English

Fujitsu Software BS2000

ESQL-COBOL

ESQL-COBOL for SESAM/SQL-Server

User Guide

Valid for:
ESQL-COBOL V3.0A

Edition November 2006

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to: bs2000.info@fujitsu.com

Certified documentation according to DIN EN ISO 9001:2015

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2015.

Copyright and Trademarks

Contents

1 Preface

ESQL-COBOL is an important add-on product for the SESAM®/SQL-Server database system. Through its functions and its architectural features, the SESAM/SQL-Server database system fulfills all today's requirements of a powerful database server. This is reflected in the product name, SESAM/SQL-Server.

For the sake of simplicity, this manual refers to SESAM/SQL when describing the SESAM/SQL-Server database system.

This chapter includes

- Brief product description for ESQL-COBOL
- Target group
- Summary of contents
- Changes since the last version of the manual
- Notational conventions

1.1 Brief product description

ESQL-COBOL allows you to embed SQL in the COBOL host language.

Using SQL statements embedded in COBOL, you can:

– define, query and update SESAM/SQL databases

– perform database administration tasks for SESAM/SQL databases

SQL

SQL (Structured Query Language) is a relational database language which has been standardized by ISO (International Organization for Standardization) in ISO/IEC 9075:2003. The standardization of SQL means that you can create portable SQL applications.

ESQL-COBOL fully supports the mandatory features of the SQL standard. In addition, it supports some optional features from the SQL standard. Moreover, ESQL-COBOL includes powerful extensions to the SQL standard. The language subset supported is described in "SQL Reference Manual Part 1: SQL Statements" [2] and "SQL Reference Manual Part 2: Utilities" [3].

1.2 Target group

This manual is aimed at COBOL programmers who wish to use SQL statements to define or access SESAM/SQL databases.

The manual assumes a basic knowledge of BS2000, a knowledge of SQL, and experience in COBOL programming. If applications are to be generated under openUTM, a knowledge of UTM is also necessary.

1.3 Summary of contents

This manual covers the following topics:

– The embedding of SQL in a COBOL program
– Precompilation of the ESQL-COBOL program with the ESQL precompiler
– Compilation of a COBOL program
– Linking of an ESQL-COBOL application
– Starting an ESQL-COBOL application
– ESQL-COBOL applications under UTM
– Example programs and messages from the ESQL-COBOL system

The examples provided are based mainly on the sample database presented in the appendix.

The manual contains a glossary which defines the technical terms used.
As well as the table of contents, you can use the running headers and the index to find the information you require. References to other publications are included in the text in abbreviated form. The complete title of the publication is included in the list of related publications at the back of the manual.

1.4 README file

Please refer to the product-specific README file for details on functional modifications and supplements to this manual regarding the current version of the product. The README file is located on your BS2000 computer under the name SYSRME.*product.version.language*. Please consult systems support for the user ID under which the README file is located. You can view the README file with the command /SHOW-FILE or view it in an editor or print it to a standard printer using the following command:

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

or, if SPOOL with a version earlier than 3.0A is used:

```
/PRINT-FILE FILE-NAME=filename,LAYOUT-CONTROL=
    PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
```

1.5 Changes since the last version of the manual

This section describes the changes made between this edition of the manual and the last version:

- Support of character representation in UTF-16:
  - Data type NCHAR
  - Data type NVARCHAR

1.6 Notational conventions

This manual uses two sets of notational conventions:

– conventions for statements which are not in SDF format

– SDF syntax for commands in SDF format

1.6.1 Notational conventions for statements

The statements are represented using a specific metasyntax. The meanings of the individual syntax elements are summarized in the table below:

Syntax element	Meaning	Example
italics	Italics denote a variable to which a current value has to be assigned.	*level-no data-name* Example of input: 01 COMPANY
Fixed-space font	The fixed-space font is used in examples.	EXEC SQL Example of input: EXEC SQL
{ }	Alternatives are enclosed in braces. Each line contains one alternative expression. Only one expression may be selected. The braces are metacharacters and must not be entered.	{ X A 9 } Example of input: 9
[ ]	Expressions in square brackets may be omitted. The brackets are metacharacters and must not be entered.	[INDICATOR]
... ,... { },...	Repetition characters: You can specify the preceding expression repeatedly; each such expression must be separated from the next by a blank or a comma. The expression enclosed in braces may be specified repeatedly; each such expression must be separated from the next by a comma.	*expression,...* Example of input: ADDRESS,SALARY

Table 1: Notational conventions for COBOL statements

1.6.2 SDF syntax

Figure 1 shows an example of how command syntax is represented in a manual. The command format comprises a field containing the command name, which is followed by a list of all the operands and their permitted settings. Operand values that introduce structures and the operands that are dependent on them are also listed.

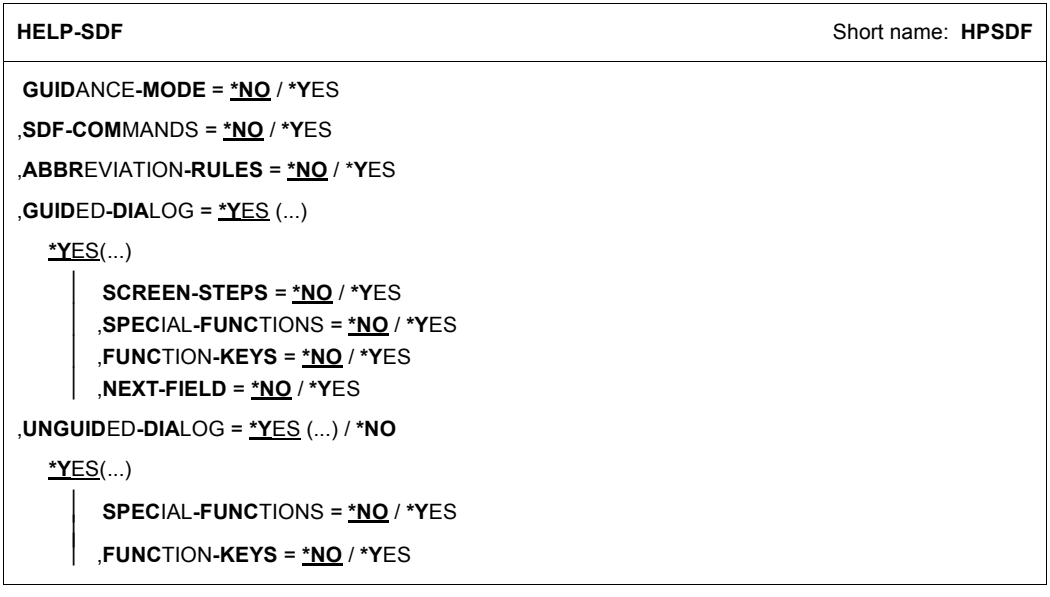


Figure 1: Syntax diagram for the user command HELP-SDF

This syntax description is based on SDF Version 4.0A. The syntax of the SDF command/statement language is explained in the three tables that follow.

Table 2: Metasyntax

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

Table 3: 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 this table.

The description of the data types is valid for the entire set of commands/statements. For this reason, only deviations (if any) are explained in table 4 in the operand descriptions.

Table 4: Suffixes for data types

Data type suffixes define additional rules for data type input. They can be used to extend or limit the set of values. The following short forms are used in this manual for data type suffixes:

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

The entry for the ‘integer’ data type in table 4 also contains a number of items set in italics; the italics are not part of the syntax and are only used to make the table easier to read.

The description of the data types suffixes is valid for the entire set of commands/statements. For this reason, only deviations (if any) are explained in table 4 in the relevant operand descriptions.

Metasyntax

Representation	Meaning	Examples
UPPERCASE LETTERS	Uppercase letters denote keywords. The keywords for constant operand values begin with *.	**HELP-SDF** **SCREEN-STEPS** = ***NO**
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** = ***NO**
< >	Angle brackets denote variables whose range of values is described by data types and suffixes (see Tables 3 and 4).	**SYNTAX-F**ILE = <full-filename 1..54>
Underscoring	Underscoring denotes the default value of an operand.	**GUID**ANCE-**MODE** = ***NO**
/	A slash serves to separate alternative operand values.	**NEXT-FIELD** = ***NO** / ***Y**ES
(…)	Parentheses denote operand values that initiate a structure.	,**UNGUID**ED-**DIA**LOG = ***Y**ES (...) / ***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 = ***Y**ES (...) ***Y**ES(...) **SCREEN-STEPS** = ***NO** / ***Y**ES

Table 2: 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**(...) / <full-filename 1..54>
Abbreviation:	The name that follows represents a guaranteed alias for the command or statement name.	**HELP-SDF** Abbreviation: **HPSDF**

Table 2: 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 full-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 interactive prompting, 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 3: Data types (part 1 of 6)

Data type	Character set	Special rules
full-filename	A…Z 0…9 $, #, @ hyphen period	Input format: [:cat:][$user.] { file | file(no) | group | group { (*abs) | (+rel) | (-rel) } } :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: $name_1$[.$name_2$[...]] $name_i$ does not contain a period and must not begin or end with a hyphen; file can have a max. length of 41 characters; it must not begin with a $ and must include at least one character from the range A...Z.

Table 3: Data types (part 2 of 6)

Data type	Character set	Special rules
full-filename (continued)		#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”) group { (*abs) / (+rel) / (-rel) } (*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 3: 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 full-filename $user. see full-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 full-filename). The final character of “partname” must be a period. At least one of the parts :cat:, $user. or partname must be specified.
posix-filename	A...Z 0...9 special characters	String with a length of up to 255 characters; consists of either one or two periods or of alpha-numeric characters and special characters. The special characters must be escaped with a preceding \ (backslash); the slash (/) is not allowed. Must be enclosed within single quotes if alter-native data types are permitted, separators are used, or the first character is a ? or ! A distinction is made between uppercase and lowercase.
posix-pathname	A...Z 0...9 special characters structure identifier: slash	Input format: [/]$part_1$/.../$part_n$ where $part_i$ is a posix-filename; max. 1024 characters; must be enclosed within single quotes if alter-native data types are permitted, separators are used, or the first character is a ? or !

Table 3: 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][n]n.nann['] correction status release status where n is a digit and a is a letter. The release and correction status must be specified if product-version does not include a suffix (see suffix without-corr and without-man in Table 4). 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 b) 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) Max. 6 characters; PUB may be prefixed, but must not be followed by $, # or @.

Table 3: 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 3: Data types (part 6 of 6)

Suffixes for data types

Suffix	Meaning
x..y *unit*	a) 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* b) with the other data types: length specification 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.
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.
-under	Permits underscores “_” for the data type “name”.

Table 4: Data type suffixes (part 1 of 6)

<table>
<tr><th>Suffix</th><th>Meaning</th></tr>
<tr><td>with (contd.)</td><td></td></tr>
<tr><td>-wild(n)</td><td>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, wildcards from the UNIX world (referred to below as POSIX wildcards) are now accepted in addition to the usual BS2000 wildcards. However, 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.
<table>
<tr><th>BS2000 wildcards</th><th>Meaning</th></tr>
<tr><td>*</td><td>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.</td></tr>
<tr><td>Termina-ting period</td><td>Partially-qualified entry of a name.
Corresponds implicitly to the string “.*”, i.e. at least one other character follows the period.</td></tr>
<tr><td>/</td><td>Replaces any single character.</td></tr>
<tr><td><s_x:s_y></td><td>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 les between s_x and s_y in the alphabetic collating sequence; numbers are sorted after letters (A...Z0...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')</td></tr>
<tr><td><s_1,…></td><td>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).</td></tr>
</table>
</td></tr>
</table>

Table 4: Data type suffixes (part 2 of 6)

Suffix	Meaning	
with-wild(n)		
(continued)	-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 full-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 the 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.
	$[c_x\text{-}c_y]$	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 $[c_x\text{-}c_y]$ and [s] can be combined into $[s_1c_1\text{-}c_2s_2]$
	$[!c_x\text{-}c_y]$	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 $[!c_x\text{-}c_y]$ and [!s] can be combined into $[!s_1c_1\text{-}c_2s_2]$
	[!s]	Replaces exactly one character not contained in string s. The expressions [!s] and $[!c_x\text{-}c_y]$ can be combined into $[!s_1c_1\text{-}c_2s_2]$

Table 4: Data type suffixes (part 3 of 6)

<table>
<tr><th>Suffix</th><th>Meaning</th></tr>
<tr><td>with (contd.)

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

Table 4: Data type suffixes (part 4 of 6)

Suffix	Meaning
with-constr (continued)	The following rules must be observed when specifying a constructor: – The constructor must include at least one wildcard of the selector. – If the number of identical wildcards exceeds those in the selector, the index notation must be used. – 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 the wildcard “*” is to be duplicated. For example: “<n><n>” must be specified instead of “**”. – 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 selector ends with a period, the constructor must also end with a period (and vice versa).

Table 4: Data type suffixes (part 5 of 6)

<table>
<tr><th>Suffix</th><th>Meaning</th></tr>
<tr><td>with-constr
(contd.)</td><td>Examples:
<table>
<tr><th>Selector</th><th>Selection</th><th>Constructor</th><th>New name</th></tr>
<tr><td>A//*</td><td>AB1
AB2
A.B.C</td><td>D<3><2></td><td>D1
D2
D.CB</td></tr>
<tr><td>C.<A:C>/<D,F></td><td>C.AAD
C.ABD
C.BAF
C.BBF</td><td>G.<1>.<3>.XY<2></td><td>G.A.D.XYA
G.A.D.XYB
G.B.F.XYA
G.B.F.XYB</td></tr>
<tr><td>C.<A:C>/<D,F></td><td>C.AAD
C.ABD
C.BAF
C.BBF</td><td>G.<1>.<2>.XY<2></td><td>G.A.A.XYA
G.A.B.XYB
G.B.A.XYA
G.B.B.XYB</td></tr>
<tr><td>A//B</td><td>ACDB
ACEB
AC.B
A.CB</td><td>G/XY/</td><td>GCXYD
GCXYE
GCXY. 1)
G.XYC</td></tr>
</table>
1) The period at the end of the name may violate naming conventions (e.g. for fully-qualified file names).</td></tr>
<tr><td>without</td><td>Restricts the specification options for a data type.</td></tr>
<tr><td>-cat</td><td>Specification of a catalog ID is not permitted.</td></tr>
<tr><td>-corr</td><td>Input format: [[C]'][V][n]n.na[']
Specifications for the data type product-version must not include the correction status.</td></tr>
<tr><td>-gen</td><td>Specification of a file generation or file generation group is not permitted.</td></tr>
<tr><td>-man</td><td>Input format: [[C]'][V][n]n.n[']
Specifications for the data type product-version must not include either release or correction status.</td></tr>
<tr><td>-odd</td><td>The data type x-text permits only an even number of characters.</td></tr>
<tr><td>-sep</td><td>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).</td></tr>
<tr><td>-user</td><td>Specification of a user ID is not permitted.</td></tr>
<tr><td>-vers</td><td>Specification of the version (see “file(no)”) is not permitted for tape files.</td></tr>
</table>

Table 4: Data type suffixes (part 6 of 6)

2 Embedding SQL in COBOL programs

This chapter tells you how to create an ESQL-COBOL application. It covers the following topics:

- Creating an ESQL-COBOL application
- Elements in an ESQL-COBOL program
- Host variables
- Assigning SQL and COBOL data types
- SQL statements in an ESQL-COBOL program
- SQL comments in an ESQL-COBOL program
- Communication area
- INCLUDE elements

2.1 Creating an ESQL-COBOL application

The figure below shows how to create an executable ESQL-COBOL application from an ESQL-COBOL program.

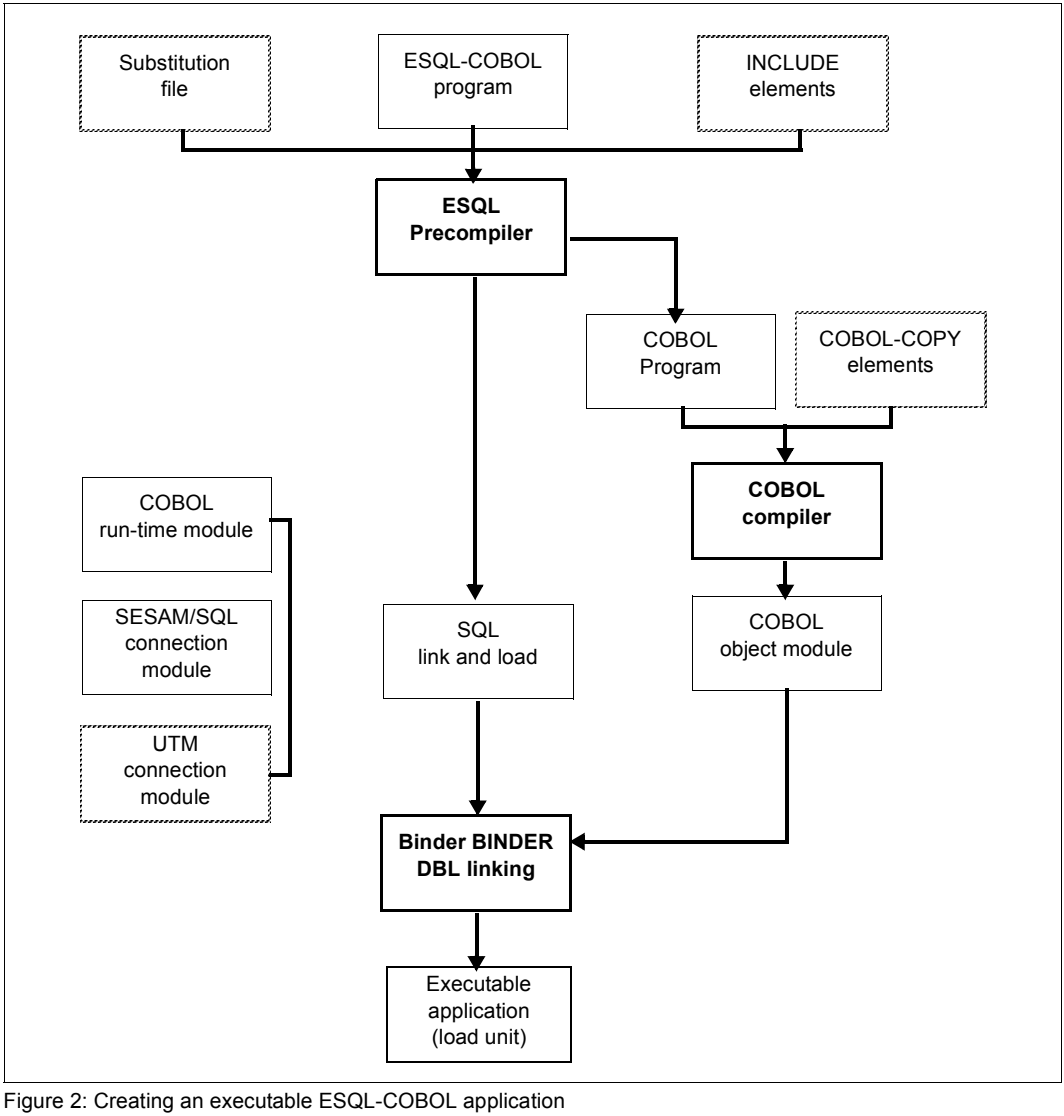


Figure 2: Creating an executable ESQL-COBOL application

Precompilation

The ESQL-COBOL program is precompiled with the ESQL precompiler. If the ESQL-COBOL program contains the SQL statement INCLUDE, the relevant INCLUDE elements are linked into it. Substitution files can be used to convert or port ESQL-COBOL applications. This facilitates the substitution of names, names in double quotes, and keywords. If the ESQL-COBOL program is error-free, the ESQL precompiler generates an SQL link and load module (LLM) and a COBOL program that does not contain any SQL statements.

Compilation

The COBOL program generated is compiled with the COBOL2000 compiler. If the COBOL program is error-free, the COBOL compiler generates a COBOL link module.

Linking

The generated modules (the COBOL runtime module, the SESAM/SQL connection module, and, possibly, UTM connection modules) are linked with the linker BINDER and the linking loader DBL to create an executable ESQL-COBOL application in the form of a load unit.

2.2 Elements in an ESQL-COBOL program

In addition to COBOL language constructs, an ESQL-COBOL program consists of the following:

- the definition of host variables
- SQL statements and comments
- a communication area

The sections that follow describe how the individual elements are embedded in a COBOL program and outline the rules that you must follow.

2.3 Host variables

A host variable is a COBOL data field that you can use in an embedded SQL statement. You can use a host variable to transfer values from the database to the ESQL-COBOL program and process them there. You can also use a host variable to transfer data in the opposite direction, to the database, to supply values needed for computations. Furthermore, you can use a host variable as an indicator variable. An indicator variable is a special host variable which is used to monitor the transfer of data and the NULL value. The SESAM/SQL manual “SQL Reference Manual Part 1: SQL Statements” [2] describes the SQL statements in which host variables can be used.

2.3.1 Defining host variables

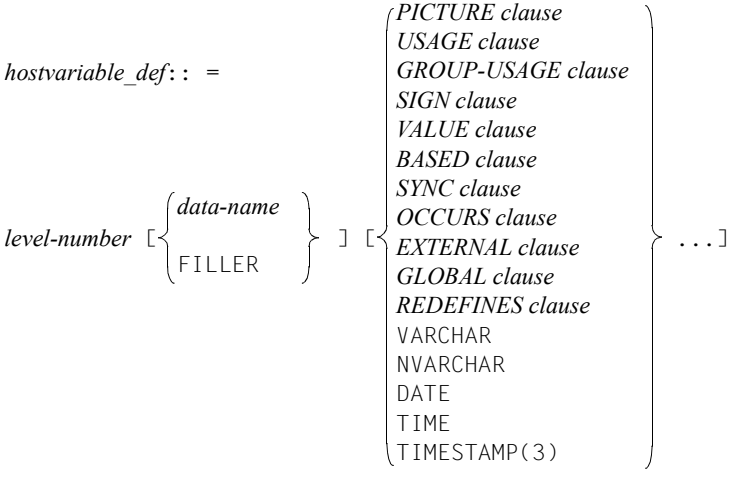
A host variable has to be defined in the source code of the ESQL-COBOL program before it is used in an SQL statement. With nested programs, the host variable definition has to be visible as defined in COBOL rules each time the host variable is used (see “Naming host variables” on page 37). When a host variable is used in a DECLARE CURSOR statement, the definition has to be visible during each OPEN CURSOR statement for the cursor in question.

You define host variables in a DECLARE SECTION in the DATA DIVISION. You can define as many DECLARE SECTIONs as you wish. However, DECLARE SECTIONs must not be nested. The definition of host variables is governed by COBOL conventions relating to the definition of data fields, and by SESAM/SQL-specific rules. In addition, data types are supported which correspond to the SQL data types VARCHAR, NVARCHAR, DATE, TIME(3) and TIMESTAMP(3). These data types and the SESAM/SQL-specific rules are described in section “Assigning SQL and COBOL data types” on page 45. A detailed description of the definition of COBOL data fields is provided in the “COBOL2000 (BS2000/OSD) Reference Manual” [16].

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC

      hostvariable_def ...

EXEC SQL END DECLARE SECTION END-EXEC
```



If column 7 of a line in the DECLARE SECTION contains a character other than a blank, the ESQL precompiler interprets the line as a comment. This means that lines of COBOL cannot be continued in the subsequent line with a hyphen within a DECLARE SECTION. It is therefore recommended that you mark comment lines with an asterisk (*) in column 7. Debugging lines marked with a D are permitted and are likewise interpreted as comment lines by the ESQL precompiler.

The clauses can be in any order. Each clause may only be specified once.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC

EXEC SQL END DECLARE SECTION END-EXEC
```

The beginning and end of a DECLARE SECTION. Each must be written out in full on a separate line between column 8 and column 72.

level-number

The level number. Level numbers are governed by COBOL conventions. The level numbers 66 is not permitted. If you have selected the ISO language subset with the precompile option SOURCE-PROPERTIES (see section “Specifying the properties of the ESQL-COBOL program” on page 101), only the level numbers 01 and 77 are permitted.

data-name

A data name compliant with COBOL rules. The permitted characters include uppercase and lowercase letters, digits and hyphens. No distinction is made between uppercase and lowercase. *data-name* must not be more than 30 characters long and must not be a reserved COBOL word. VARCHAR, TIMESTAMP, EXEC and END-EXEC, too, may not be used as *data-name*. *data-name* must not begin or end with a hyphen, and must contain at least one letter. If you have chosen the ISO language subset with the precompiler option SOURCE-PROPERTIES (see section “Specifying the properties of the ESQL-COBOL program” on page 101), you must specify a data name. The names of variables with the level number 01 or 77 must be unique in the source code. Lower-ranking data names must be unique within the data group (see “Naming host variables” on page 37). To avoid names that conflict with data names from the communication area, you should therefore not use data names that begin with the letters SQL within a DECLARE SECTION, except for SQLCODE and SQLSTATE.

PICTURE clause

PICTURE clause. The PICTURE clause for an alphanumeric data field must contain at least an `X`. The mask characters `A` and `9` are supported for alphanumeric fields in order to maintain compatibility with earlier SESAM/SQL versions. However, you should only use the mask character `X`. In the PICTURE clause for a numeric data field, the first mask character you must enter is `S`. The PICTURE clause cannot be used with floating-point numbers (USAGE IS COMP-1 or USAGE IS COMP-2) (see section “Single-precision floating-point number” on page 59 and section “Double-precision floating-point number” on page 60).

