~~~~~~~~
              A D  D C M D
(129418) 3A98C1C4 C4C3D4C4 40400000 00000000

(129428) 00000046 80080012 95D00000 00000000

(129438) 80160012 95F88016 001295EA 80160012

(129448) 95F08016 001295F4 00000000 00008007

(129458) 001295DC 00000000 00000000 00000000

(129468) 00000000 00008016 0012962C 80160012

(129478) 96308016 00129698 80130012 965E8013

(129488) 00129636 80130012 964A8013 00129672

(129498) 00000000 00000000 00000000 80050012

(1294A8) 960C8016 001295E4 80160012 95FC8013

(1294B8) 00129684 80160012 96000000 00000000

(1294D8) 00000000 00000000 00000000 00008016

(1294E8) 0012969E 80160012 96A80000 00000000

(129508) 00000000 00000000 00000000 00008016

(129518) 00129616 80160012 96260000 00000000

(129558) 00000000 00000000 00000000 80160012

(129568) 96AE0000 00000000 80160012 967E8016

(129578) 00129692 80160012 96448016 00129658

(129588) 80160012 966C8016 001296A2 00000000
                                M Y  - C O M
(1295C8) 00000000 00000000 000AD4E8 60C3D6D4
          M A N D      M Y  C O M        S A
(1295D8) D4C1D5C4 0005D4E8 C3D6D400 0004E2C1
          M E      N A M E      N O      N O
(1295E8) D4C50004 D5C1D4C5 0002D5D6 0002D5D6
              N O      N O      P R  O C E D
(1295F8) 0002D5D6 0002D5D6 0009D7D9 D6C3C5C4
```

```
          U R E        M Y  - P R O  C
(129608) E4D9C500 0007D4E8 60D7D9D6 C300000E
          C A L L  - P R O  C E D U  R E
(129618) C3C1D3D3 60D7D9D6 C3C5C4E4 D9C50004
          N O N E      N O      Y E  S
(129628) D5D6D5C5 0002D5D6 0003E8C5 E2000000
                            Y E S        A L
(129638) 80160012 963E0003 E8C5E200 0003C1D3
          L                          Y E S
(129648) D3000000 80160012 96520003 E8C5E200
              A L  L
(129658) 0003C1D3 D3000000 80160012 96660003
          Y E S        A L  L
(129668) E8C5E200 0003C1D3 D3000000 80160012
                   N O      A L L
(129678) 967A0002 D5D60003 C1D3D300 00008016
                       Y E  S        A L L
(129688) 0012968C 0003E8C5 E2000003 C1D3D300
              Y E  S        N O      A L L
(129698) 0003E8C5 E2000002 D5D60003 C1D3D300
              C M  D        A L L
(1296A8) 0003C3D4 C4000003 C1D3D300 00000000

*** STRUCTURED DESCRIPTION ***
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
INTERNAL NAME OF THE COMMAND : ADDCMD
VERSION OF THE TRANSFER AREA : x    } (only for the new format of the
VERSION OF THE CMD/STMT      : x    } transfer area (as of SDF V4.1))
MAXIMUM NUMBER OF OPERANDS   :   70
OP( 1) :
 - TYPE   : STRUCT_NAME
 - LENGTH :   10
 - VALUE  :
      M Y - C  O M M A  N D
     D4E860C3 D6D4D4C1 D5C4
OP( 3) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
      N O
     D5D6
OP( 4) :
 - TYPE   : KEYW
 - LENGTH :    4
 - VALUE  :
      N A M E
     D5C1D4C5
OP( 5) :
```

```
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
     N O
    D5D6
OP( 6) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
     N O
    D5D6
OP( 8) :
 - TYPE   : ALPHA_NAME
 - LENGTH :    5
 - VALUE  :
     M Y C O  M
    D4E8C3D6 D4
OP(12) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
     N O
    D5D6
OP(13) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     Y E S
    E8C5E2
OP(14) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     Y E S
    E8C5E2
OP(15) :
 - TYPE   : STRUCTURE
      VALUE INTRODUCING THE STRUCTURE :
     - TYPE   : KEYW
     - LENGTH :    3
     - VALUE  :
         Y E S
        E8C5E2
OP(16) :
 - TYPE   : STRUCTURE
      VALUE INTRODUCING THE STRUCTURE :
     - TYPE   : KEYW
     - LENGTH :    3
```

```
          - VALUE  :
               Y E S
              E8C5E2
    OP(17) :
     - TYPE   : STRUCTURE
           VALUE INTRODUCING THE STRUCTURE :
          - TYPE   : KEYW
          - LENGTH :    3
          - VALUE  :
               Y E S
              E8C5E2
    OP(18) :
     - TYPE   : STRUCTURE
           VALUE INTRODUCING THE STRUCTURE :
          - TYPE   : KEYW
          - LENGTH :    2
          - VALUE  :
               N O
              D5D6
    OP(21) :
     - TYPE   : C_STR
     - LENGTH :    7
     - VALUE  :
          M Y - P  R O C
         D4E860D7 D9D6C3
    OP(22) :
     - TYPE   : KEYW
     - LENGTH :    4
     - VALUE  :
          S A M E
         E2C1D4C5
    OP(23) :
     - TYPE   : KEYW
     - LENGTH :    2
     - VALUE  :
          N O
         D5D6
    OP(24) :
     - TYPE   : STRUCTURE
           VALUE INTRODUCING THE STRUCTURE :
          - TYPE   : KEYW
          - LENGTH :    3
          - VALUE  :
               Y E S
              E8C5E2
    OP(25) :
     - TYPE   : KEYW
     - LENGTH :    9
```

```
 - VALUE  :
     P R O C  E D U R  E
    D7D9D6C3 C5C4E4D9 C5
OP(32) :
 - TYPE   : KEYW
 - LENGTH :    2
 - VALUE  :
     N O
    D5D6
OP(33) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     C M D
    C3D4C4
OP(40) :
 - TYPE   : KEYW
 - LENGTH :   14
 - VALUE  :
     C A L L  - P R O  C E D U  R E
    C3C1D3D3 60D7D9D6 C3C5C4E4 D9C5
OP(41) :
 - TYPE   : KEYW
 - LENGTH :    4
 - VALUE  :
     N O N E
    D5D6D5C5
OP(53) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     A L L
    C1D3D3
OP(55) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     A L L
    C1D3D3
OP(56) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
     A L L
    C1D3D3
OP(57) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(58) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(59) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
OP(60) :
 - TYPE   : KEYW
 - LENGTH :    3
 - VALUE  :
      A L L
     C1D3D3
//
*/*
%  SDS0002 SDF-SIM TERMINATED NORMALLY
```

8 Messages

8.1 SDF-A messages

SDA0001 ’(&01)’ VERSION ’(&00)’ STARTED

SDA0003 COMMENT DOES NOT END WITH A DOUBLE QUOTE (")

Meaning
The comment has to end with a double quote.

SDA0004 CHARACTER STRING ’(&00)’ DOES NOT END WITH A SINGLE QUOTE (’)

Meaning
The character string has to end with a single quote.

SDA0030 OPERAND NAME ’(&00)’ IS LONGER THAN 20 CHARACTERS

Meaning
Maximum length permitted: 20 characters.

SDA0031 SYNTAX ERROR IN OPERAND NAME ’(&00)’

SDA0033 INVALID LIST SPECIFICATION AFTER OPERAND NAME

SDA0034 OPERAND NAME BEFORE EQUALS SIGN ’=’ MISSING

Meaning
Specify the operand name before the equals sign.

SDA0035 OPERAND ’(&00)’ NOT PERMITTED IN PRESENT MODE

Meaning
The operand (&00) is not permitted in the current input mode (see attributes xxx-ALLOWED or the privileges definition in the syntax file).
For more detailed information, see the BS2000 manual ’SDF’ (or ’SDF-A’).

SDA0037 ’(&00)’ CANNOT BE ASSIGNED TO ANY OPERAND. REASON FOR ERROR: ’(&01)’. INPUT IGNORED

SDA0038 ERROR IN COMMAND FORMAT: ’(&00)’ IGNORED

SDA0039 SEVERAL HELP TEXTS SPECIFIED FOR SAME LANGUAGE. LAST SPECIFICATION IS USED

SDA0043	LIST ELEMENT '(&00)' EXCEEDS MAXIMUM PERMITTED NUMBER
SDA0044	CLOSING PARENTHESIS ')' IN LIST '(&00)' MISSING
SDA0045	SPECIFICATION OF OPERAND VALUE '(&00)' IN LIST NOT PERMITTED
SDA0046	NO APPROPRIATE STRUCTURE ACTIVATED. VALUE '(&01)' FOR OPERAND '(&00)' COULD NOT BE INTERPRETED
SDA0047	CLOSING PARENTHESIS ')' IN STRUCTURE '(&00)' MISSING
SDA0048	STRUCTURE SPECIFICATION '(&00)' INVALID AND NOT PERMITTED FOR VALUE '(&01)'
SDA0050	'(&00)' CANNOT BE ASSIGNED TO ANY VALUE. INPUT IGNORED
SDA0052	MODIFICATION OF ADDRESSED OPERAND VALUE OF LIST ELEMENT NOT POSSIBLE BY SPECIFYING '(&00)' IN SEMANTICS ERROR DIALOG
SDA0053	VALUE '(&00)' NOT PERMITTED IN PRESENT MODE

Meaning
The value (&00) is not permitted in the current input mode (see the xxx-ALLOWED attributes or the privileges definition in the syntax file).
For more detailed information, see the BS2000 manual 'SDF' (or 'SDF-A').

SDA0055 DELETION OF LIST ELEMENT '(&00)' BY ENTERING A SHORTENED LIST NOT PERMITTED

Meaning
It is not possible to delete a list element (&00) in semantics
error dialog by entering a shortened list.

SDA0057	INVALID VALUE '(&00)' NOT CORRECTED YET
SDA0058	OPERAND VALUE '(&00)' NOT PERMITTED IN CURRENT MODE
SDA0061	OPERAND VALUE '(&00)' NOT IN PERMISSIBLE RANGE '(&01)'
SDA0062	LENGTH OF OPERAND VALUE '(&00)' NOT IN PERMISSIBLE RANGE FOR DATA TYPE '(&01)'
SDA0063	THE OPERAND VALUE '(&00)' IS NOT MEMBER OF THE SINGLE VALUE LIST OF THE SCOPE '(&01)'
SDA0064	OPERAND VALUE '(&00)' DOES NOT MATCH DATA TYPE '(&01)'
SDA0065	SPECIFICATION '(&00)' CANNOT BE ASSIGNED TO ANY OF THE VALUES '(&01)' (VALUES-OVERLAPPING=YES)
SDA0067	LENGTH OF OPERAND VALUE '(&00)' MUST BE EVEN

Meaning
The value X-TEXT is declared with ODD-POSSIBLE=NO in the syntax file.
Therefore the value given for the VALUE operand must have an even number of characters.

Response
Correct the value.

SDA0071 `NO OPERAND SPECIFIED FOR COMMAND OR STATEMENT`

SDA0072 `ATTRIBUTE SPECIFIED IN FILE NAME '(&00)' NOT PERMITTED`

SDA0073 `INVALID GENERATION OR VERSION SPECIFICATION IN FILE NAME '(&00)'`

SDA0074 `INVALID USER IDENTIFICATION SPECIFIED IN FILE NAME '(&00)'`

SDA0075 `FILE NAME '(&00)' INVALID`

Meaning
The file name (&00) contains characters that are not permitted.