The PICTURE clause for a national data item may only contain the PICTURE symbol N.

USAGE clause

USAGE clause. If you do not specify a USAGE clause, DISPLAY is chosen automatically. However, if the data item is national, defined by the PICTURE symbol N in the PICTURE character-string, USAGE NATIONAL is assumed. The USAGE clause may only be used in connection with data fields that have no lower-ranking data fields.

GROUP-USAGE clause

GROUP-USAGE clause. In COBOL, GROUP-USAGE NATIONAL defines that a group with only national subitems is treated as a national elementary item and not as an alphanumeric item as it would be otherwise.

SIGN clause

SIGN clause for numeric data fields. The SIGN clause is only permitted if no USAGE clause or USAGE IS DISPLAY is specified.

VALUE clause
VALUE clause. In the VALUE clause, literals are specified in accordance with the COBOL rules. National literals are introduced by N' or NX' in accordance with the COBOL definition. When a national literal is specified in the VALUE clause, the optional PICTURE phrase can be omitted and is derived from the literal.

Decimals can be specified using a point or a comma. An alphanumeric value specified with the VALUE clause can be enclosed in single quotes (') instead of double quotes ("), depending on how the ESQL precompiler and the COBOL compiler have been configured.

SYNCHRONIZED clause
SYNCHRONIZED clause.

OCCURS clause
OCCURS clause. The OCCURS clause may only be used to specify the exact number of repetitions of a data field (Format 1, see the „COBOL2000 (BS2000/OSD) Reference Manual" [16]). The OCCURS clause must not be nested.

ASCENDING, DESCENDING, KEY and DEPENDING are ignored by the precompiler.

EXTERNAL clause
EXTERNAL clause.

GLOBAL clause
GLOBAL clause.

`VARCHAR`
Definition of a host variable for a variable-length string. See section "Variable-length character string" on page 47 for information on this data type.

`NVARCHAR`
Definition of a host variable for a variable-length national string. See section "Variable-length national character string" on page 50 for information on this data type.

`DATE`
Definition of a host variable for a date. See section "Date" on page 61 for information on this data type.

`TIME(3)`
Definition of a host variable for a point in time. See section "Time" on page 63 for information on this data type.

`TIMESTAMP(3)`
Definition of a host variable for a timestamp. See section "Timestamp" on page 65 for information on this data type.

Example

Defining host variables:

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
     EXEC SQL BEGIN DECLARE SECTION END-EXEC
*
*** HOST VARIABLES FOR ORDERS TABLE
*
 01 ORDERS.
    02 ORDER-NUM     PIC S9(9) USAGE IS BINARY.
    02 ORDER-DATE DATE.
       03 YEAR       PIC S9(4) USAGE IS BINARY.
       03 MONTH      PIC S9(4) USAGE IS BINARY.
       03 DAY        PIC S9(4) USAGE IS BINARY.
    02 ORDER-TEXT    PIC X(30).
    02 ORDER-STATUS  PIC S9(9) USAGE IS BINARY.

     EXEC SQL END DECLARE SECTION END-EXEC
     .
     .
     .
  further COBOL data definitions
     .
     .
     .
```

Limitations of the ISO language subset

If you have used the precompiler option SOURCE-PROPERTIES to select the ISO language subset (see section “Specifying the properties of the ESQL-COBOL program” on page 101), the following clauses must not be used:

- BASED
- SYNCHRONIZED
- OCCURS
- EXTERNAL
- GROUP-USAGE
- GLOBAL
- VARCHAR
- NVARCHAR
- DATE
- TIME(3)
- TIMESTAMP(3)

These clauses are subject to a number of limitations, which are described in the sections on the individual data types (see section “Assigning SQL and COBOL data types” on page 45).

Naming host variables

The ESQL precompiler does not recognize the structure of nested ESQL-COBOL programs. This can lead to undesired referencing if a nested ESQL-COBOL program contains a host variable and a COBOL variable that have identical names. You can avoid unwanted referencing of this type by naming variables systematically.

Example

Unwanted referencing in a nested ESQL-COBOL program:

```
 PROGRAM-ID. A1.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*****************************************************************
* HOST VARIABLE X
*****************************************************************
      EXEC SQL BEGIN DECLARE SECTION END-EXEC.
  01 X PIC X(10).
      EXEC SQL END DECLARE SECTION END-EXEC.
 .
 .
 .
 PROGRAM-ID. A2.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*****************************************************************
* COBOL VARIABLE X
*****************************************************************
  01 X PIC X(11).

  PROCEDURE DIVISION.
  01.
     EXEC SQL
       SELECT A INTO :X  -- COBOL VARIABLE X FROM A2 IS REFERENCED
                         -- WHEN IT SHOULD NOT BE
              FROM T
        END EXEC
   END-PROGRAM A2.
   END-PROGRAM A1.
```

2.3.2 Specifying host variables in SQL statements

When you specify a host variable in an SQL statement, the variable must be preceded by a colon. The name of the host variable must not be followed immediately by a letter, digit or hyphen.

Example

Defining host variables for the CUSTOMERS table and using them in an SQL statement:

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.

    EXEC SQL BEGIN DECLARE SECTION END-EXEC
*
* DEFINITION OF HOST VARIABLES FOR THE CUSTOMERS TABLE
*
  01  CUSTOMERS.
      05  CUST-NUM    PIC   S9(9).
      05  COMPANY     PIC   X(40).
      05  STREET      PIC   X(40).
    EXEC SQL END DECLARE SECTION END-EXEC
 .
 .
 .

 PROCEDURE DIVISION.

*
* USING HOST VARIABLES IN AN SQL STATEMENT
*

  EXEC SQL
    INSERT INTO CUSTOMERS (CUST_NUM, COMPANY, STREET)
    VALUES (:CUST-NUM,:COMPANY,:STREET)
  END EXEC
```

2.3.2.1 Qualifying the names of lower-ranking data fields

In an SQL statement, a lower-ranking data field must be specified in such a way that it can be identified clearly and unambiguously. To do this you can qualify the name of the lower-ranking data field with the name of a higher-ranking data field by preceding the name of the lower-ranking data field with the name of the higher-ranking data field and separating the two with a period. Data fields are qualified in the opposite order to the order used in COBOL.

Example

Defining and using host variables as a data group:

```
DATA DIVISION.
WORKING-STORAGE SECTION.

*
* DEFINITION OF HOST VARIABLES
*
    EXEC SQL BEGIN DECLARE SECTION END-EXEC
 01 ADDRESS.
    02 STREET   PIC X(40).
    02 ZIP      PIC S9(6).
    02 CITY     PIC X(40).
    EXEC SQL END DECLARE SECTION END-EXEC

   .
   .
   .

PROCEDURE DIVISION.

*
* ADDRESSING HOST VARIABLES IN AN SQL STATEMENT
*
    EXEC SQL
       SELECT STREET FROM CUSTOMERS INTO :ADDRESS.STREET
    END-EXEC

   .
   .
   .
*
* ADDRESSING HOST VARIABLES IN A COBOL STATEMENT
*
    MOVE "Edlingerstrasse" TO STREET OF ADDRESS.
```

2.3.2.2 Addressing vectors

If a host variable is defined as a vector, it is possible to address a single element of the vector or a range of the vector. To address a single element of a vector, you specify the element's index. To address a range of a vector, you specify the indexes for the start and the end of the range. The counting of the indexes of vectors begins with 1 in SQL and COBOL.

Example

Defining and using the vector for color components in the COLOR_TAB table:

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
    EXEC SQL BEGIN DECLARE SECTION END-EXEC
*
*** DEFINITION OF VECTOR
*
 01 COLOR.
    02 COLOR-COMP  PIC SV99 OCCURS 3 TIMES.
*
    EXEC SQL END DECLARE SECTION END-EXEC
   .
   .
   .

  PROCEDURE DIVISION.

   .
   .
   .
*
*** USING A VECTOR IN AN SQL STATEMENT
*
    EXEC SQL
      SELECT RGB(1..3) INTO :COLOR-COMP(1..3)
          FROM COLOR_TAB
          WHERE COLOR_NAME = 'skyblue'

    END-EXEC
*
 .
 .
 .
```

```
*
*** ADDRESSING THE RANGE OF THE VECTOR IN AN SQL STATEMENT
*
    EXEC SQL
      UPDATE COLOR_TAB
          SET RGB(1..2) = :COLOR-COMP(1..2)
          WHERE COLOR_NAME = 'skyblue'
    END-EXEC.
*
*** ADDRESSING AN INDIVIDUAL ELEMENT OF THE VECTOR
*
    EXEC SQL
      UPDATE COLOR_TAB
          SET RGB(1) = :COLOR-COMP(1)
          WHERE COLOR_NAME = 'flame'
END-EXEC
```

2.3.3 Indicator variables

A host variable can be combined with an indicator variable in an SQL statement. An indicator variable is a host variable that you use to verify the transfer of values from the database to an associated host variable and to transfer the NULL value.

2.3.3.1 Defining indicator variables

You define indicator variables in the same way as other host variables within a DECLARE SECTION in the DATA DIVISION. The data type of the indicator variable must be assigned to the SQL data type SMALLINT (see section “Small integer” on page 52).

If the host variable is a vector, the associated indicator variable must also be defined as a vector with the same number of elements.

If the data type of the host variable is assigned to the SQL data type DATE, TIME(3), TIMESTAMP(3) or VARCHAR, the associated indicator variable has to be defined as a simple data field.

Example 1

Defining host variables and their associated indicator variables for the ORDERS table:

```
      EXEC SQL BEGIN DECLARE SECTION END-EXEC
*
**** DEFINITION HOST VARIABLES
*
 01 COMPANY PIC X(40).
*
 01 ORDERS.
    02 ORDER-NUM     PIC S9(9) USAGE IS BINARY.
    02 ORDER-DATE DATE.
        03 YEAR    PIC S9(4) USAGE IS BINARY.
        03 MONTH   PIC S9(4) USAGE IS BINARY.
        03 DAY     PIC S9(4) USAGE IS BINARY.
    02 ORDER-TEXT    PIC X(30).
    02 ORDER-STATUS  PIC S9(9) USAGE IS BINARY.
 *
 *** DEFINITION OF ASSOCIATED INDICATOR VARIABLES
 *
  01  IND-COMPANY  PIC S9(4) USAGE IS BINARY.
 *
  01 IND-ORDERS.
     02 IND-ORDER-NUM       PIC S9(4) USAGE IS BIANRY.
     02 IND-ORDER-DATE      PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-TEXT      PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-STATUS    PIC S9(4) USAGE IS BINARY.

      EXEC SQL END DECLARE SECTION END-EXEC
```

Example 2

Defining a vector and the relevant indicator variable for the COLOR_TAB table:

```
     EXEC SQL BEGIN DECLARE SECTION END-EXEC
*
*** HOST VARIABLES FOR COLOR_TAB
*
 01 COLOR-TAB.
    02 COLOR-NAME  PIC X(15).
    02 RGB         PIC SV99 OCCURS 3 TIMES.
*
*** INDICATOR VARIABLES FOR COLOR_TAB
*
 01 IND-COLOR-TAB.
    02 IND-COLOR-NAME   PIC S9(4) COMP.
    02 IND-RGB          PIC S9(4) COMP OCCURS 3 TIMES.
*
     EXEC SQL END DECLARE SECTION END-EXEC
```

2.3.3.2 Specifying indicator variables in an SQL statement

A host variable may only be combined with an indicator variable if the host variable is used as follows:

– to query data in the database

– to enter values in the database

– to update values in the database

– in computations (functions, expression, predicates, search conditions)

You must mark indicator variables with a colon in SQL statements, just as you would any other host variable. You specify the indicator variable after the host variable to which the indicator variable is to apply. The host variable and the indicator variable can be separated by a blank. You can also mark the indicator variable with the keyword INDICATOR.

Example

Specifying indicator variables for a simple host variable and a vector in an SQL statement:

```
EXEC SQL
     FETCH CUR_COLOR_TAB
           INTO :COLOR-NAME  INDICATOR :IND-COLOR-NAME,
                :RGB(1..3)   INDICATOR :IND-RGB(1..3)
END-EXEC
```

2.3.3.3 Verifying the transfer of values

When a value in the database is queried and assigned to a host variable, the associated indicator variable is set by SESAM/SQL as follows:

Value	Meaning
0	The host variable contains the value that is read. The assignment was error-free.
-1	The value to be assigned is the NULL value. The assignment did not take place.
>0	With alphanumeric values: The host variable was assigned a truncated string. The value of the indicator variable indicates the string's original length.

Table 5: Values stored in indicator variables and their meanings

2.3.3.4 Transferring the NULL value

The NULL value is impossible to represent in a host variable. For this reason, indicator variables are needed to transfer NULL values.

When the SQL statement is executed, the host variable's value is not used; the NULL value is used instead.

Example

The indicator variable IND1 is set to the value -2 in order to transfer the NULL value to the database with the host variable CUST-NUM:

```
IF CUST-NUM = '000000000'
   THEN
   MOVE -2 TO IND1
   ELSE
   MOVE  0 TO IND1.

EXEC SQL
   UPDATE ORDERS
   SET CUST_NUM = :CUST-NUM INDICATOR :IND1
END-EXEC
```

2.4 Assigning SQL and COBOL data types

When defining a host variable you have to choose a COBOL data type whose assigned SQL data type suits the relevant column.

SQL data types are described in the “SQL Reference Manual Part 1: SQL Statements” [2]; COBOL data types are described in the “COBOL2000 (BS2000/OSD) Reference Manual” [16].

In the sections that follow, the associated COBOL data type is specified with each SQL data type.

The ISO language subset

If you have selected the ISO language subset with the precompiler option SOURCE-PROPERTIES (see section “Specifying the properties of the ESQL-COBOL program” on page 101), certain limitations apply to individual clauses, and you must abide by the sequence specified for the clauses. The restrictions and the sequence are described for the individual data types under the heading **ISO language subset**.

Repetition of mask characters in the PICTURE clause

A mask character can be repeated in a PICTURE clause by including it more than once, or by specifying a repetition factor in brackets. You can also combine the two methods.

Example

The following are all equivalent expressions:

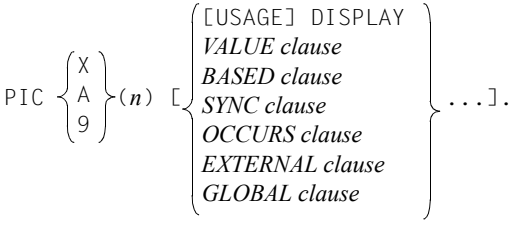
```
PIC XXX
PIC X(3)
PIC X(2)X
```

The representation of PICTURE, USAGE and SIGN clauses

For clarity’s sake, IS is not specified below in the PICTURE, USAGE and SIGN clauses. However, you can include IS if you wish. In addition, the PICTURE clause is abbreviated to PIC, but PICTURE can be written out in full.

2.4.1 Fixed-length character string

The following COBOL data type is assigned to the SQL data type **CHARACTER**(*n*):



ISO language subset

```
PIC X(n) [VALUE clause].
```

n Integer between 1 and 256 which specifies the length of the string. With host variables that are only used in the SQL statement PREPARE or EXECUTE IMMEDIATE, the string may be up to 32,000 characters long. The mask character `X` must be specified at least once. The mask characters `A` and `9` are supported in order to maintain compatibility with earlier versions of SESAM/SQL. You should only use the mask character `X`.

Example

The SQL data type and associated COBOL data type for a fixed-length character string:

SQL data type

```
COMPANY CHARACTER(40)
```

COBOL data type

```
01 COMPANY-NAME PIC X(40).
```

2.4.2 Variable-length character string

The following COBOL data type is assigned to the SQL data type **VARCHAR**(*m*) or CHAR[ACTER] VARYING(*max*):

---

st_nr1 [*dataname1*] VARCHAR [BASED].

st_nr2 dataname2 PIC S9(*n*) [USAGE] { COMP[UTATIONAL] / COMP[UTATIONAL]-5 / BINARY } [{ *VALUE clause* / *SYNC clause* } ...].

st_nr2 dataname3 PIC { X / A / 9 }(*m*) [{ [USAGE] DISPLAY / *VALUE clause* / *SYNC clause* } ...].

ISO language subset

Host variables with the data type VARCHAR are not permitted.

st_nr1 dataname1 VARCHAR.
: Declares a data group with the SQL stat type VARCHAR. The data name of the lower-ranking data fields *dataname2* and *dataname3* must be different.

st_nr2 dataname2 PIC...
: Lower-ranking data field of the data type “small integer”, where $1 \leq n \leq 4$ (see section “Small integer” on page 52). The contents of this data field specify the length of the string.

st_nr2 dataname3 PIC...
: Lower-ranking data field with the data type “fixed-length” string, where $1 \leq m \leq 32000$ (see section “Fixed-length character string” on page 46). This data field contains the string. m is the maximum length of string of the SQL data type VARCHAR(*m*).

Example

The SQL data type and associated COBOL data type for a variable-length character string:

SQL data type

```
TITLE VARCHAR(500)
```

COBOL data type

```
01 TITLE VARCHAR
   02 LENGTH PIC S9(4) USAGE IS BINARY.
   02 STRING PIC X(500).
```

2.4.3 Fixed-length national character string

The following COBOL data type is assigned to the SQL data type **NCHAR**(*n*):

```
PIC N(n) [ { [USAGE] DISPLAY
           { VALUE clause
           { BASED clause
           { SYNC clause
           { OCCURS clause
           { EXTERNAL clause
           { GLOBAL clause      } ...].
```

ISO language subset

```
PIC N(n) [VALUE clause].
```

n Integer between 1 and 128 which specifies the length of the string (in national character positions). The mask character `N` must be specified at least once.

Example

The SQL data type and associated COBOL data type for a fixed-length national character string:

SQL data type

```
COMPANY NCHAR(40)
```

COBOL data type

```
01 COMPANY-NAME PIC N(40).
```

2.4.4 Variable-length national character string

The following COBOL data type is assigned to the SQL data type **NVARCHAR**(*m*):

---

st_nr1 [*dataname1*] NVARCHAR [BASED].

st_nr2 dataname2 PIC S9(*n*) [USAGE] { COMP[UTATIONAL] | COMP[UTATIONAL]-5 | BINARY } [{ *VALUE clause* | *SYNC clause* } ...].

st_nr2 dataname3 PIC N(*m*) [{ [USAGE] DISPLAY | *VALUE clause* | *SYNC clause* } ...].

ISO language subset

Host variables with the data type NVARCHAR are not permitted.

st_nr1 dataname1 NVARCHAR.
: Declares a data group with the SQL stat type NVARCHAR. The data name of the lower-ranking data fields *dataname2* and *dataname3* must be different.

st_nr2 dataname2 PIC...
: Lower-ranking data field of the data type “small integer”, where $1 \leq n \leq 4$ (see section “Small integer” on page 52). The contents of this data field specify the length of the string.

st_nr2 dataname3 PIC...
: Lower-ranking data field with the data type “fixed-length national string”, where $1 \leq m \leq 16{,}000$ (see section “Fixed-length character string” on page 46). This data field contains the string. m is the maximum length of string of the SQL data type NVARCHAR(m).

Example

The SQL data type and associated COBOL data type for a variable-length national character string:

SQL data type

```
TITLE NVARCHAR(500)
```

COBOL data type

```
01 TITLE NVARCHAR
   02 LENGTH PIC S9(4) USAGE IS BINARY.
   02 STRING PIC N(500).
```

2.4.5 Small integer

The following COBOL data type is assigned to the SQL data type **SMALLINT**:

```
PIC S9(n) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause      }...].
                  { COMP[UTATIONAL]-5 }   { BASED clause      }
                  { BINARY            }   { SYNC clause       }
                                          { OCCURS clause     }
                                          { GLOABAL clause    }
                                          { EXTERNAL clause   }
```

ISO language subset

```
PIC S9(n) [USAGE] { COMP[UTATIONAL]   } [VALUE clause].
                  { COMP[UTATIONAL]-5 }
                  { BINARY            }
```

n Integer between 1 and 4 which indicates the number of places.

The SQL data type SMALLINT comprises integers in the range from -32768 to +32767. The COBOL data type `PIC S9(4)` comprises integers in the range from -9999 to +9999, which is narrower. When USAGE IS BINARY is specified, the value in COBOL statements is truncated to four decimal places. A COBOL statement containing such a variable can cause an overflow if the value of the variable is outside the range -9999 to +9999.

Examples

The SQL data type and associated COBOL data type for a small integer for values expected to be in the range from -9999 to +9999:

SQL data type:

```
PAGE SMALLINT
```

COBOL data type:

```
01 PAGE PIC S9(4) USAGE IS BINARY.
```

The SQL data type and associated COBOL data type for a small integer for a value expected to be in the range from -32768 to 32767:

SQL data type:

```
PAGE SMALLINT
```

COBOL data type:

```
01 PAGE PIC S9(4) USAGE IS COMP.
```

2.4.6 Integer

The following COBOL data type is assigned to the SQL data type **INTEGER**:

```
PIC S9(n) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause      }...].
                  { COMP[UTATIONAL]-5 }   { BASED clause      }
                  { BINARY            }   { SYNC clause       }
                                          { OCCURS clause     }
                                          { GLOBAL clause     }
                                          { EXTERNAL clause   }
```

ISO language subset

```
PIC S9(n) [USAGE] { COMP[UTATIONAL]   } [VALUE-clause].
                  { COMP[UTATIONAL]-5 }
                  { BINARY            }
```

n Integer between 5 and 9 which specifies the number of places.

The data type INTEGER comprises integers in the range from -2147483648 to +2147483647. The COBOL data type `PIC S9(9) COMP` comprises integers in the range from -999999999 to +999999999, which is narrower. When USAGE IS BINARY is specified, the value in COBOL statements is truncated to nine decimal places. A COBOL statement containing such a variable can cause an overflow if the value of the variable is outside the range -9999999999 to +999999999.

Examples

The SQL data type and associated COBOL data type for an integer with expected values in the range -999999999 to +999999999:

SQL data type:

```
CUST_NUM INTEGER
```

COBOL data type:

```
01 CUSTOMER-NUM PIC S9(9) USAGE IS BINARY.
```

The SQL data type and associated COBOL data type for an integer with expected values in the range -2147483648 to 2147483647:

SQL data type:

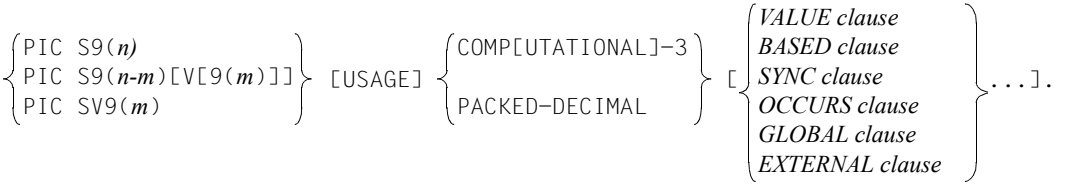
```
CUST_NUM INTEGER
```

COBOL data type:

```
01 CUSTOMER-NUM PIC S9(9) USAGE IS COMP.
```

2.4.7 Fixed-point number (packed)

The following COBOL data type is assigned to the SQL data type **DECIMAL**(*n*) or DECIMAL(*n,m*):



ISO language subset

Host variables of the data type DECIMAL are not permitted.

n Integer between 1 and 31 which specifies the total number of decimal places.

m Integer between 1 and *n* which specifies the number of decimal places.

Examples

The SQL data type and associated COBOL data type for a fixed-point number without decimal places (packed):

SQL data type:

```
YEAR DECIMAL(2)
```

COBOL data type:

```
01 YEAR PIC S9(2) USAGE IS PACKED-DECIMAL.
```

The SQL data type and associated COBOL data type for a fixed-point number with decimal places (packed):

SQL data type:

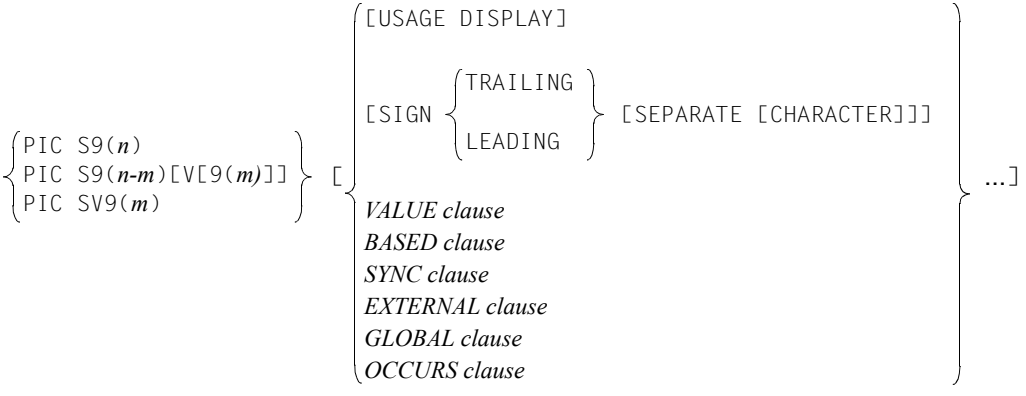
```
SUBTOTAL DECIMAL(6,4)
```

COBOL data type:

```
01 SUBTOTAL PIC S9(2)V9(4) USAGE IS PACKED-DECIMAL.
```

2.4.8 Fixed-point number (unpacked)

The following COBOL data type is assigned to the SQL data type **NUMERIC**(*n*) or NUMERIC(*n,m*):



ISO language subset

n Integer between 1 and 31 which specifies the total number of decimal places.

m Integer between 1 and *n* which specifies the number of decimal places.

Examples

The SQL data type and associated COBOL data type for a fixed-point number without decimal places (unpacked):

SQL data type:

```
NUMBER NUMERIC(8,0)
```

COBOL data type:

```
01 NUMBER PIC S9(8)
```

The SQL data type and associated COBOL data type for a fixed-point number with decimal places (unpacked):

SQL data type:

```
PRICE NUMERIC(8,2)
```

COBOL data type:

```
01 PRICE PIC S9(6)V9(2)
```

2.4.9 Single-precision floating-point number

The following COBOL data type is assigned to the SQL data type **REAL**:

```
[USAGE COMP[UTATIONAL]-1
[VALUE clause]
[BASED clause]
[SYNC clause]
[EXTERNAL clause]
[GLOBAL clause]
[OCCURS clause]
```

ISO language subset

Host variables of the data type REAL are not permitted.

Example

The SQL data type and assigned COBOL data type for a single-precision floating-point number:

SQL data type:

```
REAL_NUM REAL
```

COBOL data type:

```
01 REAL-NUM USAGE COMP-1.
```

2.4.10 Double-precision floating-point number

The following COBOL data type is assigned to the SQL data type **DOUBLE PRECISION**:

```
[USAGE] COMP[UTATIONAL]-2
[VALUE clause]
[BASED clause]
[SYNC clause]
[EXTERNAL clause]
[GLOBAL clause]
[OCCURS clause]
```

ISO language subset

Host variables of the data type DOUBLE PRECISION are not permitted.

Example

The SQL data type and assigned COBOL data type for a double-precision floating-point number:

SQL data type:

```
NUM DOUBLE PRECISION
```

COBOL data type:

```
01 NUM USAGE COMP-2.
```

2.4.11 Floating-point number

No COBOL data type is assigned to the SQL data type **FLOAT**(*n*).

However, the range of values covered by the SQL data type FLOAT(*n*) is covered by the SQL data type DOUBLE PRECISION, or if $n \leq 21$, by the SQL data type REAL (see section “Double-precision floating-point number” on page 60 and section “Single-precision floating-point number” on page 59). Host variables of the data type DOUBLE PRECISION or REAL can therefore be used to read attributes of the data type FLOAT(*n*).

2.4.12 Date

The following COBOL data type is assigned to the SQL data type **DATE**:

```
st_nr1 [d_name1] DATE [BASED].
    st_nr2 d_name2 PIC S9(y) [USAGE] { COMP[UTATIONAL]   } [{ VALUE clause }...].
                                     { COMP[UTATIONAL]-5 }  { SYNC clause  }
                                     { BINARY            }
    st_nr2 d_name3 PIC S9(m) [USAGE] { COMP[UTATIONAL]   } [{ VALUE clause }...].
                                     { COMP[UTATIONAL]-5 }  { SYNC clause  }
                                     { BINARY            }
    st_nr2 d_name4 PIC S9(d) [USAGE] { COMP[UTATIONAL]   } [{ VALUE clause }...].
                                     { COMP[UTATIONAL]-5 }  { SYNC clause  }
                                     { BINARY            }
```

ISO language subset

Host variables of the data type DATE are not permitted.

st_nr1 d_name1 DATE
: Defines a data group with the SQL data type DATE. The data group contains three lower-ranking data fields with the data type “small integer”.

st_nr2 d_name2 ... st_nr2 d_name4 **...**
: Lower-ranking data fields with the data type “small integer”. The first data field is the year, the second is the month, and the third is the day. *y,m,d* are integers between 1 and 4. The data fields must have different names.

Example

SQL data type with assigned COBOL data type for a date:

SQL data type:

```
DATE_INF DATE
```

COBOL data type:

```
01 DATE-INF DATE.

   02 YEAR PIC S9(4) BINARY.
   02 MONTH PIC S9(2) BINARY.
   02 DAY PIC S9(2) BINARY.
```

2.4.13 Time

The following COBOL data type is assigned to the SQL data type **TIME**(3):

```
st_nr1 [d_name1] TIME(3) [BASED].
    st_nr2 d_name2 PIC S9(h) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause } ...].
                                     { COMP[UTATIONAL]-5 }   { SYNC clause  }
                                     { BINARY            }
    st_nr2 d_name3 PIC S9(m) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause } ...].
                                     { COMP[UTATIONAL]-5 }   { SYNC clause  }
                                     { BINARY            }
    st_nr2 d_name4 PIC S9(s) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause } ...].
                                     { COMP[UTATIONAL]-5 }   { SYNC clause  }
                                     { BINARY            }
    st_nr2 d_name5 PIC S9(t) [USAGE] { COMP[UTATIONAL]   } [ { VALUE clause } ...].
                                     { COMP[UTATIONAL]-5 }   { SYNC clause  }
                                     { BINARY            }
```

ISO language subset: Host variables of the data type TIME(3) are not permitted.

st_nr1 d_name1 TIME(3)
: Defines a data group with the SQL data type TIME(3). The data group contains four lower-ranking data fields with the data type “small integer”.

st_nr2 d_name2 ... st_nr2 d_name5
: Lower-ranking data fields with the data type “small integer” (see section “Small integer” on page 52). The first data field contains the hours, the second contains the minutes, the third contains the seconds, and the fourth contains the milliseconds. The data fields must have different names.

Example

SQL data type and assigned COBOL data type for a time:

SQL data type:

```
TIME_INF TIME(3)
```

COBOL data type:

```
01 TIME-INF TIME(3).

   02 HOURS     PIC S9(2) BINARY.
   02 MINUTES   PIC S9(2) BINARY.
   02 SECONDS   PIC S9(2) BINARY.
   02 MILLISECS PIC S9(3) BINARY.
```

2.4.14 Timestamp

The following COBOL data type is assigned to the SQL data type **TIMESTAMP**(3):

```
st_nr1 [d_name1] TIMESTAMP(3).

    st_nr2 d_name2 PIC S9(y) [VALUE clause].
    st_nr2 d_name3 PIC S9(m) [VALUE clause].
    st_nr2 d_name4 PIC S9(d) [VALUE clause].
    st_nr2 d_name5 PIC S9(h) [VALUE clause].
    st_nr2 d_name6 PIC S9(m) [VALUE clause].
    st_nr2 d_name7 PIC S9(s) [VALUE clause].
    st_nr2 d_name8 PIC S9(t) [VALUE clause].
```

ISO language subset

Host variables with the data type TIMESTAMP(3) are not permitted.

st_nr1 d_name1 TIMESTAMP(3).
: Defines a data group with the SQL data type TIMESTAMP(3). The data group contains seven lower-ranking data fields with the data type “small integer”.

st_nr2 d_name2 ... st_nr2 d_name8 ...
: Lower-ranking data fields with the data type “small integer” (see section “Small integer” on page 52). The data fields contain the year, the month, the day, the hours, the minutes, the seconds, and the milliseconds. The data fields must have different names.

Example

The SQL data type and assigned COBOL data type for a timestamp:

SQL data type:

```
TIME_STAMP TIMESTAMP(3)
```

COBOL data type:

```
01 TIME-STAMP TIMESTAMP(3).
   02 YEAR      PIC S9(4)  BINARY.
   02 MONTH     PIC S9(2)  BINARY.
   02 DAY       PIC S9(2)  BINARY.
   02 HOUR      PIC S9(2)  BINARY.
   02 MINUTE    PIC S9(2)  BINARY.
   02 SECOND    PIC S9(2)  BINARY.
   02 MILLISECS PIC S9(3)  BINARY.
```

2.4.15 Vectors

Using OCCURS, you can define vectors for multiple columns which occur more than once. Only level numbers 02 through 49 may be selected.

ISO language subset

The OCCURS clause must not be used with host variables.

OCCURS clauses must not be nested in a DECLARE SECTION.

Example

Multiple columns with three occurrences, and the assigned COBOL data type:

SQL data type:

```
RGB(3)
    NUMERIC(2,2)
```

COBOL data type:

```
01 RGB.
    02 COLOR-COMP PIC SV99 OCCURS 3 TIMES.
```

2.5 SQL statements in an ESQL-COBOL program

An SQL statement must be introduced with EXEC SQL and concluded with END-EXEC. The SQL statements WHENEVER, DECLARE CURSOR, CREATE TEMPORARY VIEW and INCLUDE can be embedded at any point in the source code. All other SQL statements have to be embedded in a PROCEDURE DIVISION in ESQL-COBOL programs.

During precompilation, the SQL statements WHENEVER, DECLARE CURSOR, CREATE TEMPORARY VIEW and INCLUDE are removed by the ESQL precompiler or replaced with comment lines. The ESQL precompiler replaces all other embedded SQL statements with COBOL statements. Using the precompiler option PRECOMPILER-ACTION and the parameter ESQL-STATEMENTS, you can specify whether the original SQL statements and the EXEC SQL and END-EXEC strings should be removed from the source code or retained as comments. Lines that are not embedded between EXEC SQL and END-EXEC are included unaltered in the COBOL program generated by the ESQL precompiler. If END-EXEC is followed by a period, the period is included in the source code after the inserted COBOL statements. The resulting source code must be permitted in COBOL.

```
EXEC SQL
    SQL statement
END-EXEC
```

The EXEC SQL and END-EXEC strings and the embedded SQL statement must be placed in columns 8-72 of the COBOL source code. Column 7 must contain a blank. Columns 1-6 and 73-80 can be used as in COBOL. The SQL statement must not contain continuation lines with a hyphen in column 7. COBOL comment lines may be embedded between EXEC SQL and END-EXEC.

`EXEC SQL`
: A single string which must be written in full in a single line.

SQL statement
: There may only be one SQL statement between EXEC SQL and END-EXEC. Any number of blanks may be used within the SQL statement to separate the elements in the statement.

Temporary views and cursors have to have already been defined in the text of the ESQL-COBOL program before they are used for the first time. Host variables have to have been defined in the text of the ESQL-COBOL program before they are used in SQL statements. If a host variable is used in a DECLARE CURSOR statement, the definition of the host variable has to be visible to this cursor in each OPEN CURSOR statement.

END-EXEC
: A single string which must be written in full in a single line. The remainder of the line after END-EXEC is included unaltered in the COBOL program. It therefore has to be permitted at this location in COBOL.

Example 1

Embedding an UPDATE statement in an ESQL-COBOL program:

```
EXEC SQL
   UPDATE CONTACTS
      SET DEPARTMENT = :DEPARTMENT
      WHERE CONTACT_NUM = '11'
END-EXEC
```

Example 2

Embedding a WHENEVER statement:

```
PARAGRAPH01.
  EXEC SQL
     WHENEVER SQLERROR GOTO PARAGRAPH99
  END-EXEC
PARAGRAPH02.
```

Explanation:
During precompilation, the EXEC SQL and END-EXEC strings and the WHENEVER statement are removed. This means that END-EXEC must not be followed by a period because the following kind of source code is not permitted in COBOL:

```
PARAGRAPH01.
          .
PARAGRAPH02.
```

2.6 SQL comments in an ESQL-COBOL program

You can include SQL comments in SQL statements to document the program.

An SQL comment begins with the string **– –** and ends with the end of the line. The comment must be located between column 8 and column 72.

The string **– –** does not introduce a comment if it occurs in an alphanumeric literal, a name in double quotes, or in a data name. Alphanumeric literals and names in double quotes are described in the “SQL Reference Manual Part 1: SQL Statements” [2].

Example

Specifying SQL comments:

```
EXEC SQL
-- UPDATE DEPARTMENT
   UPDATE CONTACTS
      SET DEPARTMENT = :DEPARTMENT
      WHERE CONTACT_NUM = '11'   -- DEPARTMENT HAS BEEN CHANGED
END-EXEC
```

2.7 The communication area

Every ESQL-COBOL program must contain the communication area. Information on the SQL statements executed is stored in the communication area. This information can be queried in the ESQL-COBOL program and is important for error handling and success monitoring (see section “Error handling and success monitoring” on page 76). ESQL-COBOL applications should not alter the values of variables in the communication area.

2.7.1 Structure of the communication area

There are two variant forms of communication area in ESQL-COBOL. These only differ in terms of the SQLCODE data field. The codes returned by SQL statements are stored in the SQLCODE data field. Future versions of the SQL standard will not support the SQLCODE data field. In future, codes returned by SQL statements will only be placed in the SQLSTATE data field.

The variant without the SQLCODE data field

The communication area does not contain the SQLCODE data field. You should use this variant for new ESQL-COBOL applications that you are writing for SESAM/SQL.

The variant for previous versions of ESQL-COBOL

In this variant, the communication area contains the SQLCODE data field. You can use this variant for applications originally developed for previous versions of SESAM/SQL.

Variant without SQLCODE

```
01  SQLca.
    05  SQLstatementid   PIC   S9(9) BINARY.
    05  SQLcallcount     PIC   S9(9) BINARY.
    05  SQLidmark        PIC   X(16).
    05  SQLline          PIC   S9(9) BINARY.

01  SQLda.
    05  SQLda01          PIC   S9(4) BINARY.
        88 SQLda01val    VALUE mmm.
    05  SQLda02          PIC   S9(4) BINARY.
    05  SQLda03          PIC   S9(4) BINARY.
    05  SQLda04          PIC   S9(4) BINARY.
    05  SQLerrline       PIC   S9(4) BINARY.
    05  SQLerrcol        PIC   S9(4) BINARY.
    05  SQLda07          PIC   S9(4) BINARY.
    05  SQLda08          PIC   X(5).
    05  SQLCLI-SQLSTATE  redefines SQLda08 PIC X(5).
    05  SQLerrm          PIC   X(240).
    05  SQLda10          PIC   X.
    05  SQLda21          PIC   S(9)  BINARY.
    05  SQLda22          PIC   X(4)  BINARY.
    05  SQLda23          PIC   9(4)  BIANRY.
    05  SQLda24          PIC   X(2).
    05  SQLrowcount      PIC   S9(9) BINARY.
    05  SQLda99          PIC   X(nnn).
```

Variant for previous versions of ESQL-COBOL

```
01  SQLca.
    05  SQLstatementid   PIC   S9(9) BINARY.
    05  SQLcallcount     PIC   S9(9) BINARY.
    05  SQLidmark        PIC   X(16).
    05  SQLline          PIC   S9(9) BINARY.
    05  SQLCODE          PIC   S9(4) BINARY.

01  SQLda.
    .
    .      (Structure as in variant 1)
    .
```

`SQLca`

Data group that contains information on the SQL statements that have executed.

`SQLstatementid`

`SQLcallcount`

`SQLidmark`
Reserved for internal use.

`SQLline`
Contains the line number in which the last SQL statement executed begins in the COBOL program generated by the ESQL compiler.

`SQLCODE`
Contains the return code of the SQL statement last executed. An overview of the return codes is provided in section “Error handling and success monitoring” on page 76. The individual return codes are described in the “Messages” manual [7].

This data field is only contained in the communication area variant which is compatible with previous versions of SESAM/SQL.

`SQLda`
Diagnostics area for SESAM/SQL.

`SQLda01, SQLda02, ... SQLda99`
Reserved for internal use. In future versions of SESAM/SQL the names and the sequence of these data fields may be changed. When writing applications you should therefore not rely on the sequence of these data fields and should not interpret their contents in the ESQL-COBOL program.

`SQLerrline`
If an error occurs, this contains the line number of the location in the text of a prepared statement at which an error occurred. It contains 0 (null) if it was impossible to detect the location, or if to do so would be of little value.

`SQLerrcol`
If an error occurs, this contains the column number of the location in the text of a prepared statement at which an error occurred. It contains 0 (null) if it was impossible to detect the location, or if to do so would be of little value.

`SQLerrm`
After the execution of an SQL statements that returns a value other than 00000, this contains an error message text. The text contains the error class (W for WARNING or E for ERROR), the message number SQL*nnnn*, and the message text.

`SQLrowcount`

Contains the following information after the execution of the relevant statements:

- after an INSERT statement, the number of rows inserted
- after an UPDATE or DELETE statement with a search condition, the number of rows for which the search condition was fulfilled
- after an UPDATE or DELETE statement without a WHERE clause, the number of records in the referenced table
- after an UNLOAD statement, the number of rows unloaded
- after a LOAD statement, the number of rows loaded
- after an EXPORT statement, the number of rows which were copied into the export file
- after an IMPORT statement, the number of rows which were copied from the export file

Otherwise the contents are not defined.

2.7.2 Making the communication area available

The communication area must be inserted in the ESQL-COBOL program's DATA DIVISION. To do this, you use the SQL-INCLUDE statement or the COBOL-COPY statement. If you use the SQL-INCLUDE statement, you must assign the library with the communication area during precompilation as an INCLUDE library (see section "INCLUDE elements" on page 79). If you use the COBOL-COPY statement, you must assign the library with the communication area as the COBOL-COPY library during compilation with the aid of the link name COBLIB.

The following libraries contain the communication area:

- SYSLIB.ESQL-COBOL.030.INCL-V2

  contains the communication area without SQLCODE for ESQL-COBOL applications for SESAM/SQL.

ESQL-COBOL applications for SESAM/SQL

You should make available a communication area without SQLCODE for ESQL-COBOL applications for SESAM/SQL. In addition, you must define the data field SQLSTATE or SQLCODE in a DECLARE SECTION. It is recommended that you use SQLSTATE.

Defining SQLSTATE or SQLCODE

SQLSTATE or SQLCODE has to be defined in the source code before the first SQL statement. SQLSTATE has to be defined in a DECLARE SECTION. SQLCODE can be defined inside or outside a DECLARE SECTION.

If SQLSTATE and SQLCODE are both defined in a program, the SQLCODE field is not supplied with a value. SQLCODE always has the value 0. Program control operations based on SQLCODE will not then function correctly.

The data type of SQLSTATE must correspond to the SQL data type CHAR(5). The data type of SQLCODE must correspond to the SQL data type SMALLINT. The OCCURS clause must not be specified in the definition of SQLSTATE or SQLCODE, not even in a higher-ranking data field.

Example

Defining SQLSTATE and adding a communication area without SQLCODE. A prerequisite here is that the library SYSLIB.ESQL-COBOL.030.INCL-V2 is assigned as an INCLUDE library.

```
 DATA DIVISION. WORKING-STORAGE SECTION.
*********************************************
* DEFINITION OF SQLSTATE
*********************************************
EXEC SQL BEGIN DECLARE SECTION END-EXEC
01 SQLSTATE PIC X(5).

.
.
EXEC SQL END DECLARE SECTION END-EXEC
*********************************************
* INSERTION OF COMMUNICATION AREA
*********************************************
EXEC SQL
   INCLUDE SQLCA
END-EXEC
.
.
.
```

2.7.3 Error handling and success monitoring

In order to make sure that database processing is working correctly, you must check whether each executable SQL statement was completed successfully after it has executed. You do this by querying the result in the SQLSTATE or SQLCODE data field. After each executable SQL statement, SQLSTATE or SQLCODE contains the statement's return code. The individual return codes are described in the "Messages" manual [7]. Additional information on error handling and success monitoring can be obtaned by querying the communication area (see section "The communication area" on page 71).

The table below provides an overview of the return codes placed in SQLSTATE or SQLCODE and their meanings.

SQLSTATE	SQLCODE	Meaning
00000	0	The SQL statement completed successfully.
01*nnn*	0 < *nnn* < 100	Warning. The SQL statement was executed.
02*nnn*	100	No data was read. The end of the table was reached, or the table was empty.
remaining values	< 0	Error. The SQL statement was not executed.
40*nnn*	< -1000	Error. The transaction was rolled back.

Table 6: Overview of the return codes in SQLSTATE and SQLCODE

You have two options for adjusting the course of program execution in response to the return code:

– with COBOL statements

– with the SQL statement WHENEVER

2.7.3.1 Controlling program execution with COBOL statements

You can use COBOL statements and your own error-handling routines to query the return code in SQLSTATE or SQLCODE and respond to any errors that have occurred.

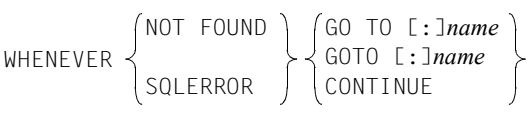
Example

Querying SQLSTATE and selecting the appropriate branch:

```
ANALYSE-SQL-ERROR.
  EVALUATE SQLSTATE(1:2) ALSO SQLSTATE(3:3)
     WHEN "00" THROUGH "01" ALSO ANY
          CONTINUE
     WHEN "02"              ALSO “000”
          PERFORM CURSOR-AT-END
     .
     .
     .
  END-EVALUATE.
```

2.7.3.2 Controlling program execution with the SQL statement WHENEVER

You use the SQL statement WHENEVER to define the actions to be carried out if the end of a table is reached, the table is empty, or an error has occurred. The WHENEVER statement does not allow a selective response to individual error situations.



`NOT FOUND`
The end of the table was reached or the table is empty.

`SQLERROR`
Error. Differentiation between different error situations is not possible here.

`GO TO [:]`*name* or `GOTO [:]`*name*
Specifies the point at which program execution is to continue. *name* is a chapter name or paragraph name. The paragraph name must not be qualified. The label specified must comply with COBOL conventions. The colon is supported to maintain compatibility with previous SESAM/SQL versions, but should not be used in new applications.

`CONTINUE`
Cancels exception handling. Branching does not take place.

The WHENEVER statement remains valid until one of the following conditions occurs:

– The source code contains a WHENEVER statement for the same error class (NOT FOUND or SQLERROR).

– The source code contains a CONTINUE which cancels the WHENEVER statement for the error class in question.

Example

Specifying and canceling exception handling:

```
*****************************************************************
* SPECIFICATION OF EXCEPTION HANDLING
*****************************************************************
  EXEC SQL
    WHENEVER SQLERROR GOTO REPORTING
  END-EXEC
.
.
.
*****************************************************************
* CANCELATION OF EXCEPTION HANDLING
*****************************************************************
  EXEC SQL
    WHENEVER SQLERROR CONTINUE
  END-EXEC
.
.
.
REPORTING SECTION.
*****************************************************************
* IF ERROR OCCURS, DISPLAY MESSAGE AND ROLL BACK TRANSACTION
*****************************************************************
DISPLAY "ERROR !" UPON DSG

EXEC SQL
  ROLLBACK WORK
END-EXEC
```

2.8 INCLUDE elements

You can store sections of code that you use repeatedly as INCLUDE elements in a PLAM library and add them to an ESQL-COBOL program as and when necessary with the SQL statement INCLUDE. During precompilation, the ESQL precompiler replaces the SQL statement INCLUDE and the EXEC SQL and END-EXEC strings with the code of the INCLUDE element.

An INCLUDE element may contain ESQL-COBOL or COBOL code. INCLUDE elements may also include the SQL statement INCLUDE (nesting). The maximum nesting depth is nine.

The SQL statement INCLUDE is an extension to the SQL standard. If you specify the ISO language subset with the precompiler option SOURCE PROPERTIES (see section “Specifying the properties of the ESQL-COBOL program” on page 101), you cannot use the SQL statement INCLUDE.

Using the precompiler option INCLUDE-LIBRARY, you can specify the libraries from which INCLUDE elements can be inserted (see section “Specifying INCLUDE libraries” on page 92).

Example

Assigning the SQL-INCLUDE library ESQL.INCLUDE.LIB.1 and the SQL-INCLUDE element ERRCOP:

1. Assigning the SQL-INCLUDE library with the precompiler option:

```
//PRECOMPILE    -
//INCLUDE-LIBRARY = ESQL.INCLUDE.LIB.1  -
  .
  .
  further precompiler options
  .
  .
//END
```

2. Inserting an SQL-INCLUDE element from the SQL-INCLUDE library into an ESQL-COBOL program with an SQL-INCLUDE statement:

```
DATA DIVISION.
WORKING-STORAGE SECTION.