SDA0076 `INVALID GENERATION OR VERSION SPECIFICATION IN FILE NAME`

SDA0077 `SPECIFICATION OF POSITIONAL OPERANDS NOT PERMITTED AFTER OPERAND NAME WITH VALUE`

SDA0078 `INVALID CATALOG IDENTIFICATION SPECIFIED IN FILE NAME '(&00)'`

SDA0079 `SPECIFIED CHARACTERS IN TEXT '(&00)' NOT PERMITTED`

Meaning
Invalid characters:
– equals sign
– blank
– semicolon
– parenthesis.

SDA008A `'PRODUCT-NAME' MUST BE SPECIFIED WHEN ADDING CORRECTION-INFORMATION TO AN INSTALLATION SYNTAX FILE`

SDA008B `PRODUCT-NAME '(&00)' DIFFERS FROM REGISTERED ONE '(&01)'`

SDA008C `PRODUCT-VERSION '(&00)' DIFFERS FROM REGISTERED ONE '(&01)'`

SDA008D `'SOURCE' CORRECTIONS CAN ONLY BE REGISTERED IN COMPONENT SYNTAX FILES`

SDA008E `'OBJECT' CORRECTIONS CAN ONLY BE REGISTERED IN KPSD, SESD, INSD SYNTAX FILES`

SDA008F `THE CORRECTION-INFORMATION CANNOT BE MODIFIED BY THE CURRENT PROGRAM VERSION`

Meaning
The program you use now is old-fashioned.

Response
Please, use an up-to-date program version.

SDA0080 `NUMBER (&00) OUTSIDE PERMITTED RANGE`

SDA0081 `VALUE '(&00)' NOT CONTAINED IN KEYWORD LIST OF VALUE RANGE '(&01)'`

SDA0082 `ABBREVIATION '(&00)' AMBIGUOUS WITH REGARD TO '(&01)'`

Response
Use an unambiguous abbreviation.

SDA0083 `NAME '(&00)' UNKNOWN`

SDA0084 `'(&00)' EQUIVALENT TO TPR COMMAND(S) '(&01)'. COMMAND REJECTED`

Meaning
Possible reasons:
- the specified command already exists
- the specified command corresponds to the abbreviation of a command that already exists or commands that are already defined.

(&00): the specified command
(&01): the command that already exists or a list of the commands that already exist.

SDA0085 `V-RECORD OF COMPONENT SYNTAX FILE IS FULL`

Meaning
The 255 positions in the correction information table are occupied.

Response
Remove the unnecessary entries from this table.

SDA0086 `NO MORE ENTRIES POSSIBLE IN V-RECORD OF COMPONENT SYNTAX FILE`

Meaning
Only some of the given PM operand values are written to the component syntax file.

Response
Remove the unnecessary entries from this table.

SDA0087 `KERNEL OR COMPONENT SYNTAX FILE CANNOT BE OF 'USER' TYPE`

SDA0088 `CORRECTION NUMBER '(&00)' NOT CONTAINED IN SYNTAX FILE`

Meaning
The correction information is not contained in the V-record.

SDA0089 `THERE ARE NO SIGNIFICANT DIGITS OR MORE THAN 10 SIGNIFICANT DIGITS HAVE BEEN FOUND`

Meaning
Only 10 digits can be specified for the SDF data type FIXED. Moreover, at least one digit must be specified.

SDA009A `CHECK ERROR: THE (&00) '(&01)' DOESN'T MATCH`

Meaning
The V_record contains a SOFTWARE-UNIT-NAME, a VERSION and a COMPONENT-VERSION specification. Those values can be used as CHECK-VALUES prior to any V_record update.
(&00) identifies the specification.
(&01) identifies its value.

SDA009B `THE CORRECTION-INFORMATION CANNOT BE MODIFIED: DATA ORGANIZATION NOT RECOGNIZED`

Meaning
The v_record structure is false.

SDA009C `THE CORRECTION-INFORMATION CANNOT BE MODIFIED: THE V-RECORD IS FULL`

Meaning
The v_record is full.

Response
Remove the unnecessary correction-informations.

SDA0090 `OPERAND VALUE '(&00)' VIOLATES PERMITTED LOGICAL LENGTH FOR DATA TYPE '(&01)'`

SDA0099 `MANDATORY OPERAND INVALID OR MISSING`

Meaning
An incorrect value has been specified for an operand which is mandatory.

SDA0159 `LANGUAGE '(&00)' NOT DEFINED IN GLOBALS OF CURRENT SYNTAX FILE HIERARCHY. LANGUAGE '(&01)' IS USED`

SDA0300 `DMS ERROR '(&01)' WHEN ACCESSING FILE '(&00)'. IN SYSTEM MODE: /HELP-MSG DMS(&01)`

Meaning
For more detailed information about the DMS error code, enter /HELP-MSG in system mode or see the BS2000 manual 'System Messages'.

SDA0301 `ERROR DURING OUTPUT TO SYSLST`

SDA0302 `INVALID VERSION OF SYNTAX FILE '(&00)' SELECTED`

SDA0303 `JVS ERROR '(&01)' WHEN ACCESSING JV '(&00)'. IN SYSTEM MODE: /HELP-MSG JVS(&01)`

Meaning
For more detailed information about the JVS error code, enter /HELP-MSG in system mode or see the manual 'JV (BS2000)'.
JVS: Job Variable Service

SDA0304 `DMS ERROR '(&00)' WHEN COPYING KERNEL SYNTAX FILE TO '(&01)'. IN SYSTEM MODE: /HELP-MSG DMS(&00)`

Meaning
For more detailed information about the DMS error code, enter /HELP-MSG in system mode or see the BS2000 manual 'System Messages'.

SDA0305 `VERSION DOES NOT MATCH DOD-FORMAT CONVENTION. INPUT IGNORED`

Meaning
The specified value does not follow the syntax rules.

SDA0306 KERNEL SYNTAX FILE MUST BE OF 'SYSTEM' TYPE. INPUT IGNORED

SDA0307 KERNEL SYNTAX FILE CANNOT BE UPDATED. INPUT IGNORED

SDA0308 ERROR DURING OPEN OF (&00). THIS MAY LEAD TO SOME PROBLEMS

SDA0316 KEYWORD '(&00)' IN INPUT ALREADY EXISTS

SDA0335 OPERAND NAME '(&00)' ALREADY EXISTS

SDA0350 INTERNAL ERROR:'(&00)'

SDA0372 AUTHORIZATION RANGE SPECIFIED IN USER SYNTAX FILE FOR COMMAND '(&00)' WIDER THAN THAT SPECIFIED IN SYSTEM AND GROUP SYNTAX FILES

SDA0373 DESCRIPTION OF SYSTEM INTERFACE FOR COMMAND '(&00)' MODIFIED IN SYNTAX FILE

Meaning
Possible reasons:
- attribute of the command has been changed in the user syntax file.
- the command is overruled by a command defined in the user syntax file.

SDA0376 SPECIFICATION OF POSITIONAL OPERANDS AT POSITION '(&00)' NOT PERMITTED

SDA0377 NUMBER OF DATA TYPES PER OPERAND TOO LARGE

SDA0378 STRUCTURE DEPTH GREATER THAN 5

SDA0379 IN COMMAND '(&00)' FOR OPERAND '(&01)', VALUES WHICH ARE NOT COMPATIBLE WITH FILE HIERARCHY ARE ENTERED IN USER OR GROUP SYNTAX FILE

SDA0380 ERROR WHEN STARTING '$.SDF-A-V1'. PROGRAM TERMINATED ABNORMALLY

Meaning
The program $.SDF-A-V1 is started when switch 15 is set to update syntax files with V1 format which were created with SDF-A 1.0D

Response
Reset switch 15 or contact the system administrator.

SDA0381 ERROR WHEN READING A STATEMENT. PROGRAM TERMINATED ABNORMALLY

SDA0382 ERROR WHEN READING //OPEN-SYNTAX-FILE STATEMENT. PROGRAM TERMINATED ABNORMALLY

SDA0383 FORMAT OF SYNTAX FILE HAS BEEN CHANGED

Meaning
The syntax file has been created using a previous version (V1.0D or earlier). Its structure is now modified and it cannot be opened anymore by those earlier versions.

SDA0384 NO VALUES SPECIFIED FOR OPERAND '(&00)'. OPERAND WILL BE DELETED

SDA0385 NO STRUCTURE OPERANDS ASSIGNED TO VALUE '(&00)'. STRUCTURE WILL BE DELETED
(C) Routing code: * Weight: 99

SDA0386 `ERROR WHEN STARTING '$.SDF-A-V3'. PROGRAM TERMINATED ABNORMALLY`

Meaning
The program $.SDF-A-V3 is started when the statement 'DEFINE-ENVIRONMENT *V3' is specified, to update or create syntax files with V3 format.

Response
Contact the system administrator.

SDA0388 `WARNING: DEFAULT VALUE OF OPERAND '(&00)' LIES OUTSIDE PERMISSIBLE RANGE BECAUSE OF '(&01)'`

SDA0389 `OBJECT '(&00)' ALREADY DEFINED IN SYNTAX FILE`

SDA0390 `COPY DESTINATION NOT DEFINED OR ITS TYPE IS NOT COMPATIBLE`

Meaning
Possible reasons:
- an //ADD or //EDIT statement is missing before the //COPY statement
- the type of the copy destination is not compatible with the type of the copied object.

SDA0391 `INVALID SYNTAX FILE OBJECT. INPUT IGNORED`

SDA0392 `INTERNAL OPERAND OR VALUE '(&00)' ALREADY ASSIGNED`

SDA0393 `CURRENT COMMAND OR CURRENT STATEMENT NOT DEFINED`

SDA0395 `CURRENT OPERAND OR VALUE NOT DEFINED`

SDA0396 `SPECIFIED DATA TYPE CANNOT BE FOUND`

SDA0397 `SELECTED OPERAND OR VALUE NOT AVAILABLE AT CURRENT HIERARCHY LEVEL OF SYNTAX FILE`

SDA0401 `WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO '(&01)'`

Meaning
To avoid inconsistency, the standard name of the control file cannot be suppressed by the syntax file.

SDA0402 `WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO OPERAND '(&01)'`

Meaning
To avoid inconsistency, the standard name of the control file cannot be suppressed by the syntax file.

SDA0403 `WARNING: STANDARD NAME '(&00)' OF CONTROL FILE ADDED TO VALUE '(&01)'`

Meaning
To avoid inconsistency, the standard name of the control file cannot be suppressed by the syntax file.

SDA0404 VALUE TYPE '(&00)' MUST BE SYNTACTICALLY SEPARATED FROM VALUE TYPE '(&01)'

SDA0405 ERROR CAN LEAD TO PROBLEMS OF PRIVILEGING IF SAME HIERARCHY OF SYNTAX FILES IS ACTIVATED

SDA0406 VALUE '(&00)' DOES NOT MATCH CORRESPONDING SYNTAX TYPE BECAUSE OF '(&01)'

SDA0407 CORRECTION REJECTED OR NOT POSSIBLE. STATEMENT IGNORED

SDA0408 ABBREVIATION '(&00)' INVALID AS AN ABBREVIATION OF MAIN VALUE '(&01)'

Meaning
Use an unambiguous abbreviation.

SDA0409 KEYWORD '(&00)' TOO LONG

Meaning
Maximum length permitted: 30 characters.

SDA0410 A SINGLE VALUE MUST BE SPECIFIED FOR 'KEYWORD' DATA TYPE

SDA0411 EXTERNAL COMMAND OR STATEMENT NAME '(&00)' ALREADY ASSIGNED

Meaning
Possible reasons:
- the command or the statement exists.
- the command or the statement is removed at user level but defined at group or system level.