  .
  .
  EXEC SQL
    INCLUDE ERRCOP
  END-EXEC
  .
  .
```

Explanation: During precompilation, the source code

```
EXEC SQL
  INCLUDE ERRCOP
END-EXEC
```

is replaced with the contents of the SQL-INCLUDE elements ERRCOP.

3 Precompiling an ESQL-COBOL program

This chapter covers the following topics:

- Calling and controlling the ESQL precompiler
- ESQL precompiler options
- Termination behavior of the ESQL precompiler and error messages

3.1 Calling and controlling the ESQL precompiler

You can call the ESQL precompiler interactively or from a procedure. You can set the options used to control the ESQL precompiler when you call it interactively or start it from a procedure. You can also specify the precompiler option SOURCE-PROPERTIES to define the properties of the ESQL-COBOL program directly in the ESQL-COBOL program.

Specifying the precompiler option SOURCE-PROPERTIES in the ESQL-COBOL program

Using the following statement, you can specify the precompiler option SOURCE-PROPERTIES in the ESQL-COBOL program:

```
//PRECOMPILE SOURCE-PROPERTIES=parameters
//END
```

You may only specify one PRECOMPILE statement. This must begin in the first line and first column of the ESQL-COBOL program.

The precompiler option specified in the ESQL-COBOL program takes precedence over any other options specified (e.g. precompiler options specified when the precompiler is called interactively). The REPLACE-TOKENS parameter constitutes an exception here: all the options specified are evaluated. A detailed description of the precompiler option SOURCE-PROPERTIES is provided in section “Specifying the properties of the ESQL-COBOL program” on page 101.

3.1.1 Assigning the requisite libraries and files

Before you start the ESQL precompiler, you must assign the following libraries:

– the SESAM/SQL module library
– the CRTE library

Assigning the SESAM/SQL module library

To do this, you can use any of the following statements:

```
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-modlib                (1)
/SET-TASKLIB LIBRARY=sesam-modlib                                       (2)
/SET-FILE-LINK LINK-NAME=BLSLIBnn,FILE-NAME=sesam-modlib                (3)
```

(1) Assigns the SESAM module library with the link name SESAMOML.

(2) Assigns the SESAM module library as TASKLIB.

(3) Assigns the SESAM module library as BLSLIB with $00 \leq nn \leq 99$

Assigning the CRTE library

The CRTE library must be assigned as BLSLIB. It is recommended that you use BLSLIB00:

```
/SET FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=crte-lib
```

3.1.2 Precompiling with database contact

When precompiling with database contact, you usually also assign a configuration file with CONNECT-SESAM-CONFIGURATION or with the link name SESCONF. If you do not assign a configuration file, the defaults apply. Detailed descriptions of the configuration files for the independent and linked-in DBHs are provided in the Core Manual [1].

3.1.3 Starting the ESQL precompiler

You can start the ESQL precompiler with the command START-ESQLCOB or the command START-PROGRAM. If you wish to monitor the ESQL program with a job variable, you have the following options:

– You use the command START-ESQLCOB and specify the precompiler option MONJV (see section “Specifying a job variable” on page 95).

– You use the command START-PROGRAM and include the parameter MONJV in the call.

Starting the ESQL precompiler with START-ESQLCOB

You can start the ESQL-COBOL precompiler as follows using the START-ESQLCOB command:

```
START-ESQLCOB precompiler-option ,...
```

START-ESQLCOB
: ESQL precompiler call.

precompiler-option
: ESQL precompiler option for controlling precompilation. A description of the individual ESQL precompiler options is provided in section “ESQL precompiler options” on page 87.

Starting the ESQL precompiler with START-PROGRAM

A detailed description of the START-PROGRAM command is provided in the BS2000 manual “BS2000/OSD-BC Commands, Volumes 1 - 5” [18]. You can start the ESQL precompiler with the START-PROGRAM command as follows:

```
/START-PROGRAM FROM-FILE=*MODULE(LIBRARY=precompiler_bibl,ELEMENT=ESQLCOB -   (1)
/                                    ,PROGRAM-MODE=ANY -                     (2)
/                                    ,RUN-MODE=ADVANCED -                    (3)
/                                    (ALTERNATE-LIBRARIES=YES -
/                                    ,LOAD-INFORMATION=REFERENCES -
/                                    ,UNRESOLVED-EXTRNS=DELAY) -
/                                    )
//PRECOMPILE precompiler-option, ...                                         (4)
//END                                                                        (5)
```

(1) ESQL precompiler call. *precompiler_lib* is the library which contains the precompiler. The standard library name is SYSLNK.ESQL-COBOL.030.

(2) The load unit’s modules can be loaded above or below 16Mbytes.

(3) The DBL, which is called implicitly by START-PROGRAM, operates in a mode that supports new functions (as of BS2000 V10.0A). This information is required to allow the processing of link and load modules. Alternative libraries are scanned. Any unresolved external references are resolved later.

(4) Introduces the precompiler options.
The precompiler option MONJV is not permitted here. If you wish to monitor the ESQL precompiler with job variables, you must specify the MONJV parameter in the START-PROGRAM command. A detailed description of the individual ESQL precompiler options is provided in section “ESQL precompiler options” on page 87.

(5) Indicates the end of the ESQL precompiler options.

The command sequence for precompiling an ESQL-COBOL program:

```
/SET-FILE-LINK LINK-NAME=BLSLIBnn,FILE=crte-lib                                (1)
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-modlib                       (2)
/CONNECT-SESAM-CONFIGURATION TO-FILE=global-configuration-file, -
/                                    CONFIGURATION-LINK=linkname
or
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=filename                            (3)
/START-ESQLCOB -                                                               (4)
/      SOURCE=esql-cobol-program -                                             (5)
/     ,INCLUDE-LIBRARY=SYSLIB.ESQL-COBOL.030.INCL-V2 -                         (6)
/     ,PRECOMPILER-ACTION=(SQL-ENTRY-NAME=entry-name) -                        (7)
/     ,SOURCE-PROPERTIES=(CATALOG=database-name -
/                        ,SCHEMA=schema-name -
/                        ,AUTHORIZATION='authorization-key' -
/                        )                                                     (8)
```

(1) Assigns the CRTE library as BLSLIB. It is recommended that you use BLSLIB00.

(2) Assigns the SESAM/SQL module library.

(3) Assigns the configuration file. This is only necessary if precompilation with database contact is to be carried out (see section “Controlling precompilation” on page 96).

(4) Calls the ESQL precompiler.

(5) The ESQL-COBOL program that is to be precompiled (see section “Controlling precompilation” on page 96).

(6) The library from which the communication area is inserted (see section “Specifying INCLUDE libraries” on page 92).

(7) Entry point for the SQL link and load module generated by the ESQL precompiler (see section “Controlling precompilation” on page 96).

(8) Specifies default database names, schema names and authorization keys (see section “Specifying the properties of the ESQL-COBOL program” on page 101).

3.2 ESQL precompiler options

You control how the ESQL precompiler works and executes using ESQL precompiler options. You specify ESQL precompiler options in SDF format.

This section provides an overview of the control options you can use and describes the individual ESQL precompiler options in alphabetical order.

3.2.1 Overview of the ESQL precompiler options

The tables that follow group together the ESQL precompiler options according to their contents:

– specification of input sources

– specification of properties of the ESQL-COBOL program

– control of precompilation

– specification of output targets

– specification of job variables

3.2.1.1 Specifying input sources

Precompiler option **INCLUDE-LIBRARY**

Parameter	Brief description	Possible information
	INCLUDE library	PLAM libraries

Table 7: Precompiler option INCLUDE-LIBRARY

Precompiler option **SOURCE**

Parameter	Brief description	Possible information
	Specifies an ESQL-COBOL program	– *SYSDATA – cataloged file – library element

Table 8: Precompiler option SOURCE

3.2.1.2 Specifying the properties of the ESQL-COBOL program

Precompiler option **SOURCE-PROPERTIES**

Parameter	Brief description	Possible information
HOST-LANGUAGE	Specifies the host language	– COBOL
ESQL-DIALECT	Specifies the language subset	– SQL standard with SESAM-specific extensions – SQL standard – SESAM-SQL V1.1
QUOTATION-CHARACTER	Specifies the delimiter for character strings	Quotes – single – double
CATALOG	Specifies the default database name	– via an option – via an embedded option in the ESQL-COBOL program
SCHEMA	Specifies the default schema name	– via an option – via an embedded option in the ESQL-COBOL program
AUTHORIZATION	Specifies access authorization for the SQL statement	– via an option – via an SQL statement
REPLACE-BY-FILE REPLACE-TOKENS	Replacements used in the conversion of existing applications for the current version of SESAM/SQL	Replacements – are read in from a file – are specified as a list

Table 9: Precompiler option SOURCE-PROPERTIES

3.2.1.3 Controlling precompilation

Precompiler option **PRECOMPILER-ACTION**

Parameter	Brief description	Possible information
DATABASE-CONTACT	Database contact during precompilation	– with database contact – without database contact
SQL-ENTRY-NAME	Specifies the entry point of the SQL link and load module	The relevant entry point
ESQL-STATEMENTS	Inclusion of SQL statements in the generated COBOL program	– inclusion as a comment – not included in the generated COBOL program
ESQL-XREF	Creation of a cross-reference list as a comment	– yes – no

Table 10: Precompiler option PRECOMPILER-ACTION

Specifying output targets

Precompiler option **HOST-PROGRAM**

Parameter	Brief description	Possible information
	Specifies the output target for the COBOL program	– cataloged file – library element

Table 11: Precompiler option HOST-PROGRAM

Precompiler option **MODULE-LIBRARY**

Parameter	Brief description	Possible information
	Defines destination for SQL link and load module	Library member

Table 12: Precompiler option MODULE-LIBRARY

Specifying job variables

Precompiler option **MONJV**

Parameter	Brief description	Possible information
	Specifies job variables for monitoring the ESQL precompiler	desired job variable

Table 13: Precompiler option MONJV

3.2.2 Specifying the output target for the generated COBOL program

You specify the output target for the COBOL program generated by the ESQL compiler with the precompiler option **HOST-PROGRAM**.

HOST-PROGRAM

HOST-PROGRAM = ***STD-FILE** / <full-filename 1..54 without-gen-vers> / ***LINK(...)** /
***LIBRARY-ELEMENT(...)**

***LINK(...)**
LINK-NAME = <full-filename 1..8 without-gen-vers>

***LIBRARY-ELEMENT(...)**
LIBRARY = <full-filename 1..54 without-gen-vers> / ***LINK(...)**
***LINK(...)**
LINK-NAME = <full-filename 1..8 without-gen-vers>
,ELEMENT = <full-filename 1..38 without-cat-user-gen-vers> (...)
VERSION = ***UPPER-LIMIT** / ***HIGHEST-EXISTING** / ***INCREMENT** / <text 1..24>

HOST-PROGRAM =
Specifies the output target for the COBOL program generated by the ESQL precompiler.

HOST-PROGRAM = *STD-FILE
The COBOL program generated is written to the cataloged file HOST.PROGRAM.

HOST-PROGRAM = <full-filename 1..54 without-gen-vers>
The COBOL program is stored in a cataloged file with the name specified.

HOST-PROGRAM = *LINK(...)

LINK-NAME = <full-filename 1..8 without-gen-vers>
The COBOL program generated is written to the cataloged file assigned via the link name.

HOST-PROGRAM = *LIBRARY-ELEMENT(...)
The generated COBOL program is stored in the specified element in the PLAM library.

LIBRARY = <full-filename 1..54 without-gen-vers>
The name of the PLAM library in which the generated COBOL program is stored.

LIBRARY = *LINK(...)
The PLAM library for the generated COBOL program is assigned via a link name.

LINK-NAME = <full-filename 1..8 without-gen-vers>
The link name of the PLAM library for the generated COBOL program.

ELEMENT = <full-filename 1..38 without-cat-user-gen-vers>(...)
The library element in which the COBOL program is stored. cat and user can be specified, but they will not be interpreted as such.

VERSION =
Specifies the version of the library element.

VERSION = *UPPER-LIMIT
The library element is assigned the highest possible version number (indicated by LMS with @).

VERSION = *HIGHEST-EXISTING
The library element with the highest version number is overwritten.

VERSION = *INCREMENT
The highest existing version number is incremented by one and assigned to the library element. A prerequisite here is that the highest existing version number must end with a digit. If it does not, an error message is displayed.

VERSION = <alphanum-name 1..24>
The library element is assigned the specified version ID. If the version ID is to be incremented, it must end with a digit.

3.2.3 Specifying INCLUDE libraries

You specify the libraries in which INCLUDE elements are to be found using the precompiler option **INCLUDE-LIBRARY**.

INCLUDE-LIBRARY

INCLUDE-LIBRARY = **<u>*NONE</u>** / list-poss(9): ***LINK**(...) / list-poss(9): <full-filename 1..54 without-gen-vers>

***LINK**(...)
LINK-NAME = <full-filename 1..8 without-gen-vers>

INCLUDE-LIBRARY =
Here you specify the PLAM libraries from which INCLUDE elements are to be inserted. The libraries are scanned in the sequence in which you specify them. You can specify up to nine libraries.

INCLUDE-LIBRARY = <u>*NONE</u>
No INCLUDE elements can be inserted.

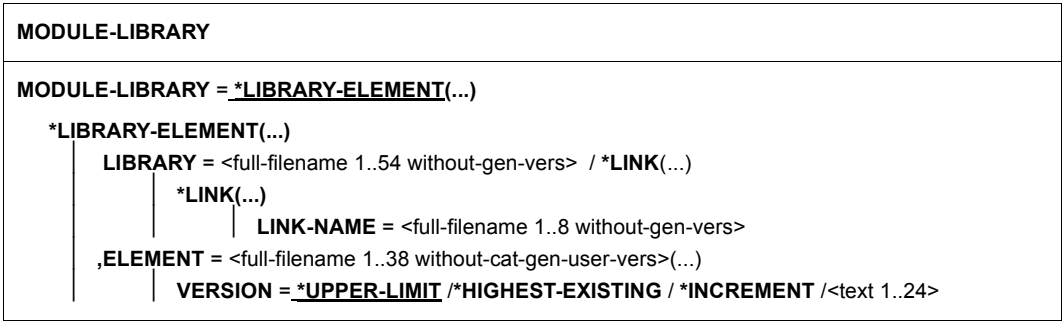
INCLUDE-LIBRARY = list-poss (9): *LINK(...)

LINK-NAME = <full-filename 1..8 without-gen-vers>
A list of references to libraries that are to be searched for INCLUDE elements.

INCLUDE-LIBRARY = list-poss(9): <full-filename 1..54 without-gen-vers>
A list of libraries that are to be searched for INCLUDE elements.

3.2.4 Specifying the output target for the SQL link and load module

You use the precompiler option **MODULE-LIBRARY** to specify the library to which the ESQL precompiler is to write the SQL link and load module during precompilation.



MODULE-LIBRARY = *LIBRARY-ELEMENT(...)
Specifies the PLAM library.

LIBRARY = <full-filename 1..54 without-gen>
The name of the PLAM library for the generated SQL link and load module. If the library does not already exist, it is created automatically. The default library is SQLPROG.LIB.

LIBRARY = *LINK(...)
Assigns the PLAM library for the generated SQL link and load module via a link name.

LINK-NAME = <full-filename 1..8 without-gen>
The link name of the PLAM library for the generated SQL link and load module.

ELEMENT = <full-filename 1..38 without-cat-user-gen-vers>(...)
The name of the library element to which the ESQL precompiler writes the generated SQL link and load module. The default is SQLPROG.OUT. cat and user can be specified, but they will not be interpreted as such.

VERSION =
Specifies the version of the library element.

VERSION = *UPPER-LIMIT
The library element is assigned the highest possible version number (indicated by LMS with @).

VERSION = *HIGHEST-EXISTING
The library element with the highest version number is overwritten.

VERSION = *INCREMENT
The highest existing version number is incremented by one and assigned to the library element. A prerequisite here is that the highest existing version number must end with a digit. If it does not, an error message is displayed.

VERSION = <alphanum-name 1..24>
The library element is assigned the specified version ID. If the version ID is to be incremented, it must end with a digit.

3.2.5 Specifying a job variable

The precompiler option **MONJV** may only be used if you start the ESQL precompiler with the START-ESQLCOB command (see section “Starting the ESQL precompiler” on page 84).

You use the precompiler option MONJV to specify a job variable for monitoring precompilation. During precompilation, the ESQL precompiler sets a status indicator and a return code in the job variable. For a description of possible status indicators and return codes, see section “Monitoring termination behavior with job variables” on page 108.

MONJV
MONJV = **_*NONE_** / <full-filename 1..54 without-gen-vers>

MONJV = _*NONE_
Precompilation is not monitored with a job variable.

MONJV = <full-filename 1..54 without-gen-vers>
Specifies the job variable used to monitor precompilation.

3.2.6 Controlling precompilation

You use the precompiler option **PRECOMPILER-ACTION** to control precompilation.

PRECOMPILER-ACTION

PRECOMPILER-ACTION = PARAMETERS(...)

PARAMETERS(...)
DATABASE-CONTACT = NO / YES(...)
YES(...)
CATALOG-CHECKS = STD / DYNAMIC
APPLICATION-TYPE = INDEPENDENT / LINKEDIN
,SQL-ENTRY-NAME = <name 1..7>
,ESQL-STATEMENTS = SOURCE-COMMENTS / REMOVED
,ESQL-XREF = NO / YES

PRECOMPILER-ACTION = PARAMETERS(...)
Controls how the ESQL precompiler operates.

DATABASE-CONTACT =
Specifies whether the ESQL-COBOL program is to be compiled with or without database contact.

DATABASE-CONTACT = NO
The ESQL-COBOL program is compiled without database contact. The database system does not need to be active during precompilation.

DATABASE-CONTACT = YES
The ESQL-COBOL program is compiled with database contact. The database system has to be active during precompilation. Before you start the ESQL precompiler you must assign a configuration file with the link name SESCONF (see section “Starting the ESQL precompiler” on page 84).

CATALOG-CHECKS =
If the ESQL-COBOL program is compiled with database contact, you can specify whether inconsistencies between the ESQL-COBOL program and the database are to be treated as errors or as warnings. Inconsistencies can occur, for example, when not all the tables planned for a database have been created. If just errors in the error classes “note” or “warning” are detected during precompilation, an SQL link and load module is generated. If an error in the error class “error” occurs, no SQL link and load module is generated.

CATALOG-CHECKS = STD
Inconsistencies between the ESQL-COBOL program and the database are treated as errors.

CATALOG-CHECKS = DYNAMIC
Inconsistencies between the ESQL-COBOL program and the database are treated as errors in the error class “warning”. The SQL link and load module is generated, even if inconsistencies are detected. You have to adapt the database or the ESQL-COBOL program accordingly before running the ESQL-COBOL application.

APPLICATION-TYPE =
Here you specify whether the independent DBH or the linked-in DBH is to be used during precompilation with database contact. For information on the DBH, see the „Core Manual“ [1] and „Database Operation“ manual [5].

APPLICATION-TYPE = INDEPENDENT
The independent DBH is used during precompilation with database contact. The independent DBH can work with a number of different users. If you use the independent DBH, the application can access the database concurrently with other applications.

APPLICATION-TYPE = LINKEDIN
The linked-in DBH is used during precompilation with database contact. The linked-in DBH can only work with a single user. You should therefore use the linked-in DBH if you wish to access a database not accessed by any other applications, e.g., a private test database.

SQL-ENTRY-NAME = <name 1..7>
The entry point used to call the SQL link and load module in the COBOL program. You must specify the entry point explicitly. In a run unit comprising several ESQL-COBOL programs, the entry point must be unique. Because of the need to maintain compatibility with ESQL-COBOL V1.1, the ESQL precompiler appends a Q to the entry point.

ESQL-STATEMENTS =
This is used to specify whether SQL statements are to be retained as comments in the COBOL program.

ESQL-STATEMENTS = SOURCE-COMMENTS
SQL statements are retained as comments in the COBOL program.

ESQL-STATEMENTS = REMOVED
SQL statements are removed from the COBOL program.

ESQL-XREF =
Specifies whether a cross-reference list should be included in the information section at the end of the generated COBOL program. The cross-reference list contains database names, file names and host variables. For each occurrence of a host variable, the list contains the line number of the ESQL-COBOL program, details of the way the host variable was used (input, output, I/O, defining, or unknown), and the line number in the COBOL program generated by the ESQL precompiler.

3.2.7 Specifying the input source for the ESQL precompiler

You use the precompiler option **SOURCE** to specify the input source from which the ESQL precompiler reads the ESQL-COBOL program.

SOURCE

SOURCE = **<u>*SYSDTA</u>** / <full-filename 1..54 without-gen-vers> / ***LINK(...)** / ***LIBRARY-ELEMENT(...)**

***LINK(...)**
LINK-NAME = <full-filename 1...8 without-gen-vers>

***LIBRARY-ELEMENT(...)**
LIBRARY = <full-filename 1..54 without-gen-vers> / *LINK(...)
*LINK(...)
LINK-NAME = <full-filename 1..8 without-gen>
,ELEMENT = <full-filename 1..38 without-cat-user-gen-vers> (...)
VERSION = **<u>*HIGHEST-EXISTING</u>** / **UPPER-LIMIT** / <text 1..24>

SOURCE =
Specifies the input source.

SOURCE = <u>*SYSDTA</u>
The ESQL precompiler reads the ESQL-COBOL program from the SYSDTA system file. If SYSDTA is assigned to the console, you can enter the ESQL-COBOL program interactively. The ESQL precompiler displays an asterisk (*) as a prompt. You complete entry of the ESQL-COBOL program as follows:

```
Key K2
/EOF
/RESUME-PROGRAM
```

Using the BS2000 command ASSIGN-SYSDTA, you can redirect SYSDTA to a cataloged file or a PLAM library before you call the ESQL precompiler. You can then read the ESQL-COBOL program and the precompiler options from the cataloged file or the library element. However, it is recommended that you save the precompiler options and the ESQL-COBOL program separately.

SOURCE = <full-filename 1..54 without-gen-vers>
The ESQL-COBOL program is read from the cataloged file specified.

SOURCE = *LINK(...)
The ESQL-COBOL program is read from a cataloged file assigned via a link name.

LINK-NAME = <full-filename 1..8 without-gen-vers>
Specifies the link name used to assign the cataloged file.

SOURCE = *LIBRARY-ELEMENT(...)
Specifies the PLAM library and the library element from which the ESQL-COBOL program is read.

LIBRARY = <full-filename 1..54 without-gen-vers>
Name of the PLAM library.

LIBRARY = *LINK(...)
The ESQL-COBOL program is read from a PLAM library assigned via a link name.

LINK-NAME = <full-filename 1..8 without-gen-vers>)
Link name of the PLAM library.

ELEMENT = <full-filename 1..38 without-cat-user-gen-vers>
Name of the library element. cat and user can be specified, but they will not be interpreted as such.

VERSION =
Specifies the version of the library element.

VERSION = *HIGHEST-EXISTING
The ESQL precompiler reads the ESQL-COBOL program from the library element with the highest version number.

VERSION = *UPPER-LIMIT
The ESQL precompiler reads the ESQL-COBOL program from the library element with the highest possible version number (indicated by LMS with @).

VERSION = <alphanum-name 1..24>
The ESQL precompiler reads the ESQL-COBOL program from the specified version of the library element.

3.2.8 Specifying the properties of the ESQL-COBOL program

You use the precompiler option **SOURCE-PROPERTIES** to specify the properties of the ESQL-COBOL program.