SDA0412 WARNING: RESULT OPERAND LEVEL ALLOCATED TO OPERAND '(&00)' IN '(&01)' IS TOO LARGE AND WILL BE CORRECTED

SDA0413 FLAT STRUCTURE SPECIFIED FOR VALUE '(&00)' ALTHOUGH STRUCTURE ALLOWED WITHIN A LIST

SDA0414 DELETION OF COMMAND OR STATEMENT '(&00)' NOT PERMITTED

SDA0415 DELETION OF OPERAND '(&00)' NOT PERMITTED

SDA0416 DELETION OF VALUE '(&00)' NOT PERMITTED

SDA0417 INTERNAL NAME '(&00)' ALREADY ASSIGNED

SDA0418 '(&00)' OPERAND IGNORED. PROCESSING CONTINUES

Meaning

- INPUT-FORM means that TPR commands can only be generated in INVARIANT or STANDARD input form if their interface is ISL of a version greater than 1.
- SOFTWARE-UNIT-NAME means that the syntax file is not concerned by this operand.
- VERSION means that on open update, it can only be evaluated when SOFTWARE-UNIT-NAME is specified.

SDA0419 ’COMMAND-REST’ DATA TYPE ONLY PERMITTED AT COMMAND LEVEL

SDA0420 WARNING: SOME OBJECTS NOT COPIED. STATEMENT PARTIALLY EXECUTED

Meaning
A problem has been encountered during the processing of one object.
Refer to the preceding error messages.

SDA0421 VALUE SPECIFIED FOR ’COMMAND-REST’ DATA TYPE NOT PERMITTED

SDA0422 FOR A VALUE INTRODUCED BY AN OPERAND DECLARED ’LIST-POSSIBLE=NO’, LIST SPECIFICATION NOT POSSIBLE

SDA0423 OPERATION ADD-OP NOT PERMITTED AT THE MOMENT

Meaning
Definition *command-rest data type only permitted for last operand

SDA0424 FILE NAME WITH GENERATION OR VERSION SPECIFICATION MUST NOT BE FOLLOWED BY A STRUCTURE

SDA0425 VALID OUTPUT POSITION SPECIFICATION FOR THIS OPERAND MANDATORY

SDA0426 SELECTED OPERAND OR VALUE DOES NOT MATCH REQUIRED TYPE OF OPERAND OR VALUE

SDA0427 NO VALUE CAN BE SPECIFIED FOR ’DEVICE’ DATA TYPE

SDA0428 WARNING: OPERAND DECLARED ’PRESENCE=INTERNAL-ONLY’ WHICH INTRODUCES A STRUCTURE CANNOT BE DEFINED IN LIST. ’PRESENCE’ WAS RESET TO ’NORMAL’

SDA0430 VALUE ’(&00)’ VIOLATES IMPLEMENTATION BOUNDARIES

Meaning
Each data type has limit values which are output in guided dialog (e.g. C_string 1..1800).

SDA0431 INVALID DEFAULT VALUES. STATEMENT IGNORED

SDA0432 TRANSFER AREA TOO SMALL. STATEMENT IGNORED

SDA0433 OPERAND DECLARED ’PRESENCE=INTERNAL-ONLY’ MUST HAVE DEFAULT VALUE

SDA0434 IF DATA TYPE IS CHANGED TO ’INTEGER’ OR ’TIME’, REPRESENTATION ’OUT-FORM=’ IS MANDATORY

SDA0435 STRUCTURE MUST NOT FOLLOW ’TEXT’ OR ’COMMAND-REST’ DATA TYPE

SDA0436 LOWEST LIMIT GREATER THAN HIGHEST LIMIT

Meaning
Possible reasons:
- SHORTEST-LENGTH is greater than LONGEST-LENGTH (for example: 10..2 is wrong, 2..10 is correct).
- LOWEST-VALUE is greater than HIGHEST-VALUE (for example: (2,-2) is wrong, (-2,2) is correct).

SDA0437 OPERAND '(&00)' IN COMMAND '(&01)' HAS INVALID RESULT OPERAND NAME. CORRECTION TO VALUE '*POS(1)'

SDA0438 START OF PROCESSING OF OBJECT '(&00)'

SDA0439 'OUTPUT=DROP-OPERAND' NOT PERMITTED FOR VALUES WITH 'LIST-ALLOWED=YES'

SDA044A WARNING: THE INTERNAL NAME OF THE COMMAND SHOULD START WITH '(&00)' OR '(&01)'

Meaning
(&01) = command class (specified at KPSD creation).

SDA0440 NO GLOBAL INFORMATION AVAILABLE IN CURRENT SYNTAX FILE. TASK TERMINATED ABNORMALLY

SDA0441 SPECIFICATION OF STRUCTURE FOR OPERAND '(&00)' AMBIGUOUS

SDA0442 RECURSIVE DEFINITION OF VALUES OR OPERANDS NOT PERMITTED

Meaning
When defining the value 'v2' or the operand 'op2' in the structure 'op1=v1(op2=v2)', it is not permissible to use 'ATT-INFO=YES' to copy the value 'v1' or the operand 'op1'.

Response
Use the SDF-A statement //ADD-VALUE or //ADD-OPERAND to define the value 'v2' or the operand 'op2'.

SDA0443 /SEND-MESSAGE (/INTR) FOLLOWS K2 INTERRUPT : STATEMENT CANCELLED

SDA0444 INTERNAL NAME OF APPLICATION DOMAIN '(&00)' FOR '(&01)' DOES NOT EXIST

SDA0445 WARNING: LIST SPECIFICATION FOR OPERAND '(&00)' DELETED BECAUSE NO CORRESPONDING VALUE PERMITTED IN LIST

SDA0446 WARNING: CURRENT SYNTAX FILE NOT CLOSED PROPERLY WHEN LAST PROCESSED

Meaning
After the previous processing, the program run was terminated abnormally.
Check the consistency of the syntax file currently used.
An update of the syntax file is possible.
The error indicator will be reset after a successful program run via the END statement.

SDA0447 FILE NAME '(&00)' SPECIFIED IN //OPEN-SYNTAX-FILE STATEMENT ALREADY EXISTS

Meaning
The file name (&00) specified in the statement has already been specified.

SDA0448 LIST OF SINGLE VALUES NOT PERMITTED FOR 'KEYWORD' DATA TYPE

SDA0449 COPYING OF OBJECTS WITHIN SAME SYNTAX FILE NOT POSSIBLE. STATEMENT REJECTED

Meaning
In the //COPY statement, it is necessary to specify a syntax file which is different from the current syntax file.

Response
Specify two different file names as syntax files.

SDA0450 `WARNING: INCORRECT 'LONGEST-LENGTH' FOR FULL-FILENAME OR PARTIAL-FILENAME SPECIFIED`

Meaning
The LONGEST-LENGTH for a full-filename is defined as follows:
- with CAT-ID=YES and USER-ID=YES, LONGEST-LENGTH=54
- with CAT-ID=NOand USER-ID=YES, LONGEST-LENGTH=48
- with CAT-ID=YES and USER-ID=NO,LONGEST-LENGTH=44
- with CAT-ID=NOand USER-ID=NO,LONGEST-LENGTH=38.

The LONGEST-LENGTH for a partial-filename is defined as follows:
- with CAT-ID=YES and USER-ID=YES, LONGEST-LENGTH=53
- with CAT-ID=NOand USER-ID=YES, LONGEST-LENGTH=47
- with CAT-ID=YES and USER-ID=NO,LONGEST-LENGTH=43
- with CAT-ID=NOand USER-ID=NO,LONGEST-LENGTH=37.

Response
Automatic correction which inserts a correct value for the LONGEST-LENGTH operand.

SDA0451 `WARNING: ALIAS NAME SUPPRESSED BECAUSE DUPLICATE OF MAIN OR STANDARD NAME`

SDA0452 `MODIFICATION OF 'NON KEYWORD' VALUE INTO 'KEYWORD' NOT PERMITTED`

SDA0453 `DEFINITION OF SDF GLOBALS NOT PERMITTED`

Meaning
Duplicates in the next fields' names (CONTINUE, TEST, EXECUTE, ...) are not permitted.

SDA0454 `OPTION 'OVERWRITE-POSSIBLE=YES' ONLY PERMITTED FOR OPERANDS WITH DEFAULT VALUE`

SDA0455 `WARNING: OUTPUT AREA TOO SHORT TO CONTAIN POSSIBLE INPUTS`

SDA0456 `OUTPUT AREA TOO SHORT TO CONTAIN INPUT VALUE`

SDA0457 `FOR ONE OPERAND ONLY ONE VALUE WITH 'NULL ABBREVIATION' IS PERMITTED`

SDA0458 `NO CURRENT OBJECT PRESENTLY DEFINED`

SDA0459 `ERROR IN //COPY STATEMENT`

SDA0460 `VALUE '(&00)' IS DEFAULT VALUE FOR OPERAND '(&01)'. DELETION NOT PERMITTED`

SDA0461 `WARNING: THE HASHING FUNCTION IS ONLY PERMITTED FOR C-STRING VALUES`

SDA0463 `LONGEST-LOGICAL-LENGTH INCORRECT`

Meaning
The longest logical length cannot be less than the minimum length or greater than the maximum length.

SDA0464 `WARNING: RECOVERY OF PARTIAL-FILENAME BY FILENAME WITH WILDCARDS`

Meaning
The datatype <partial-filename> may be suppressed because it is completely included in the datatype <filename with- wild>.

SDA0469 `ITEM CURRENTLY NOT DEFINED IN REFERENCE SYNTAX FILES`

Meaning
The input, which contains a command, a statement, an operand or a value, is not defined in the reference syntax files.

SDA0470 `INTERNAL PROGRAM NAME '(&00)' DOES NOT EXIST IN CURRENT SYNTAX FILE`

SDA0471 `HELP TEXT DOES NOT EXIST FOR SPECIFIED LANGUAGE`

SDA0472 `DEFINITION OF SYSTEM COMMAND WITH 'IMPLEMENTOR=TPR' NOT PERMITTED IN USER SYNTAX FILE`

SDA0474 `//EDIT STATEMENT NOT PERMITTED FOR THIS OBJECT`

Meaning
This function is reserved for the system administrator.

SDA0475 `//COPY STATEMENT NOT PERMITTED FOR THIS OBJECT`

Meaning
It is not permitted to copy an old command into a user syntax file.
It is not permitted to copy a common statement into a system or a group syntax file.

SDA0476 `WARNING: FOR GLOBAL INFORMATION, PROGRAMS AND DOMAINS, 'ATTACHED-INFO=ONLY' VALUE IS INTERPRETED AS 'ATTACHED-INFO=YES'`

SDA0477 `NO VALUE PRESENTLY DEFINED FOR PROCEDURE FILE NAME`

Meaning
The name of the procedure that implements the command is not defined : either a file name has not been given or some IMON information has not been given yet.

SDA0478 `OBJECT '(&00)' IS NOT FLAGGED 'REMOVED'. //RESTORE STATEMENT REJECTED`

Meaning
Possible reasons:
– the specified object is not flagged REMOVED.
– the specified object corresponds to the abbreviation of many objects which are not flagged REMOVED.

SDA0479 THE NAME OF THE INPUT-FILE IS THE SAME AS THE NAME OF THE COMPARE-FILE

SDA0480 VALUE '(&00)' DOES NOT MATCH CORRESPONDING SYNTAX TYPE BECAUSE INPUT OF BLANKS IS NOT PERMITTED

SDA0481 WARNING: 'STRUCTURE-IMPLICIT=YES' NOT PERMITTED FOR OPERAND '(&00)'. 'STRUCTURE-IMPLICIT' RESET TO 'NO'

Meaning
STRUCTURE-IMPLICIT=YES is only possible for operands of structures which are introduced by a keyword or a keyword number.