```
SOURCE-PROPERTIES

SOURCE-PROPERTIES = PARAMETERS(...) / STD
    PARAMETERS(...)
        HOST-LANGUAGE = COBOL
        ,ESQL-DIALECT = ALL-FEATURES(...) / ISO(...) / OLD(...)
            (...)
                UTM-RESTRICTIONS = NO / YES
        ,QUOTATION-CHARACTER = ESQL-STANDARD / HOST-STANDARD
        ,CATALOG = *INLINE / <text 1..18>/ <c-string 1..18>
        ,SCHEMA = *INLINE / <text 1..31> / <c-string 1 ..31>
        ,AUTHORIZATION = *IMPLICIT / <text 1..18> / <c-string 1..18>
        ,REPLACE-BY-FILE = *NONE / <full-filename 1..54 without-gen-vers> / *LINK(...) /
                                   LIBRARY-ELEMENT(...)
            *LINK(...)
                LINK-NAME = <full-filename 1..8 without-gen-vers>
            LIBRARY-ELEMENT(...)
                LIBRARY = <full-filename 1..54 without-gen> / *LINK(...)
                *LINK(...)
                        LINK-NAME = <full-filename 1..8 without-gen-vers>
        ,ELEMENT = <full-filename 1..38 without-cat-user-gen-vers(...)
                VERSION = HIGHEST-EXISTING / UPPER-LIMIT / <text 1..24>
        ,REPLACE-TOKENS = *NONE /list-poss(2000):*SUBSTITUTE(...)
            SUBSTITUE(...)
                IDENTIFIER = <c-string 1..128>
                ,REPLACEMENT = <c-string 1..128>
```

SOURCE-PROPERTIES = PARAMETERS
Specifies the properties of the ESQL-COBOL program.

HOST-LANGUAGE = COBOL
The ESQL program's host language.

ESQL-DIALECT =
Selects the language subset. This parameter facilitates the porting of ESQL-COBOL programs developed for earlier SESAM/SQL versions or other database systems. If the ESQL program contains SQL elements that are not permitted in the chosen language subset, the ESQL precompiler outputs error messages and does not create an SQL link and load module.

ESQL-DIALECT = ALL-FEATURES(...)
Selects the SQL standard ISO/IEC 9075:2003 dialect and SESAM-specific extensions. This option is recommended if you wish to use all the available features of SESAM/SQL.

ESQL-DIALECT = ISO(...)
Selects the SQL standard ISO/IEC 9075:2003 as the permitted dialect. This option is recommended when precompiling ESQL-COBOL programs developed for other SQL database systems. If this option is selected, only those keywords are reserved that are defined in the SQL standard. The keywords reserved additionally by SESAM/SQL are no longer treated as reserved when this option is selected.

ESQL-DIALECT = OLD(...)
Selects the dialect supported by SESAM/SQL V1.1. This option can be specified when precompiling ESQL-COBOL programs developed for SESAM/SQL V1.1. Keywords reserved in the current SESAM/SQL version are not treated as reserved and can be used as names.

UTM-RESTRICTIONS =
Specifies whether the dialect used in the ESQL-COBOL program should be checked for compliance with UTM. In UTM operations, only UTM language constructs are permitted for transaction management. The SQL statements COMMIT WORK and ROLLBACK WORK may therefore not be used for committing or rolling back transactions.

UTM-RESTRICTIONS = NO
The SQL statements COMMIT WORK and ROLLBACK WORK are permitted.

UTM-RESTRICTIONS = YES
The SQL statements COMMIT WORK and ROLLBACK WORK are not permitted. During program execution, the SQL statements PREPARE and EXECUTE IMMEDIATE for COMMIT WORK and ROLLBACK WORK cause an error.

QUOTATION-CHARACTER =
Defines how alphanumeric literals and names in quotes are delimited in SQL statements. In the case of literals in COBOL text, the delimiters are defined during translation with the COBOL compiler (see the „COBOL2000 (BS2000/OSD)) User's Guide“ [17]).

QUOTATION-CHARACTER = ESQL-STANDARD
Quotation marks are used in accordance with the SQL standard: alphanumeric literals are enclosed in single quotes; names in quotes are enclosed in double quotes.

QUOTATION-CHARACTER = HOST-STANDARD
Alphanumeric literals are enclosed in double quotes; names in quotes are enclosed in single quotes.

CATALOG =
Sets the default database name. In accordance with SQL rules, this database name is valid for all SQL objects of the ESQL-COBOL program for which you have not specified a database name. You must include the preset database name explicitly, even if you precompile without database contact.

CATALOG = *INLINE
The default database name is specified directly in the ESQL-COBOL program with the precompiler option SOURCE-PROPERTIES.

CATALOG = <text 1..18>/<c-string 1..18>
Sets the specified database name as the default database name. The specified database name must conform to the conventions governing database names (see the „SQL Reference Manual Part 1: SQL Statements“ [2]).

SCHEMA =
Sets the default schema name. In accordance with SQL rules, this name is valid for all SQL objects of the ESQL-COBOL program for which you have not specified a schema name. You must include the preset schema name explicitly, even if you precompile without database contact.

SCHEMA = *INLINE
The default schema name is specified directly in the ESQL-COBOL program with the precompiler option SOURCE-PROPERTIES.

SCHEMA = <text 1..31> / <c-string 1..31>
Sets the specified schema name as the default schema name. The specified schema name must be unique within the database.

AUTHORIZATION =
Specifies the authorization key.

AUTHORIZATION = *IMPLICIT
The authorization key is specified with the SQL statement SET SESSION AUTHORIZATION.

AUTHORIZATION = <text 1..18>/<c-string 1..18>
Specifies the authorization key. The authorization key specified here cannot be altered using the SQL statement SET SESSION AUTHORIZATION.

REPLACE-BY-FILE =
Specifies the replace-by file from which the replacement pairs are read.
The replace-by file has the following format:

```
//PRECOMPILE SOURCE-PROPERTIES= ( -
// ( -
// REPLACE-TOKENS= -
//  *SUBSTITUTE( '1st_string_to_replace' , 'replacement1' ) -
// ,*SUBSTITUTE( '2nd_string_to_replace' , 'replacement2' ) -
//       ...
// ))
// END
//PRECOMPILE SOURCE-PROPERTIES= -
// ( -
//   ...
// )
// END
```

The file may contain more than one PRECOMPILE statement. The END statement must be the last statement in the file. To aid debugging, it is recommended that replacement pairs be spread across a number of different PRECOMPILE statements.

Example

Qualifying the names of the temporary views and tables with the schema names. The example assumes that quotes are used in accordance with the SQL standard: names in quotes are delimited with double quotes (option SOURCE-PROPERTIES, parameter QUOTATION-CHARACTER = ESQL-STANDARD).

```
//PRECOMPILE SOURCE-PROPERTIES= ( -
// ( -
// REPLACE-TOKENS= -
//  *SUBSTITUTE( 'PERSONNELDATA' , 'MODULE.PERSONNELDATA' ) -
// ,*SUBSTITUTE( '"PROFIT&LOSS"' , 'MODULE."PROFIT&LOSS"' ) -
// ,*SUBSTITUTE( 'ITEMS' , 'PARTS_SCHEMA.ITEMS' ) -
// ))
// END
```

Explanation:
The names of the temporary views PERSONNELDATA and PROFIT&LOSS are qualified with MODULE. PROFIT&LOSS is a name in quotes. The ITEMS table is qualified with the PARTS_SCHEMA schema name.

REPLACE-BY-FILE = *NONE
No replace-by file is specified.

REPLACE-BY-FILE = <full-filename 1..54 without-gen-vers>
The ESQL precompiler uses the cataloged file specified as the replace-by file.

REPLACE-BY-FILE = *LINK(...)
The ESQL precompiler uses a cataloged file assigned via a link name as the replace-by file.

LINK-NAME = <full-filename 1..8 without-gen-vers>
Link name of the cataloged file.

REPLACE-BY-FILE = *LIBRARY-ELEMENT(...)
Specifies the PLAM library and the library element from which replacement pairs are to be read.

LIBRARY = <full-filename 1..54 without-gen-vers>
Name of the PLAM library.

LIBRARY = *LINK(...)
The ESQL precompiler reads the replacement pairs from a PLAM library which is assigned via a link name.

LINK-NAME = <full-filename 1..8 without-gen-vers>
Link name of the PLAN library.

ELEMENT = <full-filename 1..38 without-cat-user-gen-vers>(...)
Name of the library element. cat and user can be specified, but they will not be interpreted as such.

VERSION =
Specifies the version of the library element.

VERSION = *HIGHEST-EXISTING
The ESQL precompiler reads the replacements pairs from the library element with the highest version number.

VERSION = *UPPER-LIMIT
The ESQL precompiler reads the replacement pairs from the library element with the highest possible version number (indicated by LMS with @).

VERSION = <alphanum-name 1..24>
The ESQL precompiler reads the synonyms from the specified library element.

REPLACE-TOKENS =
Performs replacements when converting or porting ESQL-COBOL applications. Names, names in quotes, and keywords can be replaced. The strings to be replaced and the relevant replacements can be specified in pairs or can be read from a replace-by file. All the options specified (when the precompiler is called, in the ESQL-COBOL program, and in a replace-by file) are evaluated on equal terms.
Replacement is governed by the following rules:

- Replacements are only carried out in the SQL source code. The SQL source code can be in the source code of the ESQL-COBOL program or can be inserted with the SQL statement INCLUDE.
- Names, names in quotes and keywords can be replaced.
- The string to be replaced must be fully contained in a single line.
- The type of delimiters for names in quotes (single or double quotes) is specified with the QUOTATION-CHARACTER parameter. Quotes that are to be interpreted as alphanumeric literals rather than as quotation marks have to be doubled up.

- Replacements are not performed in the following:
  - the SQL statements INCLUDE or WHENEVER
  - strings that follow EXEC SQL and begin with INCLUDE or WHENEVER
  - the strings EXEC SQL or END-EXEC
  - numeric literals
  - alphanumeric literals
  - names and host variables

REPLACE-TOKENS = *NONE
No replacements are carried out.

REPLACE-TOKEN = list-poss(2000): *SUBSTITUTE(...)
Specifies replacement pairs. Quotes that are to be interpreted as alphanumeric literals rather than as delimiters have to be doubled up. Doubled up quotes are treated as two characters.

IDENTIFIER = <c-string 1..128>
Strings that are to be replaced. You can specify names, names in quotes and keywords.

REPLACEMENT = <c-string 1..128>
Replacements for the strings specified with IDENTIFIER.

SOURCE-PROPERTIES = STD
The default settings apply for all parameters.

3.3 ESQL precompiler termination behavior

The precompiler's termination behavior depends on whether errors occurred during precompilation and, if so, on the error class to which the errors in question belonged. This section describes the monitoring of termination behavior with job variables, the structure of error messages and how they are output and the creation of diagnostic documentation.

3.3.1 Monitoring termination behavior with job variables

The ESQL precompiler's termination behavior is particularly important when it is called from a procedure. You can monitor the execution and termination behavior of the precompiler with job variables.

The following tables provides you with an overview of possible termination states, their effects on subsequent execution of a procedure, and the contents of a job variable.

Error	**Termination**	**Job variable**		**Behavior in procedure**
		Status indicator	**Return code**	
No error	Normal SQL link and load module is generated.	$T	0000	No branching
Messages in the error class "note"			0001	
Messages in the error class "warning"			1002	
ESQL precompiler completes operation but outputs errors in the error class "error"	Error No SQL link and load module is generated.		2004	No branching to next STEP, ABEND, ABORT, END or LOGOFF command
Error, e.g. invalid option, input file does not exist, insufficient storage capacity			2005	
Internal error in ESQL precompiler		$A	3006	

Table 14: Termination behavior of ESQL precompiler

3.3.2 Messages output by the ESQL precompiler

The ESQL-COBOL system normally outputs messages to SYSOUT and in the precompiled COBOL program. Informational messages, such as messages at the start or end of precompilation, are only output to SYSOUT. If output in the COBOL program is impossible because of the nature of the error, messages pertaining to serious internal errors or system errors are only output to SYSOUT.

The message line is located in the COBOL program below the line in which the error occurred.

Messages output by the ESQL-COBOL system have the following structure:

Message number Position Error class – Message text

Message number

Each message number is unique, and you can check up on the associated message text in chapter “Messages output by the ESQL-COBOL system”. The messages are grouped in ranges of numbers as follows:

Range	Meaning
IQB0000 - IQB0899	Messages pertaining to precompiler options, I/O, storage administration, and module generation
IQB0900 - IQB0999	Internal ESQL precompiler errors
IQB1000 - IQB1999	Messages pertaining to syntax analysis
IQB2000 - IQB2999	Messages pertaining to semantic analysis
IQB3000 - IQB3099	ESQL construct-specific messages
IQB9000 - IQB9099	Informational messages

Position Error position in the format *lllll*:*fff*.

lllll Number of the line in which the error occurred.

ddd The file that contains the error. A 1 indicates the COBOL program. INCLUDE files are numbered in sequence. The INCLUDE files and the associated numbers are listed in the information section of the generated COBOL program.

If errors occur which are impossible to assign to a specific line, the position information is not included.

Error class Indication of the severity of the error. The error classes have the following meanings:

N Note
No serious error has occurred. However, it is recommended that you check the relevant location in the ESQL-COBOL program. Precompilation continues and the SQL link and load module is generated.

W Warning
The precompiler has detected an inconsistency or an item that will no longer be supported in a future version. The error will not have any immediate or serious consequences, but it is recommended that you check the relevant location in the program. Precompilation continues; the link and load module can be generated.

E Error
The ESQL precompiler has detected a lexical, syntactic or semantic error. Precompilation continues, but the SQL link and load module is not generated. Correct the ESQL-COBOL program and restart precompilation.

F Fatal
A fatal error caused precompilation to abort. The error could be in the precompiler options or a system error in the ESQL-COBOL system. The SQL link and load module is not generated. Correct the error in question and restart precompilation.

S System
Internal ESQL precompiler error. The SQL link and load module is not created. Prepare appropriate diagnostic documents and contact the relevant systems support staff (see section “Creating diagnostic documents” on page 111).

Message text The message text describes the error that occurred.

The ESQL precompiler also outputs the following information at the end of the COBOL program:

– The total number of errors that occurred.
– The number of errors in each individual error class.
– The precompiler options that were set.
– The line numbers of references to database names in the ESQL-COBOL program and the COBOL program.
– The name and number of the ESQL-COBOL program, the INCLUDE files used, and the line numbers of the ESQL-COBOL programs and the COBOL program in which the INCLUDE files are referenced or linked.

– List of cross-references, showing the names of all host variables and the type of reference, provided the parameter XREF=YES was set in the precompiler option PRECOMPILER-ACTION (see section “Controlling precompilation” on page 96).

A full list of messages is provided in chapter “Messages output by the ESQL-COBOL system” on page 153.

3.3.3 Creating diagnostic documents

In order to have access to the necessary diagnostic documents it is advisable to log the execution of ESQL-COBOL compilation runs. You can do this with the following BS2000 command:

```
MODIFY-JOB-OPTIONS LOGGING=PARAMETERS(LISTING=YES)
```

When internal ESQL precompiler errors occur (numbers in the range IQB0900 - IQB0999), create the following diagnostic information for the systems support staff responsible:

– the ESQL-COBOL program used

– the COBOL program created, if available

– the version number of the ESQL precompiler

– the version number of the CRTE and the SQL run-time system, or the SESAM/SQL server

– a dump, if you are given the option

– a list of ESQL precompiler options that were set

– the log of job execution

The ESQL precompiler’s version number is displayed on screen when you start the precompiler. It is also included in the COBOL program created. Ask your system administrator for the version numbers of the CRTE system and the SQL run-time system. The ESQL precompiler options that were set are listed at the end of the COBOL program created. If you have enabled logging of job execution with the BS2000 command MODIFY-JOB-OPTIONS, the log is written to SYSLST.

4 Compiling the COBOL program

You compile the COBOL program created by the ESQL precompiler with the COBOL2000 COBOL compiler.

If you use new language elements which were only implemented with Version V3.0 together with the COBOL85 Compiler, incompatibilities can occur.

The COBOL compiler is described in the “COBOL2000 (BS2000/OSD) Reference Manual” [16] and the “COBOL2000 (BS2000/OSD)) User’s Guide” [17].

The command sequence for compiling a COBOL program:

```
/START-COBOL2000-COMPILER -                                    (1)
/SOURCE=cobol-program -                                        (2)
/,COMPILER-ACTION=MODULE-GENERATION                            (3)
```

(1) Loads and starts the COBOL compiler. The entries are read from the console.

(2) Assigns the COBOL program that is to be compiled. Using *cobol_program*, specify the output target set with the ESQL precompiler’s HOST-PROGRAM option. If you did not set an output target, specify the default output target HOST.PROGRAM here.

(3) Specify the compilation steps to be carried out. The COBOL compiler carries out a full compilation. If it completes without errors, a COBOL link module is generated.

5 Linking an ESQL-COBOL application

To link an ESQL-COBOL application, you require the linker BINDER and the dynamic linking loader DBL. Detailed descriptions of the BINDER and the DBL are provided in the manuals “Binder in BS2000/OSD” [19] and “Dynamic Binder Loader / Starter in BS2000/OSD” [20].

During linking, the following modules are combined to create an executable load unit:

– the SQL link and load module created by the ESQL precompiler

– the COBOL link module created by the COBOL compiler

– the SESAM/SQL connection module for the independent or linked-in DBH

The COBOL runtime system can be linked with the ESQL-COBOL application. However, it is recommended that the runtime system be dynamically loaded when the ESQL-COBOL application starts. This makes the ESQL-COBOL application more compact, and the latest run-time system is always used when the ESQL-COBOL application is started.

Linking an ESQL-COBOL application:

```
/START-BINDER                                                              (1)
//START-LLM-CREATION INTERNAL-NAME=internal_name                           (2)
//INCLUDE-MODULES LIBRARY=library,ELEMENT=(sql-link-and-load-module)       (3)
//INCLUDE-MODULES LIBRARY=library,ELEMENT=(cobol-link-module)              (4)
//INCLUDE-MODULES LIBRARY=SYS.MOD,ELEMENT=sesam/sql-connection-module      (5)
//RESOLVE-BY-AUTOLINK LIBRARY=crte-lib                                     (6)
//SAVE-LLM LIBRARY=library,ELEMENT=element                                 (7)
//END
```

(1) Calls BINDER.

(2) Creates a link and load module (LLM) with the internal name *internal_name*.

(3) Links in the SQL link and load module. Enter the names of the library and the element you selected as the output target for the SQL link and load module during precompilation with the precompiler option MODULE-LIBRARY. If you did not set an output target, specify the default output target:
`LIBRARY = SQLPROG.PLIB, ELEMENT = SQLPROG.OUT`

(4) Links in the COBOL link module. Enter the names of the library and the element you selected as the output target during compilation with the COBOL compiler. If you did not set an output target, specify the default output target:
`LIBRARY=*OMF,ELEMENT=*ALL`

(5) Links in the SESAM/SQL connection module from the SESAM/SQL module library. If the ESQL-COBOL application is to work with the independent DBH, specify the element SESMOD. If the linked-in DBH is to be used instead, specify the element SESLINK.

(6) Links in the CRTE for the COBOL run-time system. This statement only has to be specified if the run-time system is linked statically. However, it is recommended that the run-time system be loaded dynamically when the ESQL-COBOL application is started (see chapter “Starting an ESQL-COBOL application” on page 117).

(7) Saves the created link and load module under the specified name as an element of the type L in the specified library.

Run units comprising a number of different ESQL-COBOL applications

You can also create a run unit (RUN UNIT) consisting of several compiled and executable ESQL-COBOL applications. It is important to pay attention to the following:

– Cursors and temporary views are only valid within the module in which they are defined and not in the whole of the run unit.

– The entry points you have defined with the precompiler option PRECOMPILER-ACTION have to be unique within the run unit (see section “Controlling precompilation” on page 96).

6 Starting an ESQL-COBOL application

Before you start an ESQL-COBOL application, you have to assign the SESAM/SQL module library to the link name SESAMOML. It is recommended that you load the COBOL run-time system dynamically when you start the application. To do this, you assign the CRTE library to BLSLIB. If the ESQL-COBOL application works with the linked-in DBH, the CRTE library must be assigned as BLSLIB, and PROGRAM-MOD=ANY must be specified in the START-PROGRAM command.

You can create and assign a configuration file. If you do not assign a configuration file, the defaults defined for the DBH name and the configuration apply. Detailed descriptions of the configuration files for the independent DBH and the linked-in DBH are provided in the “Core Manual” [1].

The command sequence used to start an ESQL-COBOL application:

```
/SET-FILE-LINK LINK-NAME=SESAMOML, FILE-NAME=sesam-modlib                    (1)
/SET-FILE-LINK LINK-NAME=BLSLIBnn, FILE-NAME=crte-lib                        (2)
/CONNECT-SESAM-CONFIGURATION TO-FILE=global-configuration-file, -
/                                    CONFIGURATION-LINK=linkname
or
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=filename                          (3)
/START-PROGRAM FROM-FILE=-                                                   (4)
/      *MODULE(LIBRARY=library, ELEMENT=element -                            (5)
/             ,PROGRAM-MODE=ANY -                                            (6)
/             ,RUN-MODE=ADVANCED -                                           (7)
/             (ALTERNATE-LIBRARIES =YES) -                                   (8)
/             )
```

(1) Assigns the SESAM/SQL module library.

(2) Assigns the CRTE library as BLSLIB. It is recommended that you use BLSLIB00. This needs to be assigned if the application is to work with the linked-in DBH, or the COBOL run-time system is to be loaded dynamically. To allow modules to be loaded from the CRTE library, ALTERNATE-LIBRARIES=YES has to be specified in the START-PROGRAM command.

(3) Assigns the configuration file.

(4) Loads and starts the ESQL-COBOL application.

(5) Calls the dynamic linking loader DBL to start the ESQL-COBOL application. The ESQL-COBOL application is loaded from the specified library or element. Specify the name of the library and the element that you specified in SAVE-LLM as the output target for the ESQL-COBOL application when it was linked.

(6) The entry is required if the ESQL-COBOL application works with the linked-in DBH.

(7) Sets the DBL's operating mode. This entry is required in order to process link and load modules.

(8) Specifies that modules are to be loaded dynamically from libraries assigned with BLSLIB. This entry is required if you wish to load the COBOL run-time system dynamically.

7 ESQL-COBOL applications under openUTM

This chapter describes important issues that require your attention when running an ESQL-COBOL application as a program unit in a UTM application.

Detailed information on UTM applications is provided in the following UTM manuals: „Generating and Handling Applications“ [11] and „Programming Applications“ [12].

7.1 The language subset under openUTM

Only UTM language constructs can be used for transaction logging in ESQL-COBOL applications for UTM operations. This means that you may not use the SQL statements COMMIT WORK or ROLLBACK WORK to commit or roll back transactions. Using the precompiler option SOURCE-PROPERTIES, you can verify the language subset used during precompilation (see section “Specifying the properties of the ESQL-COBOL program” on page 101).

UTM takes care of the synchronization of UTM and database transactions. When a UTM transaction is committed, UTM automatically completes the database transaction, too.

7.2 Generating a UTM application

This section describes special aspects that require attention when generating UTM applications with ESQL-COBOL program units. Generation is described in detail in the UTM manuals „Generating and Handling Applications“ [11] and „Programming Applications“ [12].

The KDCDEF statement DATABASE

In the KDCDEF statement DATABASE, you must specify SESSQL as the ENTRY name for SESAM/SQL. It is recommended that you load the UTM connection module SESUTMC dynamically. To do this, you specify the SESAM/SQL module library in the DATABASE statement using the LIB parameter. Specify the DATABASE statement as follows:

```
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesam-modlib
```

If the application also contains CALL-DML program units, you must also add the following DATABASE statement:

```
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=sesam-modlib
```

Compiling the KDCROOT connection program

Before compiling the KDCROOT connection program, you assign the SESAM module library. The statement depends on the assembler you use.

Example 1

Assigning the SESAM macro library with ASSEMBH:

```
/SET-FILE-LINK LINK-NAME=UTMLIB,FILE-NAME=utm_macro_library
/SET-FILE-LINK LINK-NAME=SESAMLIB,FILE=sesam_macro_library
/START-PROGRAM FROM-FILE=$ASSEMBH
//COMPILE  -
//SOURCE=kdcroot
//,MACRO-LIBRARY=(*LINK(LINK-NAME=SESAMLIB) -
.
.
```

Example 2

Assigning the SESAM macro library with ASSGEN:

```
/FILE sesam-macro-library,LINK=ALTLIB2
/EXEC $ASSGEN
*COMOPT ALTLIB2
.
.
```

Substitution of UTM programs

The entry point used to call the SQL link and load module during a UTM application must be unique.
When generating UTM applications with ESQL/COBOL program units:

– either use new entry points or

– ensure the entry points are unique by closing all those UTM application sessions with entry points already in use.

7.3 Starting a UTM application

Before you start a UTM application with ESQL-COBOL program units, you must assign the SESAM/SQL module library. It is recommended that you load the relevant run-time systems dynamically when you start the UTM application.

In a UTM application, the parameters for the SESAM/SQL connection module can be specified in a (global) configuration file or in the UTM start procedure. The specifications in the configuration file take precedence over the UTM start parameters, which are ignored if you specify a configuration file.
Refer to the "Core Manual" [1] for a description of the configuration file and the start parameters.

Add the following statements to the start procedure for the UTM application:

```
/SET-FILE-LINK LINK-NAME=SESAMOML, FILE-NAME=sesam-modlib                    (1)
.
.
/CONNECT-SESAM-CONFIGURATION TO-FILE=global-configuration file, -
/                                    CONFIGURATION-LINK=linkname
oder
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=filename                          (2)
/START-PROGRAM FROM-FILE=*MODULE(LIBRARY=userlib,ELEMENT=elementname -       (3)
/                               ,RUN-MODE=ADVANCED -
/                               (ALTERNATE-LIBRARIES =YES) -
/                               )
.UTM utm-start-parameters                                                    (4)
.FHS fhs-start-parameters                                                    (5)
.UTM END                                                                     (6)

.
.
```

(1) Assign the SESAM/SQL module library.

(2) Assigns the configuration file.

(3) Calls the dynamic linking loader DBL to start the UTM application.

(4) Specifies the UTM start parameters. The individual start parameters are described in the "Generating and Handling Applications" manual [11].

(5) Specifies the start parameters for the form generating system FHS. The individual start parameters are described in the "FHS (TRANSDATA)" manual [14].

(6) Concludes UTM start parameter entry.

8 Sample programs

The following sample programs provide an insight into the range of applications possible with ESQL-COBOL. All the sample programs are based on the sample database:

– Outputting data record by record, see section “The program QUERY” on page 124
– Updating data, see section “The program UPDATE” on page 131
– Inserting a record, see section “The program INSERT” on page 136
– Deleting a record, see section “The program DELETE” on page 140
– Specifying a dynamic SQL statement, see section “The program DYNAMIC” on page 145

After each SQL statement, a check is carried out to determine whether the statement was executed correctly. If an error has occurred, SQLSTATE contains the relevant return code. The system outputs the return code and the number of the line in the COBOL program created by the ESQL precompiler. In the interest of clarity, the indicator variables defined in the examples are not always evaluated.

8.1 The program QUERY

The program QUERY outputs all the orders issued by a company, which are contained in the ORDERS table. Users can specify the company for which they would like to see the orders. The program defines a cursor. The data in the cursor table is output record by record. The default database and the default schema are specified with the precompiler option SOURCE-PROPERTIES (CATALOG = SAMPLE, SCHEMA=ORDER_ADMIN).

```
*============================

 IDENTIFICATION DIVISION.

 PROGRAM-ID.    QUERY.

*============================

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SPECIAL-NAMES.

     TERMINAL IS DSG.

 INPUT-OUTPUT SECTION.

*============================

 DATA DIVISION.
*============================

*

 WORKING-STORAGE SECTION.

*
*** OUTPUT VARIABLES FOR ORDERS.
*

 01 CA-OUTPUT.
     02 CA-ORDER-NUM       PIC ZZZ9.
     02 CA-ORDER-DATE-Y    PIC Z9999.
     02 FILLER             PIC X VALUE "-".
     02 CA-ORDER-DATE-M    PIC 99.
     02 FILLER             PIC X VALUE "-".
     02 CA-ORDER-DATE-D    PIC 99.
     02 FILLER             PIC X VALUE " ".
     02 CA-ORDER-TEXT      PIC X(30).
     02 CA-ORDER-STATUS    PIC Z9.
```

```
*** OUTPUT VARIABLES FOR ERROR OUTPUT.
*
 01 OUTPUT.
     02 FILLER            PIC X(7) VALUE "ERROR ".
     02 O-SQLSTATE         PIC X(5).
     02 FILLER            PIC X(10) VALUE " IN LINE ".
     02 O-SQLLINE          PIC Z(8)9.

*** INSERT COMMUNICATION AREA.
*

     EXEC SQL
         INCLUDE SQLCA
     END-EXEC

*** HOST VARIABLES.
*

     EXEC SQL BEGIN DECLARE SECTION END-EXEC

*** SQLSTATE.
*

 01 SQLSTATE     PIC X(5).

*** FIELD FOR THE COMPANY NAME ENTERED.
*

 01 COMPANY-NAME   PIC X(40).

*** FIELD FOR THE CUSTOMER NUMBER FOUND.
*

 01 CUSTOMER-NUM     PIC S9(9) USAGE IS BINARY.

*** HOST VARIABLES FOR ORDERS.
*

 01 ORDERS.
     02 ORDER-NUM     PIC S9(9) USAGE IS BINARY.
     02 ORDER-DATE DATE.
         03 YEAR      PIC S9(4) USAGE IS BINARY.
         03 MONTH     PIC S9(4) USAGE IS BINARY.
         03 DAY       PIC S9(4) USAGE IS BINARY.
     02 ORDER-TEXT    PIC X(30).
     02 ORDER-STATUS  PIC S9(9) USAGE IS BINARY.
```

```
*** INDICATOR VARIABLES FOR ORDERS.
*
 01 IND-ORDERS.
     02 IND-ORDER-NUM       PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-DATE      PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-TEXT      PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-STATUS    PIC S9(4) USAGE IS BINARY.
***
*
     EXEC SQL END DECLARE SECTION END-EXEC
*============================
 PROCEDURE DIVISION.
 CONTROL SECTION.
 S-0.
*** ERROR HANDLING.
*
     EXEC SQL
         WHENEVER SQLERROR GOTO POSTPROCESSING
     END-EXEC
 S-1.
     PERFORM START.
 S-2.
     PERFORM QUERY.
 S-3.
     PERFORM END.
 S-9.
***********************END OF PROGRAM**************************
     STOP RUN.
*============================
 START SECTION.
*============================
 A-1.
     DISPLAY "** STARTING PROGRAM **" UPON DSG.
```

```
 A-9.
     EXIT.
*============================
 QUERY SECTION.
*============================
 F-1.
     DISPLAY "ENTER THE NAME OF THE COMPANY" UPON DSG.
     DISPLAY "WHOSE ORDERS YOU WISH TO DISPLAY." UPON DSG.
     ACCEPT COMPANY-NAME FROM DSG.
*** END IF CUSTOMER NOT FOUND
*
     EXEC SQL
         WHENEVER NOT FOUND GOTO CUST-NUM-END
     END-EXEC
*** SELECT ROW FROM "CUSTOMERS" TABLE.
*
     EXEC SQL
         SELECT CUST_NUM
             INTO :CUSTOMER-NUM
             FROM CUSTOMERS
             WHERE COMPANY = :COMPANY-NAME
     END-EXEC
     DISPLAY "CUSTOMER'S ORDERS ", COMPANY-NAME UPON DSG.
*** DEFINE CURSOR FOR THE SPECIFIED CUSTOMER'S ORDERS.
*
     EXEC SQL
         DECLARE CUR_ORDERS CURSOR FOR
             SELECT ORDER_NUM, ORDER_DATE, ORDER_TEXT, ORDER_STAT
                 FROM ORDERS
                 WHERE CUST_NUM = :CUSTOMER-NUM
     END-EXEC
*** OPEN CURSOR.
*
     EXEC SQL
         OPEN CUR_ORDERS
     END-EXEC.
*** END LOOP WHEN END OF TABLE IS REACHED.
*
```

```
     EXEC SQL
         WHENEVER NOT FOUND GOTO TABLE-END
     END-EXEC

*** POSITION CURSOR ON FIRST OR NEXT ROW.
*

 F-2.

     EXEC SQL
         FETCH CUR_ORDERS
             INTO :ORDER-NUM     INDICATOR :IND-ORDER-NUM,
                  :ORDER-DATE    INDICATOR :IND-ORDER-DATE,
                  :ORDER-TEXT    INDICATOR :IND-ORDER-TEXT,
                  :ORDER-STATUS  INDICATOR :IND-ORDER-STATUS
     END-EXEC.

 F-3.

MOVE ORDER-NUM OF ORDERS TO CA-ORDER-NUM
           IF IND-ORDER-DATE = -1
               THEN
                   MOVE -1 TO CA-ORDER-DATE-Y
                   MOVE 0 TO CA-ORDER-DATE-M
                   MOVE 0 TO CA-ORDER-DATE-D
               ELSE
                   MOVE YEAR OF ORDER-DATE TO CA-ORDER-DATE-Y
                   MOVE MONTH OF ORDER-DATE TO CA-ORDER-DATE-M
                   MOVE DAY OF ORDER-DATE TO CA-ORDER-DATE-D
               END-IF
           IF IND-ORDER-TEXT = -1
               THEN
                   MOVE "NULL" TO CA-ORDER-TEXT
               ELSE
                   MOVE ORDER-TEXT TO CA-ORDER-TEXT
               END-IF
           MOVE ORDER-STATUS TO CA-ORDER-STATUS
           DISPLAY CA-OUTPUT UPON DSG
           GO TO F-2.
   TABLE-END.

*** END OF TABLE REACHED
*

      EXEC SQL
           WHENEVER NOT FOUND CONTINUE
      END-EXEC
      DISPLAY "END OF TABLE REACHED" UPON DSG
```

```
*** CLOSE CURSOR.
*
     EXEC SQL
         CLOSE CUR_ORDERS
     END-EXEC.
     GO TO F-9.
 CUST-NUM-END.
*** CUSTOMER NOT FOUND
*
     EXEC SQL
        WHENEVER NOT FOUND CONTINUE
     END-EXEC
     DISPLAY "CUSTOMER ", COMPANY-NAME, " NOT FOUND" UPON DSG.
 F-9.
     EXIT.
*============================
 END SECTION.
*============================
 E-1.
*** END TRANSACTION.
*
     EXEC SQL
         COMMIT WORK
     END-EXEC.
```

```
 E-2.

     DISPLAY "** PROGRAM COMPLETED CORRECTLY **" UPON DSG.

 E-9.

     EXIT.

*============================

 POSTPROCESSING SECTION.

*============================

 N-1.

     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC

     MOVE SQLSTATE TO O-SQLSTATE
     MOVE SQLLINE TO O-SQLLINE
     DISPLAY OUTPUT UPON DSG

*** ROLL BACK TRANSACTION.
*
     EXEC SQL
         ROLLBACK WORK
     END-EXEC.

 N-2.

     DISPLAY "** THE PROGRAM COULD NOT COMPLETE
-            "CORRECTLY **" UPON DSG
     STOP RUN.

 N-9.

     EXIT.
```

8.2 The program UPDATE

The program UPDATE modifies the elements that make up a color in the COLOR_TAB table. The default database and the default schema are specified with the precompiler option SOURCE-PROPERTIES (CATALOG = SAMPLE, SCHEMA=ORDER_ADMIN).

```
*============================

 IDENTIFICATION DIVISION.

 PROGRAM-ID.    UPDATE.

*============================

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SPECIAL-NAMES.

     TERMINAL IS DSG.

 INPUT-OUTPUT SECTION.

*============================

 DATA DIVISION.

*============================

 WORKING-STORAGE SECTION.

*** OUTPUT VARIABLES FOR ERROR OUTPUT.
*

 01 OUTPUT.
     02 FILLER            PIC X(7) VALUE "ERROR ".
     02 O-SQLSTATE         PIC X(5).
     02 FILLER            PIC X(10) VALUE " IN LINE ".
     02 O-SQLLINE          PIC Z(8)9.

*** OUTPUT VARIABLES.
*

 01 C-OUTPUT.
     02 C-RED        PIC Z9.99.
     02 C-GREEN      PIC Z9.99.
     02 C-BLUE       PIC Z9.99.

*** INSERT COMMUNICATION AREA.
*
```

```
     EXEC SQL
         INCLUDE SQLCA
     END-EXEC

*** HOST VARIABLES.
*
     EXEC SQL BEGIN DECLARE SECTION END-EXEC

*** SQLSTATE.
*
 01 SQLSTATE     PIC X(5).

*** VECTOR FOR RGB COMPONENTS OF A COLOR.
*
 01 COLOR.
     02 COLOR-COMP  PIC SV99 OCCURS 3 TIMES.

     EXEC SQL END DECLARE SECTION END-EXEC

*============================

 PROCEDURE DIVISION.

 CONTROL SECTION.

 S-0.

*** ERROR HANDLING.
*
     EXEC SQL
         WHENEVER SQLERROR GOTO POSTPROCESSING
     END-EXEC

 S-1.

     PERFORM START.

 S-2.

     PERFORM UPDATE.

 S-3.

     PERFORM END.

 S-9.

**********************END OF PROGRAM*************************

     STOP RUN.
```

```
*============================

 START SECTION.

*============================

 A-1.

     DISPLAY "** STARTING PROGRAM **" UPON DSG.

 A-9.

     EXIT.

*============================

 UPDATE SECTION.

*============================

 U-1.

     EXEC SQL
         SELECT RGB(1..3) INTO :COLOR-COMP(1..3)
             FROM COLOR_TAB
             WHERE COLOR-NAME = 'skyblue'
     END-EXEC

     MOVE COLOR-COMP(1) TO C-RED
     MOVE COLOR-COMP(2) TO C-GREEN
     MOVE COLOR-COMP(3) TO C-BLUE

     DISPLAY "Color components for skyblue: ", C-OUTPUT
         UPON DSG.

 U-2.

     MOVE 0.1 TO COLOR-COMP(1)
     MOVE 0.2 TO COLOR-COMP(2)

*** UPDATE A ROW IN THE "COLOR_TAB" TABLE.
*
     EXEC SQL
         UPDATE COLOR_TAB
             SET RGB(1..2) = :COLOR-COMP(1..2)
             WHERE COLOR_NAME = 'skyblue'
     END-EXEC.

 U-3.

     EXEC SQL
         SELECT RGB(1..3) INTO :COLOR-COMP(1..3)
             FROM COLOR_TAB
             WHERE COLOR_NAME = 'skyblue'
     END-EXEC
```

```
     MOVE COLOR-COMP(1) TO C-RED
     MOVE COLOR-COMP(2) TO C-GREEN
     MOVE COLOR-COMP(3) TO C-BLUE

     DISPLAY "Color components for skyblue: ", C-OUTPUT
         UPON DSG.

 U-9.

     EXIT.

*============================

 END SECTION.

*============================

 E-1.

*** END TRANSACTION.
*
     EXEC SQL
         COMMIT WORK
     END-EXEC.

 E-2.

     DISPLAY "** PROGRAM COMPLETED CORRECTLY **" UPON DSG.

 E-9.

     EXIT.

*============================

 POSTPROCESSING SECTION.

*============================

 N-1.

     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC

     MOVE SQLSTATE TO O-SQLSTATE
     MOVE SQLLINE TO O-SQLLINE
     DISPLAY OUTPUT UPON DSG

*** ROLL BACK TRANSACTION.
*
     EXEC SQL
         ROLLBACK WORK
     END-EXEC.
```

```
 N-2.
      DISPLAY "** THE PROGRAM COULD NOT COMPLETE
-             "CORRECTLY **" UPON DSG
      STOP RUN.
 N-9.
      EXIT.
```

8.3 The program INSERT

The program INSERT inserts a row in the ORDERS table. The default database and the default schema are specified with the precompiler option SOURCE-PROPERTIES (CATALOG = SAMPLE, SCHEMA=ORDER_ADMIN).

```
*============================

 IDENTIFICATION DIVISION.

 PROGRAM-ID.    INSERT.

*============================

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SPECIAL-NAMES.

     TERMINAL IS DSG.

 INPUT-OUTPUT SECTION.

*============================

 DATA DIVISION.

*============================

 WORKING-STORAGE SECTION.

*** OUTPUT VARIABLES FOR ERROR OUTPUT.
*

 01 OUTPUT.
     02 FILLER            PIC X(7) VALUE "ERROR ".
     02 O-SQLSTATE        PIC X(5).
     02 FILLER            PIC X(10) VALUE " IN LINE ".
     02 O-SQLLINE         PIC Z(8)9.

*** INSERT COMMUNICATION AREA.
*

     EXEC SQL
         INCLUDE SQLCA
     END-EXEC

***  HOST VARIABLES.
*

     EXEC SQL BEGIN DECLARE SECTION END-EXEC
```

```
*** SQLSTATE.
*
 01 SQLSTATE     PIC X(5).
*** HOST VARIABLES FOR ORDERS.
*
 01 ORDERS.
     02 ORDER-NUM        PIC S9(9) USAGE IS BINARY.
     02 CUST-NUM         PIC S9(9) USAGE IS BINARY.
     02 CONTACT-NUM      PIC S9(9) USAGE IS BINARY.
     02 ORDER-TEXT       PIC X(30).
     02 TARGET DATE.
         03 YEAR         PIC S9(4) USAGE IS BINARY.
         03 MONTH        PIC S9(4) USAGE IS BINARY.
         03 DAY          PIC S9(4) USAGE IS BINARY.
     02 ORDER-STATUS     PIC S9(9) USAGE IS BINARY.
***
     EXEC SQL END DECLARE SECTION END-EXEC
*============================
 PROCEDURE DIVISION.
 CONTROL SECTION.
 S-0.
*** ERROR HANDLING.
*
     EXEC SQL
         WHENEVER SQLERROR GOTO POSTPROCESSING
     END-EXEC
 S-1.
     PERFORM START.
 S-2.
     PERFORM INSERT.
 S-3.
     PERFORM END.
 S-9.
**********************END OF PROGRAM*************************
     STOP RUN.
```

```
*============================

 START SECTION.

*============================

 A-1.

     DISPLAY "** STARTING PROGRAM **" UPON DSG.

 A-9.

     EXIT.

*============================

 INSERT SECTION.

*============================

 I-1.

     MOVE 345 TO ORDER-NUM
     MOVE 101 TO CUST-NUM
     MOVE 20 TO CONTACT-NUM
     MOVE "Network installation" TO ORDER-TEXT
     MOVE 1991 TO YEAR OF TARGET
     MOVE 6 TO MONTH OF TARGET
     MOVE 20 TO DAY OF TARGET
     MOVE 1 TO ORDER-STATUS

*** INSERT ROW IN THE "ORDERS" TABLE.
*
     EXEC SQL
         INSERT INTO ORDERS (ORDER_NUM, CUST_NUM, CONTACT_NUM,
                             ORDER_TEXT, TARGET, ORDER_STAT)
             VALUES (:ORDER-NUM, :CUST-NUM, :CONTACT-NUM,
                      :ORDER-TEXT, :TARGET, :ORDER-STATUS)
     END-EXEC.

 I-9.

     EXIT.

*============================

 END SECTION.

*============================

 E-1.

*** END TRANSACTION.
*
```

```
     EXEC SQL
         COMMIT WORK
     END-EXEC.

 E-2.

     DISPLAY "** THE PROGRAM COMPLETED CORRECTLY **" UPON DSG.

 E-9.

     EXIT.

*==============================

 POSTPROCESSING SECTION.
*==============================

 N-1.

     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC
     MOVE SQLSTATE TO O-SQLSTATE
     MOVE SQLLINE TO O-SQLLINE
     DISPLAY OUTPUT UPON DSG

*** ROLL BACK TRANSACTION.
*

     EXEC SQL
         ROLLBACK WORK
     END-EXEC.

 N-2.

     DISPLAY "** THE PROGRAM COULD NOT COMPLETE
-            "CORRECTLY **" UPON DSG

     STOP RUN.

 N-9.

     EXIT.
```

8.4 The program DELETE

The program DELETE deletes all orders that have already been stored (orders with an order status of 5). All information on the service is deleted from the SERVICE table. The order is then deleted from the ORDERS table. The default database and the default schema are specified with the precompiler option SOURCE-PROPERTIES (CATALOG = SAMPLE, SCHEMA=ORDER_ADMIN).

```
*============================

 IDENTIFICATION DIVISION.

 PROGRAM-ID.    DELETE.

*============================

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SPECIAL-NAMES.

     TERMINAL IS DSG.

 INPUT-OUTPUT SECTION.

*============================

 DATA DIVISION.

*============================

 WORKING-STORAGE SECTION.

*** OUTPUT VARIABLES FOR ERROR OUTPUT.
*

 01 OUTPUT.
     02 FILLER            PIC X(7) VALUE "ERROR ".
     02 O-SQLSTATE        PIC X(5).
     02 FILLER            PIC X(10) VALUE " IN LINE ".
     02 O-SQLLINE         PIC Z(8)9.

*** VARIABLE USED TO OUTPUT THE ORDER NUMBER.
*

 01 O-ORDER-NUM       PIC Z(8)9.

*** INSERT COMMUNICATION AREA.
*

     EXEC SQL
         INCLUDE SQLCA
     END-EXEC
```

```
*** HOST VARIABLES.
*
     EXEC SQL BEGIN DECLARE SECTION END-EXEC
*** SQLSTATE.
*
 01 SQLSTATE     PIC X(5).
*** HOST VARIABLES FOR ORDERS.
*
 01 ORDERS.
     02 ORDER-NUM        PIC S9(9) USAGE IS BINARY.
     02 ORDER-TEXT       PIC X(30).
*** INDICATOR VARIABLES FOR ORDERS.
*
 01 IND-ORDERS.
     02 IND-ORDER-NUM        PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-TEXT       PIC S9(4) USAGE IS BINARY.
***
     EXEC SQL END DECLARE SECTION END-EXEC
*==============================
 PROCEDURE DIVISION.
 CONTROL SECTION.
 S-0.
*** ERROR HANDLING.
*
     EXEC SQL
         WHENEVER SQLERROR GOTO POSTPROCESSING
     END-EXEC
 S-1.
     PERFORM START.
 S-2.
     PERFORM DELETE.
 S-3.
     PERFORM END.
 S-9.
```

```
**********************END OF PROGRAM*************************
     STOP RUN.
*============================
 START SECTION.
*============================
 A-1.
     DISPLAY "** STARTING PROGRAM **" UPON DSG.
 A-9.
     EXIT.
*============================
 DELETE SECTION.
*============================
 D-1.
*** DEFINE CURSOR FOR THE ORDERS
*** ALREADY STORED (ORDER-STATUS=5).
*
     EXEC SQL
         DECLARE CUR_ORDERS CURSOR FOR
             SELECT ORDER_NUM, ORDER_TEXT
                 FROM ORDERS
                 WHERE ORDER_STAT = 5
     END-EXEC
*** OPEN CURSOR.
*
     EXEC SQL
         OPEN CUR_ORDERS
     END-EXEC.
*** END LOOP IF END OF TABLE IS REACHED.
*
     EXEC SQL
         WHENEVER NOT FOUND GOTO TABLE-END
     END-EXEC.
```

```
*** POSITION CURSOR ON FIRST OR NEXT ROW.
*
 D-2.
     EXEC SQL
         FETCH CUR_ORDERS
             INTO :ORDER-NUM   INDICATOR :IND-ORDER-NUM,
                  :ORDER-TEXT  INDICATOR :IND-ORDER-TEXT
     END-EXEC.
 D-3.
     MOVE ORDER-NUM TO O-ORDER-NUM
     DISPLAY "DELETE ORDER ", O-ORDER-NUM, ": ", ORDER-TEXT
         UPON DSG
*** DELETE ALL THE INFORMATION ON THE SERVICE CONTAINED
*** IN THE SERVICE TABLE.
*
     EXEC SQL
         DELETE FROM SERVICE
             WHERE ORDER_NUM = :ORDER-NUM
     END-EXEC
*** DELETE THE ORDER FROM THE ORDERS TABLE.
*
     EXEC SQL
         DELETE FROM ORDERS
             WHERE ORDER_NUM = :ORDER-NUM
     END-EXEC
     GO TO D-2.
 TABLE-END.
     DISPLAY "ALL ORDERS ALREADY " UPON DSG.
     DISPLAY "STORED HAVE " UPON DSG.
     DISPLAY "BEEN DELETED" UPON DSG.
*** CLOSE CURSOR.
*
     EXEC SQL
         CLOSE CUR_ORDERS
     END-EXEC.
 D-9.
     EXIT.
```

```
*============================

 END SECTION.

*============================

 E-1.
*** END TRANSACTION.
*
     EXEC SQL
         COMMIT WORK
     END-EXEC.

 E-2.

     DISPLAY "** PROGRAM COMPLETED CORRECTLY **" UPON DSG.

 E-9.

     EXIT.

*============================

 POSTPROCESSING SECTION.

*============================

 N-1.

     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC
     MOVE SQLSTATE TO O-SQLSTATE
     MOVE SQLLINE TO O-SQLLINE
     DISPLAY OUTPUT UPON DSG

*** ROLL BACK TRANSACTION.
*
     EXEC SQL
         ROLLBACK WORK
     END-EXEC.

 N-2.
     DISPLAY "** THE PROGRAM COULD NOT COMPLETE
-            "CORRECTLY **" UPON DSG

     STOP RUN.

 N-9.

     EXIT.
```

8.5 The program DYNAMIC

The program DYNAMIC demonstrates the use of dynamic SQL statements. It outputs data on an order from the SERVICE table. When the program executes, users can specify an order number and a search condition for the data they wish to view. The program defines a dynamic cursor. The formulation entered is converted into a cursor description. The cursor description is translated dynamically and the assigned cursor is opened. Users can then enter a dynamic SQL statement in order to update, delete or insert data in the SERVICE table. The dynamic SQL statement is prepared and executed in a single step. The default database and the default schema are specified with the precompiler option SOURCE-PROPERTIES (CATALOG = SAMPLE, SCHEMA=ORDER_ADMIN).

```
*============================

 IDENTIFICATION DIVISION.

 PROGRAM-ID.    DYNAMIC.

*============================

 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SPECIAL-NAMES.

     TERMINAL IS DSG.

 INPUT-OUTPUT SECTION.

*============================

 DATA DIVISION.

*============================

 WORKING-STORAGE SECTION.

*** OUTPUT VARIABLES FOR ERROR OUTPUT.
*

 01 OUTPUT.
     02 FILLER            PIC X(7) VALUE "ERROR ".
     02 O-SQLSTATE        PIC X(5).
     02 FILLER            PIC X(10) VALUE " IN LINE ".
     02 O-SQLLINE         PIC Z(8)9.
```

```
*** VARIABLE USED TO OUTPUT A RECORD IN THE SERVICE TABLE.
*
 01 S-OUTPUT.
     02 SO-SERVICE-NUM     PIC ZZZZ9.
     02 SO-ORDER-NUM       PIC ZZZZ9.
     02 SO-SERVICE-DATE.
         03 YEAR           PIC Z9999.
         03 FILLER         PIC X      VALUE "-".
         03 MONTH          PIC 99.
         03 FILLER         PIC X      VALUE "-".
         03 DAY            PIC 99.
         03 FILLER         PIC X      VALUE " ".
     02 SO-SERVICE-TEXT    PIC X(25).
     02 FILLER             PIC X      VALUE " ".
     02 SO-SERVICE-UNIT    PIC X(10).
     02 SO-SERVICE-TOTAL   PIC ZZ9.99.
     02 SO-SERVICE-PRICE   PIC ZZZZ9.
     02 SO-VAT             PIC Z9.99.
     02 SO-INV-NUM         PIC ZZZ9.
*** FIELD USED TO STORE THE ORDER NUMBER ENTERED
*
 01 ORDER-NUM-ENTRY   PIC Z(9).
*** FIELD USED TO STORE CONDITION FOR SEARCH
*
 01 CONDITION    PIC X(200).
*** INSERT COMMUNICATION AREA.
*
     EXEC SQL
         INCLUDE SQLCA
     END-EXEC
*** HOST VARIABLES.
*
     EXEC SQL BEGIN DECLARE SECTION END-EXEC
*** SQLSTATE.
*
 01 SQLSTATE     PIC X(5).
*** FIELD USED TO STORE THE ORDER NUMBER ENTERED.
*
 01 ORDER-NUMBER   PIC S9(9) USAGE IS BINARY.
```

```
*** FIELD USED TO STORE CURSOR DESCRIPTION.
*
 01 DESCRIPTION PIC X(200).
*** FIELD USED TO STORE THE DYNAMIC STATEMENT.
*
 01 STATEMENT    PIC X(200).
*** HOST VARIABLES FOR SERVICE.
*
 01 SERVICE.
     02 SERVICE-NUM         PIC S9(9) USAGE IS BINARY.
     02 ORDER-NUM           PIC S9(9) USAGE IS BINARY.
     02 SERVICE-DATE DATE.
         03 YEAR            PIC S9(4) USAGE IS BINARY.
         03 MONTH           PIC S9(4) USAGE IS BINARY.
         03 DAY             PIC S9(4) USAGE IS BINARY.
     02 SERVICE-TEXT        PIC X(25).
     02 SERVICE-UNIT        PIC X(10).
     02 SERVICE-TOTAL       PIC S999V99.
     02 SERVICE-PRICE       PIC S9(4).
     02 VAT                 PIC SV99.
     02 INV-NUM             PIC S9(4).
*** INDICATOR VARIABLES FOR SERVICE.
*
 01 IND-SERVICE.
     02 IND-SERVICE-NUM     PIC S9(4) USAGE IS BINARY.
     02 IND-ORDER-NUM       PIC S9(4) USAGE IS BINARY.
     02 IND-SERVICE-DATE    PIC S9(4) USAGE IS BINARY.
     02 IND-SERVICE-TEXT    PIC S9(4) USAGE IS BINARY.
     02 IND-SERVICE-UNIT    PIC S9(4) USAGE IS BINARY.
     02 IND-SERVICE-TOTAL   PIC S9(4) USAGE IS BINARY.
     02 IND-SERVICE-PRICE   PIC S9(4) USAGE IS BINARY.
     02 IND-VAT             PIC S9(4) USAGE IS BINARY.
     02 IND-INV-NUM         PIC S9(4) USAGE IS BINARY.
***
     EXEC SQL END DECLARE SECTION END-EXEC
```

```
============================

 PROCEDURE DIVISION.