SDA0482 WARNING: 'ANALYSE-DEFAULT' WAS RESET TO 'NO' FOR OPERAND '(&00)'

Meaning
- Due to previous error.
- For Device Data type.
- For Secret value wanted.
- For default value containing a list or a structure.

SDA0483 'OVERWRITE-POSSIBLE=YES' NOT PERMITTED FOR OPERANDS AND VALUES

SDA0485 'EXTERNAL-ATTRIBUTES' VALUE ONLY VALID FOR PROGRAMS OR DOMAINS

SDA0486 EXTERNAL DOMAIN OR PROGRAM NAME '(&00)' ALREADY ASSIGNED

SDA0487 //RESTORE STATEMENT IN SYSTEM SYNTAX FILE NOT POSSIBLE

Meaning
An object removed from the system syntax file really has been deleted; a restoration is not possible.

SDA0488 OBJECT '(&00)' NOT REMOVED. //RESTORE STATEMENT REJECTED

Meaning
The specified object is not removed. It corresponds to the abbreviation of already existing objects, which are not removed.

SDA0489 OBJECT '(&00)' CANNOT BE RESTORED IN USER SYNTAX FILE

Meaning
The object defined in the system syntax file was removed when the group syntax file was opened.

Response
Open the group syntax file in order to restore this object.

SDA0490 SYNTAX FILE '(&00)' WAS NOT CLOSED CORRECTLY IN PREVIOUS PROCESSING. FILE IS CORRUPTED AND CAN NO LONGER BE USED

SDA0491 OVERWRITING OF AN OBJECT DEFINED IN HIGHER LEVEL OF HIERARCHY NOT POSSIBLE

SDA0492 WARNING: ERROR ON 'STXIT' INITIALISATION. /INTR COMMAND REJECTED

SDA0493 `ERROR ON STXIT INITIALISATION : ABEND/K2 CAN NOT BE USED. DO NOT INTERRUPT THE PROGRAM AT THIS POINT`

SDA0494 `THE OVERWRITE OF COMMON STATEMENTS IS ONLY POSSIBLE FROM SDF-U PROGRAM`

SDA0495 `THE TPR-COMMAND '(&00)' DOESN'T EXIST OR IS NOT A TPR-COMMAND`

SDA0496 `WARNING : THE RESULT-INTERNAL-NAME MAY NOT BE DEFINED WITH IMPLEMENTOR=*BY-TPR`

SDA0497 `NO VALUE PRESENTLY DEFINED FOR TPR COMMAND NAME`

SDA0499 `WARNING: THERE IS NO OBJECT CORRESPONDING TO THE SPECIFIED INPUT`

SDA0507 `FILE '(&00)' IS NOT A SYNTAX FILE OR FILE TYPE INVALID`

SDA0508 `SYNTAX FILE '(&00)' NOT CLOSED CORRECTLY`

SDA0509 `'(&00)' IS NOT A NEW KERNEL SYNTAX FILE. INPUT IGNORED`

SDA0510 `VERSION VALUE '(&00)' IGNORED BECAUSE '(&01)' IS NOT A COMPONENT SYNTAX FILE. PROCESSING CONTINUES`

Meaning
The version specified in the //OPEN-SYNTAX-FILE statement has not been found. The specified file is not a component syntax file.

SDA0511 `TERMINATE THE PROGRAM BY '//END' ELSE THE LAST OBJECT MAY BE INCOMPLETE IN THE SYNTAX FILE`

Meaning
Return to the program to complete the last cmd/stmt processed by a CLOSE-CMD-STMT or END otherwise the file will be rejected by SDF.

SDA0512 `THE PROCESSED SYNTAX FILE IS IN AN INCONSISTENT STATE`

Meaning
The syntax file will be rejected by SDF. Open it again in update mode and try to correct the last processed objects.

SDA0571 `TABLE OF COMMAND NAMES DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0572 `TABLE OF NAMES OF APPLICATION DOMAINS DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0573 `TABLE OF PROGRAM NAMES DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0574 `TABLE OF ALL PRIVILEGES DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0575 `STATEMENT FOR PROGRAM '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0576 `MCLP ERROR '(&00)'. COMMAND NOT PROCESSED`

Meaning
For more detailed information about the MCLP error code, see the description of the CMD macro in the BS2000 manual 'Executive Macros'.
MCLP: Macro Command Language Processor.

SDA0580 `NAME OF APPLICATION DOMAIN '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0581 `GLOBAL INFORMATION DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0582 `COMMAND '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0583 `STATEMENT '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0584 `V-RECORD DOES NOT EXIST IN SYNTAX FILE`

SDA0585 `DATA TYPE '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0586 `PROGRAM '(&00)' DOES NOT EXIST IN ACTIVE SYNTAX FILE`

SDA0600 `INTERNAL NUMBER '(&00)' ALREADY ASSIGNED TO ANOTHER PRIVILEGE`

Meaning
Each privilege is internally identified by the positioning of the corresponding number in the 64 bits string representing the privileges. Therefore the position of the number must be different for each privilege.

Response
Change the internal number assigned to the privilege.

SDA0601 `INCONSISTENCY IN PRIVILEGE DEFINITION`

Meaning
In a command or statement definition, a node (operand or value) is defined with more privileges than its 'father' nodes. The privileges for the operands and values of the lowest level cannot be further modified as one of these operands or values has no more privilege assigned to it.

Response
Check privileges assigned to each command node (command, operand or value) and change inconsistencies.

SDA0602 `WARNING: INCONSISTENCY IN PRIVILEGE DEFINITION FOR OPERAND '(&00)'`

Meaning
It must be possible to give a value to the operand for every task, whatever its privilege(s). Therefore the default value must have the same privilege(s) as the operand.
If the operand has no default value, then the operand must have the same privilege(s) as its father node (command or value node), and for each privilege assigned to the operand, there must be at least one value with this privilege.

SDA0603 `//REMOVE STATEMENT NOT PERMITTED FOR THIS OBJECT`

Meaning
Removal of a privilege is only permitted in the system kernel file. This function is reserved for the privilege administrator.

SDA0605 `MINIMAL-ABBREVIATION '(&00)' NOT ABBREVIATION OF MAIN NAME '(&01)'`

Meaning
The minimal abbreviation must be an abbreviation of the main name (&01) defined with the NAME operand.

Response
Use an unambiguous abbreviation.

SDA0606 `WARNING: NAME '(&00)' SHORTER THAN REQUIRED MINIMAL-ABBREVIATION '(&01)'. INPUT IGNORED`

Meaning
A shorter input than the minimal abbreviation can be unequivocal, but will not be executed for security reasons.

Response
Use an unambiguous abbreviation.

SDA0607 `MINIMAL-ABBREVIATION '(&00)' AMBIGUOUS WITH REGARD TO AN OPERAND NAME OR VALUE`

Meaning
The minimal abbreviation is ambiguous regarding the main, alias or standard name of another operand or value of the same level.

Response
Use an unambiguous abbreviation.

SDA0608 `WARNING: 'NULL-ABBREVIATION' RESET TO 'NO' BECAUSE 'MINIMAL-ABBREVIATION' IS GIVEN`

Meaning
NULL-ABBREVIATION=YES cannot be used when a previous MINIMAL-ABBREVIATION has been defined.

SDA0609 `WARNING: 'PRIVILEGE DEFINITION' RESET BECAUSE PRIVILEGES ASSOCIATED WITH OBJECT TO BE COPIED DO NOT EXIST IN HIERARCHY`

Meaning
For command or statement definition, the operands and values must have the same privilege as the command or statement node.

SDA0610 `WARNING: 'PRIVILEGE DEFINITION' RESET BECAUSE THE OBJECT TO BE COPIED IS DEFINED WITH MORE PRIVILEGES THAN ITS 'FATHER' NODE`

Meaning
The privilege definitions of the object and of the operands and values of the lowest level are reset to the same privilege as its 'father' node.

8.2 SDF-SIM messages

SDS0001 `SDF-SIM VERSION '(&00)' STARTED`

SDS0002 `SDF-SIM TERMINATED NORMALLY`

SDS0003 `SDF-SIM TERMINATED ABNORMALLY`

SDS0004 `INVALID SYNTAX FILE FOR OPERAND '(&00)'`

SDS0005 `'(&00)' SYNTAX FILE '(&01)' ACTIVATED`

SDS0006 `PROGRAM '(&00)' IS NOT DEFINED IN THE ACTIVATED SYNTAX FILES`

SDS0007 `ERROR '(&00)' RETURNED BY '(&01)' CALL`

SDS0008 `DO YOU WANT TO DISPLAY TRANSFER AREA? REPLY: NO/LONG/SHORT`

SDS0009 `ENTER THE NAME OF THE OML TO USE FOR ENTRY '(&00)' OR *NO`

Meaning
The user has specified EXECUTION=*YES (in DEFINE-ENVIRONMENT) and SDF-SIM has not been able to find the entry corresponding to a command. SDF-SIM therefore asks the name of the OML which contains this entry then links it to the simulation phase.

SDS0010 `INVALID OML NAME`

SDS0011 `SDF INTERNAL ERROR: ERROR '(&00)', RETURNED BY MACRO '(&01)', 'SERSLOG' LEVEL`

SDS0012 `DO YOU WANT TO DISPLAY STACK? REPLY: YES/NO`

Meaning
An SDF internal error has occurred. Displaying the stacks (ABNORM routine) can facilitate diagnosis.

SDS0013 `SDF INTERNAL ERROR: ERROR '(&00)', RETURNED BY INTERFACE NUMBER '(&01)', 'TERMINATE' LEVEL`

9 Appendix

9.1 Changes to the SDF program interface

The layout of the standardized transfer area was modified for SDF V4.1, and for this reason the CMDSTRUC macro for generating the transfer area was replaced by the CMDTA macro. Additionally, the macros RDSTMT, TRSTMT and CORSTMT used in connection with the statements were replaced by the new macros CMDRST, CMDTST and CMDCST.

The format used for the standardized transfer are up to V4.0 and the macros CMDSTRUC, RDSTMT, TRSTMT and CORSTMT are still supported for reasons of compatibility. However, they should no longer be used in new programs.

9.1.1 Format of the standardized transfer area up to SDF V4.0

The standardized transfer area begins on a word boundary. The header field is in bytes 0 through 19. It contains, amongst other things, the internal statement name (see ADD-STMT ...,INTERNAL-NAME=...). The array for operands valid for all statements, i.e. for operands defined with RESULT-OPERAND-LEVEL=1 (see ADD-OPERAND), begins with byte 20. It accommodates a 6-byte description for each of these operands. The operand descriptions are arranged in the order resulting from the operand positions established in the statement definition (see ADD-OPERAND ...,RESULT-OPERAND-NAME=*POSITION (POSITION=<integer>)). Each operand description contains, among other things, the absolute address where the associated operand value, or the description of the associated list or non-linearized structure, is stored in the transfer area. The description of a non-linearized structure contains an operand array describing the operands contained in the structure. It has the same format as the array for the operands valid for all statements.

In the simplest case, an operand has only one simple value (Figure 8 without standard header, page 369).

If the operand value introduces a structure (Figure 9 without default header, page 368), there is a structure description for this value. This contains an operand array with descriptions for all operands of the structure, as well as for the operands from linearized substructures. The operand descriptions are arranged in the order resulting from the structure-oriented operand positions established in the statement definition (see ADD-OPERAND...,RESULT-OPERAND-NAME=*POSITION (POSITION=<integer>)). The operand values corresponding to the operands of the structure may introduce further structures and/or consist of a list of values.