 CONTROL SECTION.

 S-1.

*** ERROR HANDLING.
*
     EXEC SQL
         WHENEVER SQLERROR GOTO POSTPROCESSING
     END-EXEC

 S-2.

     PERFORM START.

 S-3.

     PERFORM CURSOR-ACCESS.

 S-4.

     PERFORM DYNAMIC.

 S-5.

     PERFORM END.

 S-9.

**********************END OF PROGRAM*************************

     STOP RUN.

*============================

 START SECTION.

*============================

 A-1.

     DISPLAY "** STARTING PROGRAM **" UPON DSG.

 A-9.

     EXIT.

*============================

 CURSOR-ACCESS SECTION.

*============================

 C-1.

     DISPLAY "ENTER THE ORDER NUMBER (NINE DIGITS)." UPON DSG.

     ACCEPT ORDER-NUM-ENTRY FROM DSG

***
```

```
     DISPLAY "ENTER THE CONDITION THAT IS " UPON DSG.
     DISPLAY "TO APPLY TO THE SERVICES TO BE LISTED."
         UPON DSG

     ACCEPT CONDITION FROM DSG.

 C-2.

*** CREATE CURSOR DESCRIPTION.
*

     INITIALIZE DESCRIPTION
     STRING "SELECT * FROM SERVICE WHERE ORDER_NUM = ? AND " CONDITION
         DELIMITED BY SIZE INTO DESCRIPTION

*** PREPARE CURSOR DESCRIPTION.
*

     EXEC SQL
         PREPARE CUR_DESCRIPTION FROM :DESCRIPTION
     END-EXEC.

 C-3.

*** DEFINE CURSOR FOR THE SERVICES ASSOCIATED WITH
*** THE SPECIFIED ORDER.
*

     EXEC SQL
         DECLARE CUR_SERVICE CURSOR FOR CUR_DESCRIPTION
     END-EXEC

*** OPEN CURSOR.
*** THE PLACEHOLDER IN THE CURSOR DESCRIPTION IS SET
*** TO THE SELECTED ORDER NUMBER.
*

     MOVE ORDER-NUM-ENTRY TO ORDER-NUMBER
     EXEC SQL
         OPEN CUR_SERVICE USING :ORDER-NUMBER
     END-EXEC.

*** END LOOP IF THE END OF THE TABLE IS REACHED.
*

     EXEC SQL
         WHENEVER NOT FOUND GOTO TABLE-END
     END-EXEC

*** POSITION CURSOR ON FIRST/NEXT RECORD.
*
```

```
 C-4.

     EXEC SQL
         FETCH CUR_SERVICE
             INTO :SERVICE-NUM    INDICATOR :IND-SERVICE-NUM,
                  :ORDER-NUM      INDICATOR :IND-ORDER-NUM,
                  :SERVICE-DATE   INDICATOR :IND-SERVICE-DATE,
                  :SERVICE-TEXT   INDICATOR :IND-SERVICE-TEXT,
                  :SERVICE-UNIT   INDICATOR :IND-SERVICE-UNIT,
                  :SERVICE-TOTAL  INDICATOR :IND-SERVICE-TOTAL,
                  :SERVICE-PRICE  INDICATOR :IND-SERVICE-PRICE,
                  :VAT            INDICATOR :IND-VAT,
                  :INV-NUM        INDICATOR :IND-INV-NUM
     END-EXEC

     MOVE SERVICE-NUM TO SO-SERVICE-NUM
     MOVE ORDER-NUM TO SO-ORDER-NUM
     MOVE YEAR OF SERVICE-DATE TO YEAR OF SO-SERVICE-DATE
     MOVE MONTH OF SERVICE-DATE TO MONTH OF SO-SERVICE-DATE
     MOVE DAY OF SERVICE-DATE TO DAY OF SO-SERVICE-DATE
     MOVE SERVICE-TEXT TO SO-SERVICE-TEXT
     MOVE SERVICE-UNIT TO SO-SERVICE-UNIT
     MOVE SERVICE-TOTAL TO SO-SERVICE-TOTAL
     MOVE SERVICE-PRICE TO SO-SERVICE-PRICE
     MOVE VAT TO SO-VAT
     MOVE INV-NUM TO SO-INV-NUM
     DISPLAY S-OUTPUT UPON DSG
     GO TO C-4.

 TABLE-END.

*** END OF TABLE REACHED
*
     EXEC SQL
               WHENEVER NOT FOUND CONTINUE
     END-EXEC
     DISPLAY "END OF TABLE REACHED" UPON DSG

*** CLOSE CURSOR.
*
     EXEC SQL
         CLOSE CUR_SERVICE
     END-EXEC.
```

```
 C-9.

     EXIT.

*============================

 DYNAMIC SECTION.

*============================

 D-1.

     DISPLAY "ENTER AN SQL STATEMENT" UPON DSG
     DISPLAY "(INSERT, UPDATE OR DELETE). CANCEL WITH '*'"
         UPON DSG
     INITIALIZE STATEMENT
     ACCEPT STATEMENT FROM DSG.

 D-2.

     IF STATEMENT NOT = "*"
         THEN

*** PREPARE AND EXECUTE DYNAMIC STATEMENT
*** IN A SINGLE STEP.
*
            EXEC SQL
                EXECUTE IMMEDIATE :STATEMENT
            END-EXEC

          DISPLAY "STATEMENT EXECUTED" UPON DSG
     END-IF.

 D-9.

     EXIT.

*============================

 END SECTION.

*============================

 E-1.

*** END TRANSACTION.
*
     EXEC SQL
         COMMIT WORK
     END-EXEC.

 E-2.

     DISPLAY "** PROGRAM COMPLETED SUCCESSFULLY **" UPON DSG.
```

```
 E-9.

     EXIT.

*============================

 POSTPROCESSING SECTION.

*============================

 N-1.

     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC
     MOVE SQLSTATE TO O-SQLSTATE
     MOVE SQLLINE TO O-SQLLINE
     DISPLAY OUTPUT UPON DSG

*** ROLL BACK TRANSACTION.
*
     EXEC SQL
         ROLLBACK WORK
     END-EXEC.

 N-2.

     DISPLAY "** THE PROGRAM COULD NOT COMPLETE
-            "CORRECTLY **" UPON DSG
     STOP RUN.

 N-9.

     EXIT.
```

9 Messages output by the ESQL-COBOL system

IQB0001 ESQL-COBOL V2.0A00

IQB0101 (&00) INSUFFICIENT MEMORY (REASON (&01))

IQB0201 (&00) OPEN FAILURE ON FILE <(&01)> (REASON (&02))

IQB0202 (&00) CLOSE FAILURE ON FILE <(&01)> (REASON (&02))

IQB0203 (&00) WRITE FAILURE ON FILE <(&01)> (REASON (&02))

IQB0204 (&00) READ FAILURE ON FILE <(&01)> (REASON (&02))

IQB0205 (&00) NO LIBRARY PATH DECLARED TO INCLUDE <(&01)>

IQB0206 (&00) LIBRARY PATH FAILURE ON INCLUDE <(&01)> (REASON (&02))

IQB0207 (&00) NESTING LIMIT EXCEEDED ON INCLUDE OF <(&01)>

IQB0208 (&00) INPUT FILE <(&01)> RECORD SIZE EXCEEDS NORM, TRUNCATED

IQB0209 (&00) OUTPUT FILE <(&01)> RECORD SIZE <(&02)> EXCEEDS NORM, TRUNCATED

IQB0220 (&00) UNRECOVERABLE I/O ERROR OCCURRED

IQB0301 (&00) SDF: ERROR CODE (0X(&01))

IQB0302 (&00) SDF: ILLEGAL STATEMENT (0X(&01))

IQB0303 (&00) SDF: UNRECOVERABLE SYSTEM ERROR

IQB0304 (&00) SDF: OPERAND ERROR DETECTED

IQB0305 (&00) SDF: TRANSFER AREA IS TOO SMALL

IQB0306 (&00) SDF: INCORRECT STATEMENT PROCESSED

IQB0307 (&00) SDF: INCORRECT STATEMENT (END/STEP PROCESSED)

IQB0308 (&00) SDF: ERRONEOUS DEFAULT VALUES IN STATEMENT SYNTAX

IQB0309 (&00) SDF: DIALOG ERROR CORRECTION NOT POSSIBLE

IQB0310 (&00) SDF: LOGGING AREA TOO SMALL

IQB0311 (&00) SDF: UNEXPECTED END-STATEMENT FOUND

IQB0312 (&00) SDF: DIALOG ERROR CORRECTION REFUSED

IQB0313 (&00) SDF: NOT AVAILABLE FOR EXECUTION

IQB0314 (&00) SDF: PROGRAM NOT KNOWN IN SYNTAX FILE

IQB0315 (&00) SDF: NOT LOADED IN MEMORY

IQB0316 (&00) SDF: STATEMENT NOT KNOWN OR WRONG SYNTAX FILE IN USE

IQB0317 (&00) TOO MANY PRECOMPILE STATEMENTS ((&01)) READ FROM SYSSTMT

IQB0318 (&00) *SUBSTITUTE IDENTIFIER-TOKEN <(&01)> DUPLICATE

IQB0319 (&00) *SUBSTITUTE REPLACEMENT-TOKEN <(&01)> FOR NULL IDENTIFIER-TOKEN

IQB0320 (&00) ESQL-DIALECT=ISO AND QUOTATION-CHARACTER=HOST-STD CONFLICT

IQB0321 (&00) CATALOG AND SCHEMA OPTIONS MUST BE EXPLICITLY ASSIGNED

IQB0330 (&00) SQLOPT (V1.0B) STATEMENTS FOUND IN ESQL-HOST-PROGRAM

IQB0350 (&00) PREMATURE END OF ESQL-HOST-PROGRAM INPUT FILE

IQB0351 (&00) CONFLICTING OR UNORDERED OPTIONS IN ESQL-HOST-PROGRAM

IQB0352 (&00) //PRECOMPILE ... OPTIONS EXCEED SDF TRANSLATION-AREA SIZE

IQB0357 (&00) CONTINUATION LINE EXPECTED IN ESQL-HOST-PROGRAM LINE (&01)

IQB0358 (&00) //PRECOMPILE NOT TERMINATED BY //END IN ESQL-HOST-PROGRAM

IQB0359 (&00) INCORRECT //PRECOMPILE ... OPTIONS IN ESQL-HOST-PROGRAM

IQB0360 (&00) COBRUN/COMOPT NOT TERMINATED BY END IN ESQL-HOST-PROGRAM

IQB0361 (&00) //COMPILE ... NOT TERMINATED BY //END IN ESQL-HOST-PROGRAM

IQB0362 (&00) UNEXPECTED SQLOPT OPTION IN ESQL-HOST-PROGRAM LINE (&02): (&01)

IQB0370 (&00) REPLACE-BY-FILE SYNONYM-INPUT-FILE IS EMPTY

IQB0371 (&00) //PRECOMPILE ... STATEMENT NOT FOUND IN REPLACE-BY-FILE INPUT

IQB0372 (&00) //PRECOMPILE ... IN REPLACE-BY-FILE INPUT NOT TERMINATED BY //END

IQB0373 (&00) INCONSISTENCIES IN SDF-SYNTAX-FILE (REPLACE-BY-FILE)

IQB0374 (&00) UNRECOGNIZABLE DATA IN INPUT (REPLACE-BY-FILE)

IQB0375 (&00) CONTINUATION LINE EXPECTED IN REPLACE-BY-FILE LINE (&01)

IQB0376 (&00) INCORRECT //PRECOMPILE ... OPTIONS IN INPUT (REPLACE-BY-FILE)

IQB0501 (&00) WRITE FAILURE ON OBJECT MODULE LIBRARY (INSUFFICIENT MEMORY)

IQB0502 (&00) WRITE FAILURE ON OBJECT MODULE LIBRARY (REASON (&01))

IQB0601 (&00) SESMOD/SESLINK NOT FOUND DURING DYNAMIC LOAD

IQB0701 (&00) ERROR LIMIT REACHED BEFORE ALL ERRORS COULD BE FLAGGED

IQB0900 (&00) INTERNAL ERROR: (&01) WHILST OUTPUTTING OBJECT MODULE

IQB0901 (&00) (&01)

IQB0903 (&00) INTERNAL ERROR: <(&01)>. REASON (&02)

IQB0905 (&00) INTERNAL ERROR: <(&01)> (&02)

IQB0907 (&00) INTERNAL ERROR: <(&01)> <(&02)>

IQB0909 (&00) INTERNAL ERROR: (&01)

IQB0910 (&00)(&01)

IQB0921 (&00) INTERNAL ERROR: (&01) (&02)

IQB0931 (&00) INTERNAL ERROR: DC, (&01) TOO LONG OPTION-IDENTIFIER

IQB0932 (&00) INTERNAL ERROR: DC, (&01) SQL-SYSTEM-FAILURE RC=(&02)/EC=(&03)

IQB0933 (&00) INTERNAL ERROR: DC, (&01): UNKNOWN COMBINATION RC=(&02)/EC=(&03)

IQB0934 (&00) INTERNAL ERROR: DC, (&01): UNDEF. REACTION FOR RC=(&02)/EC=(&03)

IQB0935 (&00) INTERNAL ERROR: DC, AT TO ICSQLE CONVERSION FOR (&01)

IQB0936 (&00) INTERNAL ERROR: DC, ICSQLE TO AT CONVERSION

IQB0937 (&00) INTERNAL ERROR: DC, PARAMETER-NAME ICSQLE TO AT CONVERSION

IQB0938 (&00) INTERNAL ERROR: DC, DATATYPE ICSQLE TO AT CONVERSION RESULT (&01)

IQB0939 (&00) INTERNAL ERROR: DC, TOO MANY SUBSEQUENT CALLS TO (&01)

IQB0940 (&00) INTERNAL ERROR: DC, SQLROW OUT OF RANGE <(&01)>

IQB0941 (&00) INTERNAL ERROR: DC, USAGE ICSQLE TO AT CONVERSION RESULT (&01)

IQB0942 (&00) INTERNAL ERROR: DC, CATALOG-NAME ICSQLE TO NT CONVERSION

IQB0943 (&00) INTERNAL ERROR: DC, VARIABLE BUFFER SIZE INCONSISTENCY

IQB0945 (&00) INTERNAL ERROR: HG, LABEL-NAME TOO LONG <(&01)>

IQB0946 (&00) INTERNAL ERROR: HG, NODE <KIND ZERO> DETECTED

IQB0947 (&00) INTERNAL ERROR: HG, IDMARK TOO LONG <(&01)>

IQB0948 (&00) INTERNAL ERROR: HG, ENTRY-NAME TOO LONG <(&01)>

IQB0949 (&00) INTERNAL ERROR: HG, FORGOTTEN MESSAGE FOR LINE <(&01)> FOUND

IQB0950 (&00) INTERNAL ERROR: HG, INCLUDE STACK <(&01)>

IQB0951 (&00) INTERNAL ERROR: HG, UNKNOWN EXCEPTION-CONDITION <(&01)> FROM AT

IQB0952 (&00) INTERNAL ERROR: HG, WRONG STATE <(&01)> IN FUNCTION <(&02)>

IQB0953 (&00) INTERNAL ERROR: HG, MORE THAN ONE COLLISION RANGE FOUND

IQB0955 (&00) INTERNAL ERROR: LC, OPERATION (&01): LINE-NUMBER OUT OF SEQUENCE

IQB0956 (&00) INTERNAL ERROR: LC, OPERATION (&01): RANGE NOT SET

IQB0977 (&00) INTERNAL ERROR: AT, (&01)

```
IQB1001    (&00) INVALID EXCEPTION CONDITION
IQB1002    (&00) <END DECLARE SECTION> EXPECTED
IQB1003    (&00) INVALID TOKEN; CONTINUE, GO TO OR GOTO EXPECTED
IQB1004    (&00) INVALID TOKEN; TO EXPECTED
IQB1005    (&00) INVALID TOKEN; FOUND EXPECTED
IQB1006    (&00) RIGHT-HAND DELIMITER <QUOTE> EXPECTED
IQB1008    (&00) <BEGIN DECLARE SECTION> MISSING
IQB1010    (&00) TOO LONG COBOL DATA-NAME
IQB1011    (&00) LABEL NAME EXPECTED
IQB1012    (&00) INVALID SCALE; (3) EXPECTED
IQB1013    (&00) <INCLUDE> NOT ISO-CONFORM
IQB1014    (&00) INVALID OR MISSING <INCLUDE> SPECIFICATION
IQB1015    (&00) INVALID ESQL STATEMENT
IQB1016    (&00) INVALID OR MISSING COBOL DATA-NAME
IQB1017    (&00) TOO LONG PICTURE STRING
IQB1018    (&00) INVALID OR MISSING DATA-ITEM PICTURE SIZE SPECIFICATION
IQB1019    (&00) PICTURE STRING NOT ISO-CONFORM
IQB1020    (&00) INVALID OR MISSING COBOL PROCEDURE-NAME
IQB1021    (&00) FILE TO BE INCLUDED NOT FOUND
IQB1022    (&00) TOO MANY NESTED FILES FOR INCLUSION
IQB1023    (&00) REPLACEMENT OBJECT LENGTH EXCEEDS LIMIT
IQB1024    (&00) <INCLUDE> WITHIN REPLACEMENT OBJECT NOT ALLOWED
IQB1025    (&00) <WHENEVER> WITHIN REPLACEMENT OBJECT NOT ALLOWED
IQB1026    (&00) INVALID REPLACEMENT OBJECT
IQB1027    (&00) DELIMITED IDENTIFIER NOT ENDED BY <QUOTE> IN REPLACEMENT OBJECT
IQB1028    (&00) IDENTIFIER EXPECTED AFTER < . > IN REPLACEMENT OBJECT
IQB1029    (&00) LEVEL NUMBER NOT SPECIFIED AS 1 OR 2 DIGIT(S)
IQB1030    (&00) PARSER STACK EXAUSTED
IQB1031    (&00) INVALID INDEX OR RANGE
IQB1032    (&00) UNSIGNED INTEGER EXPECTED AFTER < ( >
IQB1033    (&00) INVALID OR MISSING COBOL PROCEDURE-NAME
```

IQB1034 (&00) END-EXEC EXPECTED

IQB1035 (&00) INVALID COBOL DATA-ITEM DEFINITION

IQB1036 (&00) QUALIFICATION OF COBOL IDENTIFIER NOT ISO-CONFORM

IQB1037 (&00) INDEX NOT ISO-CONFORM

IQB1038 (&00) INVALID <INCLUDE>

IQB1039 (&00) LEVEL NUMBER OUT OF RANGE

IQB1040 (&00) INVALID PICTURE STRING

IQB1041 (&00) PERIOD < . > EXPECTED

IQB1042 (&00) INVALID TOKEN; LEADING OR TRAILING EXPECTED

IQB1043 (&00) INVALID OR MISSING USAGE CLAUSE

IQB1044 (&00) INVALID OR MISSING OCCURS CLAUSE

IQB1045 (&00) INVALID VALUE CLAUSE

IQB1046 (&00) INVALID TIMESTAMP SPECIFICATION

IQB1047 (&00) INVALID TIME SPECIFICATION

IQB1048 (&00) LEVEL NUMBER EXPECTED

IQB1049 (&00) INVALID OR MISSING COBOL DATA-NAME

IQB1050 (&00) INVALID OR MISSING PICTURE CLAUSE

IQB1051 (&00) INVALID OR MISSING SIGN CLAUSE

IQB1052 (&00) INVALID OR MISSING LEADING CLAUSE

IQB1053 (&00) INVALID OR MISSING SEPARATE CLAUSE

IQB2001 (&00) SIGN CLAUSE NOT ALLOWED

IQB2002 (&00) ONLY USAGE DISPLAY ALLOWED

IQB2003 (&00) SIGN CLAUSE ONLY ALLOWED WITH USAGE DISPLAY

IQB2004 (&00) NO PICTURE CLAUSE ALLOWED WITH COMP-1 OR COMP-2

IQB2005 (&00) USAGE BINARY OR COMP IN SQL ONLY ALLOWED FOR INTEGERS

IQB2006 (&00) TOO MANY DIGIT POSITIONS FOR USAGE BINARY OR COMP

IQB2007 (&00) INCORRECT LEVEL NUMBER

IQB2008 (&00) INCORRECT LEVEL NUMBER : LEVEL NUMBER OUT OF RANGE

IQB2009 (&00) DUPLICATE PICTURE CLAUSE ILLEGAL

IQB2010 (&00) DUPLICATE SIGN CLAUSE ILLEGAL

IQB2011 (&00) DUPLICATE SYNCHRONIZED CLAUSE ILLEGAL

IQB2012 (&00) DUPLICATE USAGE CLAUSE ILLEGAL

IQB2013 (&00) DUPLICATE EXTERNAL CLAUSE ILLEGAL

IQB2014 (&00) DUPLICATE GLOBAL CLAUSE ILLEGAL

IQB2015 (&00) DUPLICATE OCCURS CLAUSE ILLEGAL

IQB2016 (&00) DUPLICATE VALUE CLAUSE ILLEGAL

IQB2017 (&00) ILLEGAL NESTING OF OCCURS

IQB2018 (&00) OCCURS CLAUSE NOT ALLOWED FOR THIS LEVEL NUMBER

IQB2019 (&00) ILLEGAL ITEM SUBORDINATE TO VARCHAR

IQB2020 (&00) CLAUSE ILLEGAL WITH VARCHAR

IQB2021 (&00) ITEM SUBORDINATE TO DATETIME ILLEGAL

IQB2022 (&00) CLAUSE ILLEGAL WITH DATETIME

IQB2024 (&00) TOO FEW SUBORDINATE COMPONENTS

IQB2025 (&00) VALUE ILLEGAL IN OCCURS CLAUSE

IQB2026 (&00) TOO BIG OCCURS VALUE

IQB2027 (&00) ILLEGAL REDEFINITION OF HOST VARIABLE

IQB2028 (&00) COBOL TYPE NOT ALLOWED

IQB2029 (&00) CLAUSE NOT ALLOWED FOR GROUP ITEMS

IQB2030 (&00) HOST VARIABLE UNDEFINED

IQB2031 (&00) AMBIGUOUS QUALIFICATION

IQB2032 (&00) ILLEGAL GROUP REFERENCE IN HOST IDENTIFIER

IQB2033 (&00) SUBSCRIPT OUT OF RANGE

IQB2034 (&00) SUBSCRIPT ON ITEM NOT SUBORDINATE TO OCCURS

IQB2035 (&00) ILLEGAL SUBSCRIPT RANGE

IQB2036 (&00) TYPE OF HOST VARIABLE NOT ALLOWED IN SQL STATEMENT

IQB2037 (&00) TOO MANY QUALIFICATIONS IN HOST IDENTIFIER

IQB2038 (&00) INDEX OR INDEX RANGE MISSING

IQB2040 (&00) INCORRECT TYPE FOR SQLCODE

IQB2041 (&00) ILLEGAL DECLARATION OF SQLCODE AFTER SQL STATEMENT

IQB2042 (&00) INCORRECT TYPE FOR SQLSTATE

IQB2043 (&00) ILLEGAL DECLARATION OF SQLSTATE AFTER SQL STATEMENT

IQB2044 (&00) SQLCODE MUST NOT BE A GROUP ITEM