Values for operands defined with ADD-OPERAND ...,LIST-POSSIBLE=*YES (...,FORM=*NORMAL) are transferred in the form shown in Figure 10 (page 369, without default header). A structure may be attached to a list element. In this case, “value of the list element” is a structure description.

Header field of a standardized transfer area

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1	Length of the transfer area (maximum 65536 bytes)	Program	unchanged	Program
2 to 9	Internal name of the statement	SDF	unchanged	Program
10 to 17	Reserved	–	–	–
18 to 19	Number of positions in the operand array	SDF	unchanged	Program

The internal name of the statement and the number of positions in the operand array are stored in the syntax file in the statement definition
(see ADD-STMT...,INTERNAL-NAME=...,MAX-STRUC-OPERAND=...).

Description of operands

The operand array and the descriptions therein for the operands of a structure have exactly the same format as the one for the operands valid for all statements.

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1 0 1	Value description Additional information Description of type	See below SDF	See below unchanged	See below Program [1)]
2 to 5	Absolute address (stored unaligned) of the value assigned to the operand or, in the case of structures or lists, of the further description	SDF	unchanged	Program [1)]

[1] Entry only for operands for which there are values to be converted, i.e. when bit 0 of the additional information is set

Additional information

The additional information is contained in the first byte of the value description. The following specifications regarding the additional information apply regardless of whether the additional information appears in an operand description, in the header field of a structure description, in the description of a list element or in an OR list description.

Bit	Value	Meaning	Source of field contents in case of		
			analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0	0	Value not available	SDF [1)]	unchanged	Program [2)]
0	1	Value available	SDF	unchanged	Program [2)]
1	0	Value changeable	–	Program[3)]	–
1	1	Value not changeable	–	Program[3)]	–
2	0	Value not errored	–	Program	–
2	1	Value errored	–	Program	–
3	0	Value is not to be used as default value	–	–	Program
3	1	Value is to be used as default value	–	–	Program
4 to 7	–	Reserved	–	–	Program

1 For example: values for operands defined with ADD-OPERAND..., PRESENCE= *EXTERNAL-ONLY, or values in structures not referenced.

2 0 for operands for which there are neither operand values to be converted nor structures containing operands with values to be converted.
1 for operands for which there are either operand values to be converted or structures containing operands with values to be converted.

3 All list values coming after the first changeable list value are considered changeable by SDF, regardless of whether bit 1 is set. In this way, list elements that have already been processed are protected against being overwritten.

Type description

The type description is contained in the second byte of the value description. The following specifications regarding the type description apply regardless of whether the type description appears in an operand description, in the header field of a structure description, in the description of a list element or in an OR list description.

Structure descriptions can be entered by a program in order to specify custom defaults. Defaults can be entered:

– in the internal format (like the OUTPUT operand in the SDF-A statement ADD-VALUE) or

– in the external format as a string analogous to the operand description. The auxiliary data type <input-text> must be used, and the value is analyzed as if it had been entered via the user interface.

Value (decimal)	Meaning
1	Command rest
2	Integer
4	X-string
5	C-string
6	Name
7	Alphanumeric name
8	Structured name
9	Label
11	Fully qualified file name
12	Partially qualified file name
13	Time
14	Date
15	Composed name

Continued ➠

Value (decimal)	Meaning
16	Text
17	Catalog identifier (cat-id)
18	Input text
19	Structure
20	List
21	OR list
22	Keyword
23	Reserved for internal use
24	VSN
25	X-text
26	Fixed-point number
27	Device
28	Product version
29	POSIX pathname
35	POSIX file name

Header field of a structure description

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1	Number of positions in the operand array	SDF	unchanged	Program
2 to 3	Value description for the operand value introducing the structure			
2	Additional information	–	See above	See above
3	Type description	SDF	unchanged	Program
4 to 7	Absolute address (stored unaligned) of the operand value introducing the structure	SDF	unchanged	Program

The number of positions in the operand array is stored in the statement definition in the syntax file (see ADD-VALUE...,STRUCTURE=*YES(..,MAX-STRUC-OPERAND=...).

The operand array belonging to the structure begins immediately after the header field. It has exactly the same format as the one for operands valid for all statements.

List element

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	errored statement back to SDF	default values to SDF
0 to 1 0 1	Value description Additional information Description of type	 See above SDF	 See above unchanged	 See above Program[1)]
2 to 5	Absolute address (stored unaligned) of the value assigned to the list element or, in the case of structures, of the further description	SDF	unchanged	Program[1)]
6 to 9	Absolute address (stored unaligned) of the next list element	SDF	unchanged	Program[1)]

[1] Specified only for operands with operand values to be converted, i.e. when bit 0 of the additional information is set

In the last element of the list, the “absolute address of the next list element” has the value 0.

An OR-list consists of a single element. The “absolute address of the next list element” is redundant in this case.

For an operand defined with LIST-POSSIBLE=*YES(FORM=*NORMAL), the number of list elements must be restricted (LIMIT=...) so as to prevent an overflow in the standardized transfer area. The size of a list in the standardized transfer area can be calculated by using the following formula:

$n * (10 + 2 + l)$

where:	n	is the number of list elements, and
	l	is the length of a single list element (rounded to a multiple of 2).

Example:

```
ADD-OPERAND ... LIST-POSSIBLE=*YES(LIMIT=100,FORM=*NORMAL)
  ADD-VALUE *NAME(1,8)
```

A list that is defined in this way can occupy up to 2000 bytes in the standardized transfer area:
(100 * (10 + 2 + 8) = 2000).

How values are stored

Byte	Contents	Source of field contents in case of		
		analyzed statement to program	**errored statement back to SDF**	**default values to SDF**
0 to 1	Length specifications	SDF	unchanged	Program
2 to...	Value	SDF	unchanged	Program

How the values are passed depends on the definition in the syntax file (see ADD-VALUE..., OUTPUT=*NORMAL(...). In this regard, the following points apply:

– A value defined with ADD-VALUE TYPE=*INTEGER(...,OUT-FORM=*BINARY) is stored as a signed four-byte string.

– A value defined with ADD-VALUE TYPE=*TIME is stored as a four-byte string, with 2 bytes (binary) for the hours and one byte each for the minutes and seconds.

9.1.2 CMDSTRUC Generate transfer area for analyzed statement

The CMDSTRUC macro generates a DSECT. This defines the standardized transfer area up to SDF V4.0. This is needed for the following three purposes:

1. SDF passes an analyzed statement to the program (see CORSTMT, RDSTMT and TRSTMT)
2. The program returns a semantically incorrect statement to SDF (see CORSTMT)
3. The program passes values to SDF that are to replace specified operand values (see RDSTMT and TRSTMT).

The standardized transfer area is described in detail in section "Format of the standardized transfer area up to SDF V4.0" on page 593ff.

Operation	Operands
CMDSTRUC	[P = <u>CMD</u> / prefix]

P = <u>CMD /</u> prefix
specifies a character string that is to be concatenated with the beginning of all names in the DSECT. It may be up to three characters long. Unless otherwise specified, the string "CMD" will be used.

9.1.3 CORSTMT Initiate semantic error dialog

The CORSTMT macro causes SDF to conduct a dialog with the user in which the user corrects semantic errors in a statement. Immediately beforehand, SDF has analyzed the statement and passed it to the program as being syntactically correct.
Secret operand values that were entered in blanked input fields by the user must be repeated during the correction process.

Prerequisites for the semantic error dialog are:

- The program is running in an interactive task and an error dialog was permitted in the syntax analysis, i.e.:
  - temporary or permanent guided dialog must be set
  - the SDF option PROCEDURE-DIALOGUE=YES must be set in procedures
  - if CORSTMT is called after TRSTMT, DIALOG=ERROR must be set for TRSTMT
- The same syntax file is available as when the statement was first analyzed (no intervening change of syntax files).

If these prerequisites are not satisfied, SDF rejects the dialog (error code X’20’).

Operation	**Operands**
CORSTMT	INOUT = addr / (r1)
	,MESSAGE = addr / (r3)
	[,DEFAULT = *NO / (addr,...)]
	[,INVAR = *NO / addr / (r)]
	[,CALLID = *NO / addr / (r7)]
	[,CCSNAME = *NO / *EXTEND / name]
	[,MF = $\left\{\begin{array}{l}\text{L}\\ \text{(E,(1))}\\ \text{(E,opaddr)}\end{array}\right\}$]

INOUT=addr / (r1)
Address of the standardized transfer area, or a register that contains this address. The area must begin on a word boundary. It contains the results of analysis of the incorrect statement, passed previously to the program by SDF. The program has identified the operand values it has found to be errored. SDF does not accept changes in input values made by

the program. Following the error dialog and renewed analysis, SDF stores the modified analysis results back into this area (see section “Format of the standardized transfer area up to SDF V4.0” on page 593).

MESSAGE=addr / (r3)
Address of the text to be output for the error dialog, or a register that contains this address. In guided dialog, this text is integrated into the statement menu. The text is expected in the form of a variable-length record, and may occupy up to four lines on the terminal. If the text contains screen control characters, the menu mask may be destroyed.
This area must be aligned on a halfword boundary.

DEFAULT=
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
– operand values entered or
– operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES,... and ADD-VALUE...,VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value.
In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:

In the MODIFY statements entered, SDF-A replaces the value UNCHANGED by the current value. If the operands to be defaulted are in a structure whose introductory value is defined by LIST-ALLOWED=*YES (see ADD-VALUE), the following may occur:
The conversion description contains several list elements to which a structure with operands to be defaulted is attached. At the user end, the user also enters several list elements to which a structure with operands to be defaulted is attached. SDF attempts first to match the structures entered by the user to those specified in the conversion description via the value that introduces the structure. If this value does not allow an unambiguous allocation because none of the values entered matches any of those in the conversion description, or because the user has entered the matching value several times, the allocation is made via the position in the list of the value introducing the structure.

***NO**
The operand values entered are not replaced by values generated dynamically by the program.

(addr,...)
In one or more of the possible statements, operand values entered are to be replaced by values generated dynamically by the program. The conversion descriptions for these statements are located at the specified addresses (addr21,...) in the program. A conversion description contains, among other things, the internal statement name and information regarding which of the operand values entered are to be changed and what values they are to be changed to.
The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the user-defined operands, all operands having default values and all operands currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement available to a user who has certain privileges and who is working in the selected dialog mode. In contrast to LOGGING=*INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form *does not* mask out keywords and secret operands.

<u>*NO</u>
The INVARIANT-INPUT form of the statement is not stored.

addr / (r)
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword (HW) must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword. The contents of the buffer are then as follows:

1st HW	2nd HW	3rd HW	
buflen	reclen	filler	invariant-input

buflen:	Length of the buffer
reclen:	Length of the record written by SDF
filler:	Filler
invariant-input:	INVARIANT-INPUT form of the statement, starting at the seventh byte

CALLID =
Refers to a context (= syntax file hierarchy) which was opened by an OPNCALL macro. The name of the syntax file hierarchy (callid) must have the 4-byte value returned by SDF to the field which was designated by the CALLID operand in the Open Context macro. This function applies to the OPNCALL and CLSCALL macros.

***NO**
The current syntax file hierarchy (context) of the task is used for analyzing the statement.

addr /(r7)
Address of the call check field or register containing this address.
The area must be aligned on a word boundary.

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the “XHCS” manual [11]). This parameter has no effect on message output.

***NO**
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

name
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long.

MF =
Defines special requirements for macro expansion (the “Executive Macros” manual [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This either appears in register 1 or is specified directly.

Return information and error flags

The format of the transfer area is described on page 593ff.

Register 15 contains a return code in the right-aligned byte. EQUATE statements for this can be generated with the CMDANALY macro.

X’00’	Normal termination
X’04’	Unrecoverable system error
X’08’	Operand error in the macro
X’0C’	Transfer area too small
X’10’	End-of-file (EOF), or error in statement, end-of-file (EOF) was then detected
X’14’	Error in statement, a command was then detected
X’18’	Statement is correct but the default values provided by the system are errored
X’1C’	Error in statement, STEP was then detected
X’20’	Error dialog not possible
X’24’	Error dialog rejected by user (Exit function activated)
X’2C’	END statement has been read
X’34’	Error in statement, END was then detected
X’38’	SDF not available
X’44’	Syntax file not found
X’4C’	The program is not executable above the 16-Mbyte boundary, since SDF is not loaded. Please notify the system administration.
X’5C’	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
X’64’	XHCS error

9.1.4 RDSTMT Read and analyze statement

The RDSTMT macro causes SDF to

- read in a program statement from SYSSTMT (For the system file SYSSTMT the same assignment applies as was made for the system file SYSDTA. With regard to continuation lines, continuation characters and notes, the same rules apply to statement input from SYSSTMT as to command input from SYSCMD.)
- analyze the statement read in, and
- pass the results of the analysis to the program.

This presupposes that an activated syntax file contains the definition of the program and its statements. The input length for a statement read via RDSTMT is 16364 bytes.

Operation	**Operands**
RDSTMT	PROGRAM = name
	,OUTPUT = addr / (r1)
	[,STMT = <u>*ALL</u> / (name,...) / *ADDR(addr/(r))]
	[,PREFER = <u>*ALL</u> / name / *ADDR(addr/(r))]
	[,DEFAULT = <u>*NO</u> / (addr,...)]
	[,MESSAGE = <u>*NO</u> / addr / (r3)]
	[,PROT = <u>YES</u> / NO]
	[,BUFFER = <u>*NO</u> / addr / (r)]
	[,INVAR = <u>*NO</u> / addr / (r)]
	[,SPIN = <u>NO</u> / YES]
	[,ERRSTMT = <u>STEP</u> / NEXT]
	[,CALLID = <u>*NO</u> / addr / (r7)]
	[,CCSNAME = <u>*NO</u> / *EXTEND / name]
	[,MF = $\left\{\begin{array}{l}\text{L}\\ \text{(E,(1))}\\ \text{(E,opaddr)}\end{array}\right\}$]

PROGRAM = name
Internal name of the program that generates the macro. This name is stored in the program definition in the syntax file (see ADD-PROGRAM). It is at least one byte and at most eight bytes long.

OUTPUT = addr / (r1)
Address of the standardized transfer area, or a register that contains this address. The area must begin on a word boundary. Prior to calling the RDSTMT macro the program must ensure that the first two bytes of this area contain the maximum length possible for the area (see section “Format of the standardized transfer area up to SDF V4.0” on page 593ff). In it, SDF stores the analysis results

STMT =
specifies which statements are permitted as input.

***ALL**
All statements are permitted.

(name,...)
Only the statements whose internal names are specified are permitted. The internal statement name is stored in the statement definition in the syntax file (see ADD-STMT). It is at least one byte and at most eight bytes long.
The standard SDF statements are always permitted, regardless of the specification made here.

***ADDR(addr/(r))**
Address of the list of permissible statements. This list must be generated beforehand with the CMDALLW macro.

PREFER =
Relevant only for guided dialog; specifies whether a particular statement is expected as the next input.

***NO**
No particular statement is expected. SDF asks the user via the statement menu which statement is to be entered.

name
Internal name of the statement most likely to be entered. SDF does not display a statement menu in which the user selects the statement to be entered, but instead immediately displays the form listing the operand values for the expected statement. The user may of course enter another statement instead of the one expected.

Example:

Following MODIFY-OPERAND, SDF-A expects MODIFY-VALUE as the next statement. The internal statement name is stored in the statement definition in the syntax file (see ADD-STMT). It is at least one byte and at most eight bytes long.

***ADDR(addr/(r))**
Address of an area, 8 bytes long, containing the internal name of the expected statement. The name must be left-justified and padded with blanks as necessary (X'40').

DEFAULT =
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
– operand values entered or
– operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD-OPERAND...,OVERWRITE-POSSIBLE=*YES,... and ADD-VALUE...,VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value.
In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:

In the entered MODIFY statements SDF-A replaces the value *UNCHANGED by the current value.
If the operands to be given default values are in a structure introduced by a value defined with LIST-ALLOWED=*YES (see ADD-VALUE), the following situation may arise: The conversion description contains several list elements to which structures with operands to be defaulted are attached. On the other hand, the user likewise enters several list elements to which structures with operands to be defaulted are attached. SDF first tries to match the structures entered by the user to those specified in the conversion description by means of the values introducing the structures. If an unambiguous allocation cannot be made on the basis of the values introducing the structures because none of the values entered matches any of the ones in the conversion description or because the user has entered the matching value more than once, the allocation is then made on the basis of the list position of the introductory value.

<u>*NO</u>
SDF is not to replace the entered operand values by values generated dynamically by the program.

(addr,...)
In one or more of the possible statements, SDF is to replace entered operand values by values generated dynamically by the program. The conversion descriptions for these statements (see section “Format of the standardized transfer area up to SDF V4.0” on page 593ff) are located at the specified addresses (addr21,...) in the program. Only one conversion description can be specified per statement. A conversion description contains, among other things, the internal statement name and information as to which of the operand values entered are to be changed and what values they are to be changed to. The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

MESSAGE =
specifies whether SDF is to issue a message when requesting statement input. In guided dialog this message is integrated into the statement menu.

***NO**
SDF is not to issue a message.

addr / (r3)
Address of the message text to be issued, or a register that contains this address. The text is expected in the form of a variable-length record with a maximum length of 400 characters. However, only the first 80 characters are displayed on SDF-formatted screens. If the text contains screen control characters, the menu mask may be destroyed. This area must be aligned on a halfword boundary.

PROT =
Specifies whether SDF is to log input and messages to SYSOUT. If they are not written to SYSOUT, the user of the program should be informed of this in the program documentation. In contrast to the TRSTMT macro, this parameter is a flag (set=NO, reset=YES). No logging buffer is available.

YES
SDF is to log input and messages to SYSOUT.

NO
SDF is not to perform any logging. The following behavior may be expected:

Result of analysis		
	PROT=YES	**PROT=NO**
No error	Input statement	– / –
Syntax error	1. Input statement 2. Syntax error message 3. Spin-off message	Spin-off message

BUFFER =
The statement log and the error messages can be written into an area provided by the user.

<u>NO</u>
No buffer area is provided.

addr / (r)
Address of an area or a register in which the log of the entered statements and the messages are written, regardless of what was specified for PROT. The area must be aligned on a word boundary. The first halfword must contain the total length of the area. SDF writes the actual length of the output log to the second halfword. The log records are then written to the area as variable-length records.
If the buffer is not empty, the first record is generally the log of the input command. Subsequent records contain messages. If no input log is available, or if the input log cannot be output, two slashes (“//”) are written into the output area.

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the user-defined operands, all operands having default values and all operands currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement available to a user who has certain privileges and who is working in the selected dialog mode. In contrast to LOGGING=INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form *does not* mask out keywords and secret operands.

<u>*NO</u>
The INVARIANT-INPUT form of the statement is not stored.

addr / (reg)
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword (HW) must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword. The contents of the buffer are then as follows:

1st HW	2nd HW	3rd HW (halfword)	
buflen	reclen	filler	invariant-input

buflen:	Length of the buffer
reclen:	Length of the record written by SDF
filler:	Filler
invariant-input:	INVARIANT-INPUT form of the statement, starting at the seventh byte

SPIN =
Specifies which statement, in batch mode, SDF is to read and analyze next.

NO
SDF is to read and process the next statement in the statementsequence.

YES
SDF is to skip all statements until the next STEP statement (or, as the case may be, until the END statement) and, if there is a STEP statement, continue processing with the statement following it.

ERRSTMT =
defines which statement terminates the spin-off mechanism if SDF senses a syntax error for the read statement.

STEP
SDF initiates spin-off until STEP or END is recognized. The return code is X'1C', X'34',...

NEXT
SDF does not initiate spin-off. The next statement is read on the next RDSTMT call. The return code is then X'50'.

CALLID =
This function applies to the OPNCALL and CLSCALL macros.
CALLID specifies the program context (=syntax file hierarchy opened by an OPNCALL macro) in which the statement must be read and analyzed. The name of the syntax file hierarchy (CALLID) must have the 4-byte value returned by SDF to the field which was designated by the CALLID operand in the OPNCALL macro.

***NO**
The current syntax file hierarchy (context) of the task is used for analyzing the statement. This can, for example, be the syntax file hierarchy opened for the task at LOGON.

addr / (r7)
Address of the call check field or register containing this address. The area must be aligned on a word boundary.

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the “XHCS” manual [11]). This parameter has no effect on message output.

***NO**
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

name
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long.

MF =
Defines special requirements for macro expansion (see the “Executive Macros” manual [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This either appears in register 1 or is specified directly.

Return information and error flags

The format of the transfer area is described in section "Format of the standardized transfer area up to SDF V4.0" on page 593ff.

Register 15 contains return code in the right-aligned byte and specifications regarding the assignment of SYSSTMT in the leftmost byte. EQUATE statements for all these can be generated with the aid of the CMDANALY macro.

X'00'	Normal termination
X'04'	Unrecoverable system error
X'08'	Operand error in the macro
X'0C'	Transfer area too small
X'10'	Error in end-of-file (EOF) or in statement, end-of-file (EOF) was detected
X'14'	Error in statement, a command was then detected
X'18'	Statement is correct but the default values provided by the program are errored
X'1C'	Error in statement, STEP was detected
x'28'	Buffer too small, logging aborted
X'2C'	END statement has been read
X'34'	Error in statement, the next statement to be processed is //END
X'38'	SDF not available
X'3C'	Program not known in syntax file
X'44'	Syntax file not found
X'4C'	The program is not executable above the 16-Mbyte boundary, since SDF is not loaded. Please notify system administration.
X'50'	Error in statement, spin-off not initiated
X'5C'	Not enough space in INVAR buffer, INVARIANT-INPUT truncated
X'64'	XHCS error

Assignment of SYSSTMT:

X'01'	SYSSTMT = terminal
X'02'	SYSSTMT = file
X'03'	SYSSTMT = card reader
X'04'	SYSSTMT = floppy disk
X'05'	SYSSTMT = SYSCMD in S procedure
X'06'	SYSSTMT = S variable

9.1.5 TRSTMT
Analyze statement

The TRSTMT macro causes SDF to

– analyze a program statement stored in the program itself, and

– pass the results of the analysis to the program.

Additional statements may result from the analysis of the transferred statement, due to the fact that a system exit may replace the transferred statement by several statements. Dealing with these additional statements is the responsibility of the program.

An activated syntax file must contain the definition of the program and its statements.

Operation	Operands
TRSTMT	PROGRAM = name
	,INPUT = *NO / addr / (r1)
	,OUTPUT = addr / (r2)
	[,STMT = *ALL / (name,...) / *ADDR(addr/(r))]
	[,DIALOG =NO / YES / ERROR]
	[,MESSAGE = *NO / addr / (r3)]
	[,PROT = *NO / *YES / addr / (r4)]
	[,INVAR = *NO / addr / (r)]
	[,DEFAULT = *NO / (addr,...)]
	[,ERROR = NO / YES]
	[,CALLID = *NO / addr / (r7)]
	[,EXECUTE = NO / YES]
	[,PROCMOD = ANY / NO / YES]
	[,CCSNAME = *NO / *EXTEND / name]
	[,INPUTSAV = *NO / YES]
	[,MF = $\left\{\begin{array}{l}\text{L}\\ \text{(E,(1))}\\ \text{(E,opadr)}\end{array}\right\}$]

PROGRAM = name
Internal name of the program that is executed by the macro. This name is stored in the program definition in the syntax file (see ADD-PROGRAM). It is at least one byte and at most eight bytes long.

INPUT =
specifies which statement SDF is to analyze.

***NO**
SDF is not to analyze any statement stored in the program, but is instead to analyze an additional statement provided by a system exit.

addr / (r1)
SDF is to analyze the statement whose address is specified or whose address is contained in the specified register. SDF expects the statement in the form of a variable-length record in the usual BS2000 format. The record area must be aligned on a half-word boundary.

OUTPUT = addr / (r2)
Address of the standardized transfer area, or a register that contains this address. The area must begin on a word boundary. Prior to calling the RDSTMT macro the program must ensure that the first two bytes of this area contain the maximum length possible for the area (see section “Format of the standardized transfer area up to SDF V4.0” on page 593ff). In this area SDF stores the analysis results.

STMT =
Specifies which statements are permitted as input.

<u>*ALL</u>
All statements are permitted.

(name,...)
Only the statements with the internal names specified are permitted. The internal statement name is stored in the statement definition in the syntax file (see ADD-STMT). It is at least one byte and at most eight bytes long.
The standard SDF statements are always permitted, regardless of the specification made here.

***ADDR(addr/(r))**
Address of the list of permissible statements. This list must be generated beforehand with the CMDALLW macro.

DIALOG =
Specifies whether SDF is to conduct a dialog when analyzing statements. This operand is relevant only when the program is executing in an interactive task.

NO
SDF is not to conduct a dialog.

YES
SDF is to present the statement given to it by the program to the user in dialog for possible modification, provided this is compatible with the current SDF specifications for the dialog (see MODIFY-SDF-OPTIONS and SET-GLOBALS).

ERROR
SDF is to conduct a dialog only when it has detected a syntax error. If the statement contains a semantic error, the program can initiate a semantic error dialog by means of CORSTMT.

MESSAGE =
Specifies whether SDF is to issue a message when presenting the statement to the user for checking and possible modification (relevant only for DIALOG ≠ NO). SDF integrates this message into the form.

***NO**
SDF is not to issue a message.

addr / (r3)
Address of the message text to be issued, or a register that contains this address. The text is expected in the form of a variable-length record with a maximum length of 400 characters. However, only the first 80 characters are displayed on SDF-formatted screens. If the text contains screen control characters, the menu mask may be destroyed. The record area must be aligned on a halfword boundary.

PROT =
specifies whether SDF is to log input and messages to SYSOUT. If they are not written to SYSOUT, the user of the program should be informed of this in the program documentation. In contrast to the RDSTMT macro, this parameter has been implemented as an integer (byte field). A logging buffer is available.

***NO**
SDF is not to perform any logging.

YES
SDF is to log input and messages to SYSOUT.

addr / (r4)
Address of a buffer, or a register that contains this address. SDF is to write input and messages to be logged into this buffer. The buffer must begin on a halfword boundary. The length of the buffer is contained in bytes 0 and 1, the length of the record to be logged is contained in bytes 2 and 3.
If the buffer is not empty, the first record is generally the log of the input command. Subsequent records contain messages. If no input log is available, or if the input log cannot be output, two slashes (“//”) are written into the output area.
The PROT parameter has the following effect:

Result of analysis	**PROT parameter**	
	YES (or addr / reg)	**NO**
No error	Input statement	– / –
Syntax error	1. Input statement 2. Syntax error message	– / –

INVAR =
Specifies whether the INVARIANT-INPUT form of the statement is stored. This means that the statement is stored with all the user-defined operands, all operands having default values and all operands currently allowed for this task. INVARIANT-INPUT is thus the largest input form for a statement available to a user who has certain privileges and who is working in the selected dialog mode. In contrast to LOGGING=INVARIANT-FORM (see MODIFY-SDF-OPTIONS), this form *does not* mask out keywords and secret operands.

INVAR = <u>*NO</u>
The INVARIANT-INPUT form of the statement is not stored.

INVAR = addr / (reg)
Specifies the address of a buffer into which SDF writes the INVARIANT-INPUT form of the statement. The buffer must be aligned on a word boundary and the first halfword (HW) must contain the length of the buffer. SDF stores the INVARIANT-INPUT form as a record of variable length beginning with the second halfword. he contents of the buffer are then as follows:

1st HW	2nd HW	3rd HW (halfword)	
buflen	reclen	filler	invariant-input

buflen:	Length of the buffer
reclen:	Length of the record written by SDF
filler:	Filler
invariant-input:	INVARIANT-INPUT form of the statement,starting at the seventh byte

DEFAULT =
specifies whether the following values are to be replaced by SDF with values dynamically generated by the program:
– operand values entered or
– operand default values

The operands, or operand values, must have been defined accordingly in the syntax file (see ADD OPERAND..., OVERWRITE-POSSIBLE=*YES,... and ADD-VALUE...,VALUE=<c-string> (OVERWRITE-POSSIBLE=*YES),...). The program-generated value must be a valid operand value. In guided dialog, the values generated by the program are displayed by SDF in the form.

Example:

In the MODIFY statements issued to SDF-A, the value *UNCHANGED is replaced by the current value.
If the operands to be given default values are in a structure introduced by a value defined with LIST-ALLOWED=*YES (see ADD-VALUE), the following situation may arise:
The conversion description contains several list elements to which a structure with operands to be defaulted is attached. At the same time, the user likewise enters several list elements to which a structure with operands to be defaulted is attached.
SDF first tries to match the structures entered by the user to those specified in the conversion description by means of the value introducing the structure. If an unambiguous allocation cannot be made on the basis of the value introducing the structure because none of the values entered matches any of the ones in the conversion description or because the user has entered the matching value more than once, the allocation is then made based on the list position of the introductory value.

<u>*NO</u>
SDF is not to replace the operand values entered by values generated dynamically by the program.

(addr,...)
In one or more of the possible statements, SDF is to replace entered operand values by values generated dynamically by the program. The conversion descriptions for these statements (see section “Format of the standardized transfer area up to SDF V4.0” on page 593ff) are located at the specified addresses (addr51,...) in the program. A conversion description contains, among other things, the internal statement name and information regarding which of the operand values entered are to be changed and what values they are to be changed to. The areas for the conversion descriptions which are passed for the default values of the program must be aligned on a word boundary. The same is true of the output area of the macros (OUTPUT operand).

ERROR =
Specifies how the message text specified with the MESSAGE operand is to be issued.

NO
SDF is to issue the message text as a message.

YES
SDF is to issue the message text as an error message.

CALLID =
Defines the context to be used by SDF for analyzing the command.

***NO**
The currently active syntax file hierarchy is used.

addr / (r7)
Address of a 4-byte field or register containing this address. The caller transfers the callid of the context to be used. This field must be aligned on a word boundary.

EXECUTE = NO/YES
Specifies whether standard SDF statements are to be executed.
EXECUTE is irrelevant if the TRSTMT macro does not refer to a new syntax file hierarchy (CALLID=*NO), e.g. if the current syntax file hierarchy is used. In this case, the standard SDF statements are always executed by TRSTMT, provided they exist in the current hierarchy. The operand belongs to the multihierarchical attribute introduced with the preceding CALLID operand. If TRSTMT refers to a hierarchy opened in parallel (CALLID=addr7/r(7)), the standard SDF statements are executed if EXECUTE=YES.

PROCMOD =
Defines the environment in which the user works. SDF performs a check, as a result of which it rejects commands which are illegal in the specified environment and are not accepted by SDF. The operand refers to the option of opening several syntax hierarchies. It is only important if the macro call refers to a new syntax file hierarchy opened in addition to the current one. If no CALLID has been defined (CALLID=*NO), PROCMOD is irrelevant. For instance, the value ANY is set automatically and reference is made to the current procedure mode.

ANY
No check is made. Statements, however, are always analyzed.

YES
Statements are handled as if they were read from a procedure file. For instance, they are analyzed if they have been defined in the syntax file with DIALOG-PROC-ALLOWED=YES and if the program runs in interactive mode, or if BATCH-PROC-ALLOWED=YES in batch jobs.

NO
Statements are handled as if they were read from a primary level, e.g. from terminal input or from a batch job. They are analyzed if they have been defined in the syntax file with DIALOG-ALLOWED=*YES and if the program runs in interactive mode, or if BATCH-ALLOWED=*YES in batch jobs.

CCSNAME =
Specifies the name of the character set used for the correction dialog on 8-bit terminals and for conversion from lowercase to uppercase letters. Each terminal uses a certain character set. A coded character set (CCS) is the unique representation of the characters in a character set in binary form. Each coded character set is defined by its coded character set name, or CCSN (see the “XHCS” manual [11]). This parameter has no effect on message output.

***NO**
Standard 7-bit code is used for I/O operations.

***EXTEND**
Standard 8-bit code is used for I/O operations.

name
Specifies the name of a special 8-bit code for I/O operations. The name must be 8 bytes long.

INPUTSAV =
Specifies whether a history of past inputs is to be saved in the form of a list that can be subsequently accessed again via the built-in RESTORE mechanism (see the MODIFY-SDF-OPTIONS and RESTORE-SDF-INPUT statements).

***NO**
The inputs are not saved.

***YES**
Inputs are saved in a buffer.

MF =
Defines special requirements for macro expansion (see the “Executive Macros” manual [8] for details).

L
Only the data part of the macro expansion (operand list) is generated. This requires that no operand types with executable code appear in the macro. The data part generated has the address specified in the name field of the macro.

(E,(1)) / (E,opaddr)
Only the instruction part of the macro expansion is generated. The associated data part (operand list) is referenced by the address “opaddr”. This either appears in register 1 or is specified directly.

Return information and error flags

The format of the transfer area is described in section “Format of the standardized transfer area up to SDF V4.0” on page 593ff.

Register 15 contains a return code and a flag for the existence of additional statements in the right-aligned and left-aligned bytes, respectively. EQUATE statements for all these can be generated with the aid of the CMDANALY macro.

X’00’	Normal termination
X’04’	Unrecoverable system error
X’08’	Operand error in the macro
X’0C’	Transfer area too small
X’18’	The statement is correct but the default values provided by the system are errored
X’1C’	Error in statement
X’20’	Error dialog not possible
X’24’	Error dialog rejected
X’28’	Error or logging area too small
X’2C’	END statement has been read
X’38’	SDF not available
X’3C’	Program not known in syntax file
X’44’	Syntax file not found
X’48’	SDF command/statement executed
X’4C’	The program is not executable above the 16-Mbyte boundary, since SDF is not loaded. Please notify system administration.
X’5C’	Not enough space in INVAR buffer, INVARIANT-INPUT truncated

X’64’	XHCS error

Indicators of additional statements (created by a system exit):

X’00’	There are no further statements
X’01’	There are further statements

9.2 Mutually exclusive data types

This table shows which combinations of data types are valid for the various operands.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1: alphan.-name	()	x	x	x			x	x	x	x	x	x			x			x	x		x		x	x	
2: cat-id	x	n	x	x					x		x	x			x			x	x		x		x	x	
3: command-rest	x	x	n	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
4: comp.-name	x	x	x	()		x	x	x	x	x	x	x			x		x	x	x		x		x	x	
5: c-string			x		()												x		x					x	x
6: date			x	x		n			x										x					x	
7: device	x		x	x			x		x		x	x			x			x	x		x		x	x	
8: fixed	x		x	x				()		x		x					x		x	x	x		x	x	
9: filename	x	x	x	x		x	x		()		x	x			x		x	x	x		x		x	x	
10: integer	x		x	x				x		()		x							x	x	x		x	x	
11: keyword	x	x	x	x			x		x						x			x	x		x		x	x	
12: keyw-number	x	x	x	x			x	x	x	x					x			x	x		x		x	x	
13: *keyword			x																x					x	
14: *keyw-numb.			x																x					x	
15: name	x	x	x	x			x		x		x	x			()			x	x		x		x	x	
16: partial-filen.			x													()			x					x	
17: prod.-version			x	x	x			x	x								()		x		x			x	x
18: struct.-name	x	x	x	x			x		x		x	x			x			()	x		x		x	x	
19: text	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	()	x	x	x	x	x	x
20: time			x					x		x									x	n	x			x	
21: vsn	x	x	x	x			x	x	x	x	x	x			x		x	x	x	x	()		x	x	
22: x-string			x																x			()		x	
23: x-text	x	x	x	x			x	x	x	x	x	x			x			x	x		x		()	x	
24: posix-filen. / posix-pathn.	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	()	x
25: 'posix-filen.' / 'posix-pathn.'			x		x												x		x					x	()

No entry	Combination of data types permitted without restriction.
x	Combination of data types is permitted only if VALUE-OVERLAPPING=*YES.
( )	Combination of the same data types of differing lengths or with differing value range is permitted only if VALUE-OVERLAPPING=*YES.
n	Multiple specification of the same data type is not permitted as no additional attributes have been specified to permit differentiation.

Glossary

data type

Basic syntax type, which can be checked for syntax errors by the SDF command processor. The possible data types are shown in the tables starting on page 11.

domain

A set of commands, compiled according to user criteria, from which the user can select the command desired in guided dialog. Domains may overlap.

list

Assignment of several operand values to one operand in a command or statement. The list is formed by enclosing the values, separated by commas, within parentheses. Whether a given operand permits a list is an attribute of that operand. List parentheses are distinguishable from parentheses enclosing a structure by means of the context.

structure

Syntactical combination of several operands. This combination is expressed syntactically by enclosing the operands in parentheses. Structure parentheses are distinguishable from parentheses enclosing a list by means of the context. A structure can either be an operand value itself or it can be dependent on the specification of an input alternative that introduces the structure.

Related publications

[1] **SDF V4.5A** (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

Target group
BS2000/OSD users
Contents
This manual describes the interactive input of commands and statements in SDF format. A Getting Started chapter with easy-to-understand examples and further comprehensive examples facilitates use of SDF. SDF syntax files are discussed.
Order number
U2339-J-Z125-8-76

[2] **SDF V4.5A** (BS2000/OSD)
SDF Management
User Guide

Target group
This manual is intended for system administrators and experienced BS2000 users.
Contents
It describes how SDF is installed and administered using SDF commands and the SDF-I, SDF-U and SDF-PAR utility routines. It includes a description of SDF-I, SDF-U and SDF-PAR statements.
Order number
U2622-J-Z125-10-76

[3] **SDF-A** (BS2000/OSD)
Ready Reference

Target group
This publication is intended for experienced BS2000 users with a good knowledge of SDF-A.
Contents
It contains the SDF-A statements in alphabetical order as well as the macros and function calls of the SDF program interface, the formats of the transfer area and the SDF-SIM statements.
Order number
U2285-J-Z125-9-76

[4] **BS2000/OSD-BC V5.0**
Commands, Volumes 1 - 5
User Guide

Target group
This manual is addressed to nonprivileged users and systems support staff.
Contents
Volumes 1 through 5 contain the BS2000/OSD commands ADD-... to WRITE-... (basic configuration and selected products) with the functionality for all privileges. The command and operand functions are described in detail, supported by examples to aid understanding. An introductory overview provides information on all the commands described in Volumes 1 through 5.
The Appendix of Volume 1 includes information on command input, conditional job variable expressions, system files, job switches, and device and volume types.
The Appendix of Volumes 4 and 5 contains an overview of the output columns of the SHOW commands of the component NDM. The Appendix of Volume 5 contains additionally an overview of all START commands.
There is a comprehensive index covering all entries for Volumes 1 through 5.
Order numbers

U2338-J-Z125-15-76	Commands, Volume 1, A – C
U41074-J-Z125-2-76	Commands, Volume 2, D – MOD-JO
U21070-J-Z125-5-76	Commands, Volume 3, MOD-JV – R
U41075-J-Z125-2-76	Commands, Volume 4, S – SH-PRI
U23164-J-Z125-4-76	Commands, Volume 5, SH-PUB – Z

[5] **BS2000/OSD-BC V5.0**
Commands, Volume 6, Output in S Variables and SDF-P-BASYS
User Guide

Target group
This manual is addressed to programmers and users who write procedures.
Contents
Volume 6 contains tables of all S variables that are supplied with values by the SHOW commands in conjunction with structured output. Further chapters deal with:
– introduction to working with S variables
– SDF-P-BASYS V2.2A
Order number
U23165-J-Z125-4-76

[6] **BS2000/OSD-BC V5.0**
Introductory Guide to Systems Support
User Guide

Target group
This manual is addressed to BS2000/OSD systems support staff and operators.
Contents
The manual covers the following topics relating to the management and monitoring of the BS2000/OSD basic configuration: system initialization, parameter service, job and task control, memory/device/system time/user/file/pubset management, assignment of privileges, accounting and operator functions.
Order number
U2417-J-Z125-14-76

[7] **BS2000/OSD-BC V5.0**
Introductory Guide to DMS
User Guide

Target group
This manual is addressed to nonprivileged users and systems support staff.
Contents
It describes file management and processing in BS2000.
Attention is focused on the following topics:
– volumes and files
– file and catalog management
– file and data protection
– OPEN, CLOSE and EOV processing
– DMS access methods (SAM, ISAM,...)
The main new features of OSD-BC V5.0 are the introduction of files larger than or equal to 32 Gbytes and the possibility of restricting TSOS co-ownership of files.
Order number
U4237-J-Z125-7-76

[8] **BS2000/OSD-BC V5.0**
Executive Macros
User Guide

Target group
This manual is addressed to all BS2000/OSD assembly language programmers.
Contents
The manual contains a summary of all Executive macros:
– linking and loading
– virtual storage, memory pool, ESA
– task and program control
– ITC, serialization, eventing, DLM, contingencies, STXIT
– messages, accounting, JMS, TIAM, VTSU, ...
Detailed description of all macros in alphabetical order and with examples; general training section dealing with ITC, serialization, eventing, DLM, contingencies, STXIT, virtual storage, memory pool, ESA, ...
Order number
U3291-J-Z125-10-76

[9] **BS2000/OSD-BC V5.0**
System Installation
User Guide

Target group
This manual is intended for BS2000/OSD system administration.
Contents
The manual describes the generation of the hardware configuration with UGEN and the following installation services: disk organization with MPVS, the installation of volumes using the SIR utility routine, and the IOCFCOPY subsystem.
Order number
U2505-J-Z125-15-76

[10] **SECOS V4.0A** (BS2000/OSD
Security Control System
User Guide

Target group
– BS2000 system administrators
– BS2000 users working with extended access protection for files
Contents
Capabilities and application of the functional units:
– SRPM (System Resources and Privileges Management)
– SRPMSSO (Single Sign On)
– GUARDS (Generally Usable Access Control Administration System)
– GUARDDEF (Default Protection)
– GUARDCOO (Co-owner Protection)
– SAT (Security Audit Trail).
Order number
U5605-J-Z125-6-76

[11] **XHCS V1.3** (BS2000/OSD)
8-Bit Code Processing in BS2000/OSD
User Guide

Target group
Users of the DCAM, TIAM and UTM access methods, system administrators, and users migrating from EHCS to XHCS.
Contents
XHCS (Extended Host Code Support) is a software package of BS2000/OSD that lets you use extended character sets in conjunction with 8-bit terminals. XHCS is also the central source of information on the coded character sets in BS2000/OSD. XHCS replaces EHCS.
Order number
U9232-J-Z135-4-76

[12] **SDF-P V2.2A** (BS2000/OSD)
Programming in the Command Language
User Guide

Target group
This manual is addressed to BS2000 users and systems support staff.
Contents
SDF-P is a structured procedure language in BS2000. The manual begins with introductory chapters dealing with the basic principles of procedures and variables, and goes on to provide detailed descriptions of SDF-P commands, functions and macros.
Overview of contents:
– brief introduction to SDF-P
– procedure concept in SDF-P
– creating, testing, calling and controlling S procedures
– S variables, S variable streams, functions, expressions
– converting non-S procedures
– macros, predefined (built-in) functions, SDF-P commands

SDF-P V2.2A can only be used in conjunction with SDF-P-BASYS ≥ V2.1A, VAS ≥ V2.0A and SDF ≥ V4.1A.
Order number
U6442-J-Z125-5-76

[13] **IMON V2.5** (BS2000/OSD)
Installation Monitor
User Guide

Target group
This manual is intended for systems support staff of the BS2000/OSD operating system.
Contents
The manual describes the installation and administration of BS2000 software using the IMON installation monitor and its three components IMON-BAS, IMON-GPN and IMON-SIC. Installation (standard and customer-specific) using the component IMON-BAS for systems with BS2000-OSD V2.0 and as of BS2000-OSD V3.0/V4.0 is described in detail with the aid of examples in two separate chapters.
Order number
U21926-J-Z125-3-76

[14] **DSSM V4.0/SSCM V2.3**
Subsystem Management in BS2000/OSD
User Guide

Target group
This manual addresses systems support staff and software consultants of BS2000/OSD.
Contents
The following are described: BS2000/OSD subsystem concept, dynamic subsystem management (DSSM) V4.0, subsystem catalog management (SSCM) V2.3 and the associated commands and statements.
DSSM supports the option of creating and managing user-specific subsystem configurations on a task-local basis.
Order number
U23166-J-Z125-3-76

[15] **BS2000/OSD**
Softbooks English
CD-ROM

Target group
BS2000/OSD users
Contents
The CD-ROM "BS2000/OSD SoftBooks English" contains almost all of the English manuals and README files for the BS2000 system software of the latest BS2000/OSD version and also of the previous versions, including the manuals listed here.
These Softbooks can also be found in the Internet on our manual server. You can browse in any of these manuals or download the entire manual.
Order number
U26175-J8-Z125-1-76
Internet address
http://manuals.fujitsu-siemens.com

Index

Contents

SDF-A V4.1E (BS2000/OSD)

User Guide

Target group
This manual is intended for experienced BS2000 users and system administration staff.

Contents
It describes how to process syntax files and explains the SDF-A functions on the basis of examples. The SDF-A statements are listed in alphabetical order.
The manual also includes a description of the SDF-SIM utility routine.

Edition: March 2002

File: sdf_a.pdf

This manual was produced by
cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Fujitsu Siemens computers GmbH
User Documentation
81730 Munich
Germany

Fax: (++49) 700 / 372 00000

e-mail: manuals@fujitsu-siemens.com
http://manuals.fujitsu-siemens.com

Comments
Suggestions
Corrections

Submitted by

Comments on SDF-A V4.1E