```
IQB2045    (&00) SQLCODE MUST NOT CONTAIN, NOR BE SUBORDINATE TO OCCURS
IQB2046    (&00) SQLSTATE MUST NOT BE A GROUP ITEM
IQB2047    (&00) SQLSTATE MUST NOT CONTAIN, NOR BE SUBORDINATE TO OCCURS
IQB2048    (&00) SQLCODE ALREADY DEFINED
IQB2049    (&00) SQLSTATE ALREADY DEFINED
IQB2050    (&00) DATA DESCRIPTION NOT CONFORMING WITH ISO SQL
IQB2051    (&00) HOST VARIABLE NOT CONFORMING WITH ISO SQL
IQB2052    (&00) ILLEGAL ITEM SUBORDINATE TO NVRCHAR
IQB2053    (&00) CLAUSE ILLEGAL WITH NVARCHAR
IQB2054    (&00) ONLY USAGE NATIONAL ALLOWED
IQB3000    (&00) (&01)
IQB3001    (&00) EMBEDDED SQL STATEMENT TOO LONG
IQB3002    (&00) EMBEDDED SQL STATEMENT CONTAINS TOO MANY HOST VARIABLES
IQB9001    (&00)BEGIN (&01)
IQB9002    (&00)COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1994.
IQB9003    (&00)ALL RIGHTS RESERVED.
IQB9004    (&00)MESSAGE STATISTICS: (&01) NOTES, (&02) WARNINGS, (&03) ERRORS
IQB9006    (&00)MODULE NOT GENERATED
IQB9007    (&00)MODULE GENERATED
IQB9008    (&00)END (&01) -- TIME USED = (&02) SECS.
SIS0001    COMP=SISMF,VER=100,DATE=930131,COMPNR=00000000
SIS0003    No previous error occured
SIS0004    Not enough or too many inserts passed
```

Meaning
Missing inserts are substituted by the empty string.
Unnecessary inserts are ignored.

SIS0005 PROSYS messages are not available

SIS0006 Version of message class (&00) is wrong

SIS0007 Version of set (&00) in the specified catalog is wrong

SIS0008 Message (&00)(&01) is not defined

SIS0009 In the specified catalog message (&00),(&01) is not defined

SIS0010 Message catalog (&00) can not be opened

SIS0011 Parameter (&00) is wrong

SIS0012 System specific error occured

SIS0036 PROSOS internal Error

SIS0100 Error during initialisation of SHS

SIS0101 reserved

SIS0102 Insufficient memory

SIS0103 Invalid size parameter -- (&00)

SIS0104 Size (&00) too big

SIS0105 Option requires an argument -- (&00)

SIS0106 Illegal option -- (&00)

SIS0107 Handle does not point to string container

SIS0108 Invalid close mode

SIS0109 String container cannot be opened

SIS0110 Invalid access of a string container

SIS0111 Write access of a string container not allowed

SIS0112 System specific error occured

SIS0200 No more space available

SIS0201 Error during object generation: (&00)

SIS0202 File (&00) cannot be opened

SIS0601 End of file detected

SIS0602 Specified record does not exist

SIS0603 Specified record exists

SIS0604 Begin of file detected

SIS0605 Specified link does not exist

SIS0606 Length of file name exceeds P_MAXFILENAME

SIS0607 Length of path string exceeds P_MAXPATHSTRG

SIS0608 Length of path name exceeds P_MAXPATHNAME

SIS0609 Length of link name exceeds P_MAXLINKNAME

SIS0610 No more space available

SIS0611 Number of path elements exceeds P_MAXHIERARCHY

SIS0612 Function not supported

SIS0613 Missing file name or syntax error in file name

SIS0614 Number of secondary keys exceeds P_MAXKEYS

SIS0615 Number of open files exceeds system specific boundary

SIS0616 Specified file does not exist

SIS0617 Write access not allowed

SIS0618 No file name found

SIS0619 File locked

SIS0620 Illegal combination of file attributes

SIS0621 File handle is invalid

SIS0622 Current record smaller than MINSIZE

SIS0623 Current record bigger than MAXSIZE

SIS0625 No read sequential performed before rewrite sequential

SIS0626 Record format out of range or not allowed

SIS0627 MINSIZE greater than MAXSIZE

SIS0628 Organisation out of range or not allowed

SIS0629 File exists but MUST_NOT_EXIST specified

SIS0630 Specified access function not allowed

SIS0631 Key parameters out of range or not allowed

SIS0632 Duplicate key not allowed

SIS0633 Record currently locked

SIS0634 Current key out of sequence

SIS0635 Specified path undefined

SIS0637 End of line detected

SIS0638 Record truncated

SIS0640 No more space available to extend the file

SIS0643 Open type out of range or not allowed

SIS0644 Length of link string exceeds P_MAXLINKSTRG

SIS0645 Invalid version identification specified

SIS0646 Existence out of range

SIS0647 Syntax error in file, link or path string

SIS0649 Close type out of range or not allowed

SIS0650 Access not permitted

SIS0651 Parameter error

SIS0652 Invalid pointer to I/O area

SIS0653 Invalid record length detected

SIS0654 Space limit on device reached

SIS0655 Print control out of range or not allowed

SIS0656 Code set out of range or not allowed

SIS0657 Combination of open type and existence not allowed

SIS0658 I/O interrupted

SIS0659 Length of keyword exceeds P_MAXKEYWORD

SIS0660 Keyword ambiguous

SIS0661 Number of exits exceeds P_MAXEXITS

SIS0662 New line character detected

SIS0663 New page character detected

SIS0664 Not all pathes closed

SIS0665 Next indexed record has same secondary key

SIS0666 Secondary key of written record exists already

SIS0667 Current record number exceeds specified MAX_REC_NR

SIS0668 Path name exists already

SIS0669 Link name exists already

SIS0670 Positioning condition out of range

SIS0671 Unknown control character detected

SIS0672 No file name created

SIS0673 Last partial record not completed

SIS0674 Seek type out of range

SIS0675 Record format not determinable

SIS0676 MAXSIZE not determinable

SIS0677 PROSOS-D internal error

SIS0678 Specified file is a container of files

SIS0679 Specified file unreachable with given path

SIS0680 Version not incrementable

SIS0681 Reopen after implicit close failed

SIS0682 Failure on initialisation of PROSOS-D

SIS0683 Link indirections by extern variables exceed P_MAXLINKNESTING

SIS0901 Catalog handle is invalid

SIS0902 Catalog or view (&OO) not present

SIS0903 End of catalog or view reached

SIS0904 No more space available

SIS0905 Function not supported

SIS0906 (&OO) is file

10 Appendix

This chapter covers the following topics:

- Mixed-mode operation of the SQL and CALL DML interfaces
- Demonstration database

10.1 Mixed-mode operation of SQL and CALL DML interfaces

SESAM/SQL supports the CALL DML interface and the SQL interface. The interfaces can be used together in an ESQL-COBOL application (mixed-mode operation). On SESAM/SQL server V2.1 and lower, the two interfaces cannot be used together within a single transaction. Since SESAM/SQL server V2.2 there could be SQL statements in CALL DML transactions.

Resources

In mixed-mode operation, CALL DML statements only address resources that have been reserved as a result of CALL DML statements in the SESAM/SQL DBH. SQL statements only address resources reserved as a result of SQL statements. An application that uses both interfaces has signed itself off correctly from the DBH if the resources of both interfaces have been released.

Status codes

An application returns SQL status codes (SQLCODE or SQLSTATE or both) if the last statement passed to the DBH was an SQL statement. If the last statement passed to the DBH was a CALL DML statement, the application returns CALL DML status codes.

10.2 Demonstration database

Most of the examples in this manual reference the demonstration database described below. You are provided with the definition of each table and a representation of the data in the table. The demonstration database contains the schemas ORDER_PROC and PARTS.

10.2.1 Schema ORDER_PROC

Data concerning customers, contact persons, orders, order status and service can be administered in the order processing schema. The order processing schema comprises the tables CUSTOMERS, CONTACTS, ORDERS, SERVICE and ORDER_STAT.

10.2.1.1 CUSTOMERS table

The CUSTOMERS table is defined as follows:

```
CREATE TABLE          customers
(cust_num             INTEGER CONSTRAINT cust_num_primary PRIMARY KEY,
 company              CHAR(40) CONSTRAINT company_notnull NOT NULL,
 street               CHAR(40),
 zip                  NUMERIC(5),
 city                 CHAR(40),
 country              CHAR(3),
 cust_tel             CHAR(25),
 cust_info            CHAR(50),
CONSTRAINT PlausPlz   CHECK( country IS NULL OR zip IS NULL OR
                      (country = 'D' AND zip >= 00000)
                      OR (country <> 'D'))
)
```

The CUSTOMERS table contains the following data:

cust_ num	company	street	zip	city	country	cust_tel	cust_info
100	Siemens AG	Otto-Hahn-Ring 6	81739	Munich	D	089/636-8	Electrical
101	Login GmbH	Rosenheimer Str. 34	81667	Munich	D	089/4488870	PC networks
102	JIKO GmbH	Posener Str. 12	30659	Hanover	D	0551/123874	Import/ export
103	Plenzer Trading	Paul-Heyse-Str. 12	80336	Munich	D	089/923764	Fruit market
104	Freddy´s Fishery	Hirschgartenstr. 12	12587	Berlin	D	016/5739921	Unit retail
105	The Poodle Parlor	Am Muehlentor 26	41179	Moenchen-gladbach	D	040/873562	Service
106	Foreign Ltd.	26 West York St.		New York, NY	USA	001703/ 2386532	Commercial agency
107	Externa & Co KG	Berner Weg 78	3000	Berne 33	CH		Lawyer

Table 15: Data in the table CUSTOMERS

10.2.1.2 CONTACTS table

The CONTACTS table is defined as follows:

```
CREATE TABLE     contacts
(contact_num     INTEGER CONSTRAINT contact_num_primary PRIMARY KEY,
 cust_num        INTEGER CONSTRAINT co_cust_num_notnull NOT NULL,
 fname           CHAR(25),
 lname           CHAR(25) CONSTRAINT name_notnull NOT NULL,
 title           CHAR(20),
 contact_tel     CHAR(25),
 position        CHAR(50),
 department      CHAR(30),
 contact_info    CHAR(50),
 CONSTRAINT      co_cust_num_ref_customers FOREIGN KEY (cust_num)
                   REFERENCES customers
)
```

The CONTACTS table contains the following data:

contact_ num	cust_ num	fname	lname	title	contact_tel	position	depart- ment	contact_ info
10	100	Walter	Kuehne	Dr.	089/6361896	CEO	Personnel	
11	100	Stefan	Walkers	Mr.	089/63640182	Secretary	Sales	
20	101	Roland	Loetzerich	Mr.	089/4488870	Manager		Networks
25	102	Ewald	Schmidt	Mr.	0551/123873	Training		
26	103	Beate	Kredler	Ms.	089/923764	Organization		SQL course
30	104	Xaver	Bauer	Mr.	016/6739921	Sales exec.		
35	105	Anke	Buschmann	Ms.	02161/584097	Manager		
40	106	Mary	Davis	Ms.	001703/2386531	Management	Purchas- ing	
41	106	Robert	Heinlein	Mr.	001703/2386532	Trainer	Purchas- ing	

Table 16: Data in the table CONTACTS

10.2.1.3 ORDERS table

The ORDERS table is defined as follows:

```
CREATE TABLE     orders
(order_num       INTEGER CONSTRAINT order_num_primary PRIMARY KEY,
 cust_num        INTEGER CONSTRAINT o_cust_num_notnull NOT NULL,
 contact_num     INTEGER,
 order_date      DATE DEFAULT CURRENT_DATE,
 order_text      CHAR(30),
 actual          DATE,
 target          DATE,
 order_stat      INTEGER DEFAULT 1 CONSTRAINT order_stat_notnull NOT NULL,
 CONSTRAINT      o_cust_num_ref_customers FOREIGN KEY (cust_num)
                   REFERENCES customers,
 CONSTRAINT      contact_num_ref_contacts FOREIGN KEY (contact_num)
                   REFERENCES contacts,
 CONSTRAINT      order_stat_ref_ordstat FOREIGN KEY (order_stat)
                   REFERENCES ordstat(ord_stat_num,)
)
```

The ORDERS table contains the following data:

order_num	cust_num	contact_num	order_date	order_text	actual	target	order_stat
200	102	25	1990-04-10	Staff training	1990-05-01	1990-05-01	5
210	106	40	1990-12-13	Customer administration	1991-04-20	1991-04-01	3
211	106	41	1990-12-29	Database CUSTOMERS	1991-04-10	1991-04-01	4
250	105	35	1991-01-17	Mailmerge intro		1991-03-01	2
251	105	35	1991-01-17	Customer administration		1991-05-01	2
300	101	20	1991-02-15	Network test/comparison			1
305	105	35	1991-05-01	Staff training		1991-05-01	2

Table 17: Data in the table ORDERS

10.2.1.4 SERVICE table

The SERVICE table is defined as follows:

```
CREATE TABLE     service
(service_num     INTEGER CONSTRAINT service_num_primary PRIMARY KEY,
 order_num       INTEGER CONSTRAINT s_order_num_notnull NOT NULL,
 service_date    DATE,
 Service_text    CHAR(25),
 service_unit    CHAR(10),
 service_total   INTEGER CONSTRAINT service_total_pos
                   CHECK (service_total > 0),
 service_price   NUMERIC (5,0),
 vat             NUMERIC(2,2),
 inv_num         NUMERIC(4,0),
 CONSTRAINT      order_num_ref_orders FOREIGN KEY(order_num)
                   REFERENCES orders
)
```

The SERVICE table contains the following data:

service_num	order_num	service_date	service_text	service_unit	service_total	service_price	vat	inv_num
1	200	1990-04-20	Training documentation	Pages	45	75	0.15	3
2	200	1990-04-25	Training	Day	1	1500	0.15	3
3	200	1990-04-26	Training	Day	1	1500	0.15	3
4	211	1991-01-20	Systems analysis	Day	8	1200	0.00	10
5	211	1991-01-29	Database design	Day	10	1200	0.00	10
6	211	1991-02-15	Copies/transparencies	Pages	30	50	0.15	10
7	211	1991-03-27	Manual	Fixed price	1	200	0.07	10
10	250	1991-02-20	Travel expenses	Fixed price	2	125	0.00	
11	250	1991-02-20	Training	Day	0.5	1200	0.15	

Table 18: Data in the table SERVICE

10.2.1.5 ORDSTAT table

The ORDSTAT table is defined as follows:

```
CREATE TABLE     ordstat
(ord_stat_num    INTEGER CONSTRAINT ord_stat_num_primary PRIMARY KEY,
 ord_stat_text   CHAR(15) CONSTRAINT ord_stat_text_notnull NOT NULL
)
```

The ORDSTAT table contains the following data:

ord_stat_num	ord_stat_text
1	planned
2	contract
3	completed
4	paid
5	archived

Table 19: Data in the table ORDSTAT

10.2.2 Schema PARTS

The PARTS schema is used for managing parts. It comprises the tables ITEMS, PURPOSE, WAREHOUSE, COLOR_TAB and TABTAB.

10.2.2.1 ITEMS table

The ITEMS table is defined as follows:

```
CREATE TABLE   items
(item_num      INTEGER CONSTRAINT item_num_primkey PRIMARY KEY,
 item_name     CHARACTER(20) CONSTRAINT item_name_notnull NOT NULL,
 color         CHARACTER(15),
 price         NUMERIC(8,2) CONSTRAINT price_notnull NOT NULL,
 stock         INTEGER CONSTRAINT i_stock_notnull NOT NULL,
 min_stock     INTEGER,
)
```

The ITEMS table contains the following data:

item_num	item_name	color	price	stock	min_stock
1	Bicycle	black	700.50	2	1
10	Frame	black	150.00	10	5
11	Frame	edelweiss	150.00	10	5
120	Front wheel	metallic	40.00	3	5
130	Back wheel	metallic	40.00	12	5
200	Handlebars	metallic	60.00	1	5
210	Front hub	metallic	5.00	15	1
220	Back hub	metallic	5.00	14	10
230	Felge	black	10.00	9	10
240	Spoke	black	1.00	211	00
500	Screw M5	black	1.10	300	00
501	Nut M5	black	0.75	295	00

Table 20: Data in the table ITEMS

10.2.2.2 PURPOSE table

The PURPOSE table is defined as follows:

```
CREATE TABLE          purpose
(item_num             INTEGER CONSTRAINT p_item_num_notnull NOT NULL,
 part                 INTEGER CONSTRAINT part_notnull NOT NULL,
 number               INTEGER CONSTRAINT number_notnull NOT NULL,
 CONSTRAINT           p_item_num_ref_items FOREIGN KEY (item_num)
                        REFERENCES items,
 CONSTRAINT           part_ref_items FOREIGN KEY (part) REFERENCES items
)
```

The PURPOSE table contains the following data:

item_num	part	number
1	10	1
1	120	1
1	130	1
1	200	1
120	210	1
120	230	1
120	240	15
120	500	5
120	501	5
200	500	10
200	501	10

Table 21: Data in the table PURPOSE

10.2.2.3 WAREHOUSE table

The WAREHOUSE table is defined as follows:

```
CREATE TABLE     warehouse
(item_num        INTEGER CONSTRAINT w_item_num_notnull NOT NULL,
 stock           INTEGER CONSTRAINT w_stock_notnull NOT NULL,
 location        CHAR(25),
 CONSTRAINT      w_item_num_ref_items FOREIGN KEY (item_num)
                   REFERENCES items
)
```

The WAREHOUSE table contains the following data:

item_num	stock	location
1	2	Main warehouse
10	10	Main warehouse
11	10	Main warehouse
120	3	Main warehouse
130	3	Main warehouse
130	9	Parts warehouse
240	11	Main warehouse
240	200	Parts warehouse
500	120	Main warehouse
500	180	Parts warehouse

Table 22: Data in the table WAREHOUSE

10.2.2.4 COLOR_TAB table

The COLOR_TAB table is defined as follows:

```
CREATE TABLE      color_tab
(color_name       CHARACTER(15),
 rgb              (3)NUMERIC(2,2)
)
```

The COLOR_TAB table contains the following data:

color_name	**rgb**		
flame	0.98	0	0
orange	0.9	0.3	0
skyblue	0	0	0.99
aquamarine	0	0.1	0.99
edelweiss	0.99	0.99	0.99
black	0	0	0
metallic	0	0.2	0.3

Table 23: Data in the table COLOR_TAB

10.2.2.5 TABTAB table

The TABTAB table is defined as follows:

```
CREATE TABLE       tabtab
(table_num         INTEGER CONSTRAINT table_num_primkey PRIMARY KEY,
 table_name        CHARACTER(20) CONSTRAINT table_name_notnull NOT NULL,
 comment           CHARACTER(50),
)
```

The TABTAB table contains the following data:

table_num	table_name	comment
1	ITEMS	Parts data
2	PURPOSE	Related bicycle data
3	WAREHOUSE	Warehouse locations
4	COLOR_TAB	Permissible colors
5	TABTAB	Tables used

Table 24: Data in the table TABTAB

Glossary

This glossary contains the definitions of the most important terms used in this manual. Terms in *italics* indicate that there is actually an entry for this term in this glossary.
The “Synonym(s)” line, refers to terms with similar or identical meanings that are used in other documentation, but not in SESAM/SQL manuals.

base table

Table created using the CREATE TABLE statement. A base table is permanently stored in the *database*. Base tables are also generated as the result of migration. Base tables can have different table styles.
The number and data type of the columns as well as any integrity constraints can be defined using the *SQL* statement CREATE TABLE. ALTER TABLE can be used to modify them. The number of *rows* is not part of the table definition.

column

Part of a *table*. Each column is assigned a name and a data type and contains column values of this data type. Columns may be atomic or multiple columns.
Synonym: attribute

cursor

Pointer within a special type of derived *table*, the cursor table, that allows rows to be retrieved one at a time.
A cursor name is defined when the cursor is declared. The cursor description also includes an indication whether the cursor table is updatable or subject to a particular order criterion.

database

Related collection of data that is processed, manipulated and administered by the database system.
In SESAM/SQL, a database consists of the catalog in the catalog space and user data in the associated user spaces. A database is identified by its database name.

database handler (DBH)

SESAM/SQL component that analyzes, executes and coordinates all the database accesses of a DBH session .
The database handler (DBH) is available in two variants:

– the independent DBH
  This type of database handler is an independent program system that supports multi-user operation. The independent DBH executes under a separate task.
– the linked-in DBH
  This type of database handler is linked into, and exclusively processes the requests of, a single application program. The linked-in DBH executes under the same task as this application program.

Synonyms: SESAM/SQL DBH, DBH

embedded SQL statement

SQL statement embedded in a host language (e.g. C, COBOL) program. Its is started using EXEC SQL and ended using END-EXEC or ';'. This allows SQL statements to be clearly distinguished from host language statements and precompiled.

host variable

Variable in a host language (e.g. C, COBOL) referred to in an *embedded SQL statement*. Host variables are prefixed with a colon. They are declared in the DECLARE section.

indicator variable

Special type of *host variable* of the numeric data type SMALLINT, which is assigned to another host variable. The indicator variable indicates whether the other host variable contains the NULL value or whether data was lost during transfer of character-string values.

multiple column

Column which can contain more than one value of the same data type for each *row*. Each of these values is called an occurrence.
Synonyms: multiple attribute, multiple field

preparable SQL statement

SQL statement that is not compiled until the host language (e.g. C or COBOL) program is executed. Thus, database queries which were not known when the program was generated can be formulated dynamically.

row

Ordered sequence of values arranged horizontally in an SQL table.
Synonym: tuple

schema

Schemas are held in the catalog space of a *database*.
Schemas are subdivided into two categories: user-defined schemas and information schemas.
User-defined schemas contain the metadata for the base tables and *views* in the database. Moreover, user-defined schemas contain information on *privileges*.
A user-defined schema is assigned a name and an owner who can access it using his/her authorization identifier. User-defined schemas are created using the SQL statement CREATE SCHEMA and can be modified using various other statements (e.g. CREATE TABLE).
There must be two information schemas for each database: INFORMATION_SCHEMA and SYS_INFO_SCHEMA. They enable the user to access the metadata contained in the user-defined schemas, such as base table definitions, views, integrity constraints, privileges, etc.
The INFORMATION_SCHEMA can be accessed by every user using SQL. The SYS_INFO_SCHEMA contains system-specific data and can only be accessed by the universal user.

SQL

Structured query language. SQL is the language most commonly used for processing relational databases. In contrast to the procedural languages of non-relational database systems, SQL is descriptive, i.e. the user describes, in set-oriented form, the result of the operation rather than the steps to get there. SQL is based on the English language and can be easily comprehended. It offers extensive means of data definition, data manipulation, transaction management, access control, etc. Embedded SQL (ESQL) makes it possible to access a database using *embedded SQL statements* from host language programs (e.g. in C or COBOL). SQL was first standardized by the International Organization for Standardization in 1987. SESAM/SQL supports the current international standard ISO/IEC 9075:2003. This standard is referred to as SQL2 standard in this manual.

table

A table is a two-dimensional arrangement of data elements comprising *rows* (horizontal) and *columns* (vertical).
A distinction is made between *base tables*, *views* and derived tables.
Synonym: relation

transaction

Sequence of related statements (either SQL statements only or CALL DML statements only) which move a *database* from one consistent state to another consistent state. A transaction is performed either in its entirety or not at all.
Special statements are available for transaction processing:
CALL DML has one statement each for the following operations: opening, closing (committing) and rolling back a transaction.
SQL has special statements for ending and rolling back a transaction only. In UTM appliocations, the corresponding UTM calls must be used for this purpose.
There is no special SQL statement for defining the start of a transaction: the first transaction initiating statement after the previous transaction has been committed or rolled back or after the program has been started is taken by SQL to be the start of the transaction.

view

Named virtual table which is combined from one or more *tables*. A view is defined in a query expression using the CREATE VIEW or CREATE TEMPORARY VIEW (temporary view) statement. The derived table produced by the query expression is generated anew every time the view is referenced in an SQL statement.
The derived table thus always contains the most current data from the database.

Related publications

The manuals are available as online manuals, see *http://manuals.fujitsu-siemens.com*, or in printed form which must be paid and ordered separately at *http://FSC-manualshop.com*.

[1] **SESAM/SQL-Server** (BS2000/OSD)
Core Manual
User Guide

Target group
The manual is intended for all users and to anyone seeking information on SESAM/SQL.
Contents
The manual gives an overview of the database system. It describes the basic concepts. It is the foundation for understanding the other SESAM/SQL manuals.

[2] **SESAM/SQL-Server** (BS2000/OSD)
SQL Reference Manual Part 1: SQL Statements
User Guide

Target group
The manual is intended for all users who wish to process an SESAM/SQL database by means of SESAM/SQL statements.
Contents
The manual describes how to embed SQL statements in COBOL, and the SQL language constructs. The entire set of SQL statements is listed in an alphabetical directory.

[3] **SESAM/SQL-Server** (BS2000/OSD)
SQL Reference Manual Part 2: Utilities
User Guide

Target group
The manual is intended for all users responsible for SESAM/SQL database administration.
Contents
An alphabetical directory of all utility statements, i.e. statements in SQL syntax implementing the SESAM/SQL utility functions.

[4] **SESAM/SQL-Server** (BS2000/OSD)
CALL DML Applications
User Guide

Target group
SESAM application programmers
Contents
– CALL DML statements for processing SESAM databases using application programs
– Transaction mode with UTM and DCAM
– Utility routines SEDI61 and SEDI63 for data retrieval and direct updating
– Notes on using both CALL DML and SQL modes

[5] **SESAM/SQL-Server** (BS2000/OSD)
Database Operation
User Guide

Target group
The manual is intended for SESAM/SQL system administrators.
Contents
The manual covers the options available to the system administrator for controlling and monitoring database operation.

[6] **SESAM/SQL-Server** (BS2000/OSD)
Utility Monitor
User Guide

Target group
The manual is intended for SESAM/SQL-Server database and system administrators.
Contents
The manual describes the utility monitor. The utility monitor can be used to administer the database and the system. One aspect covered is its interactive menu interface.

[7] **SESAM/SQL-Server** (BS2000/OSD)
Messages
User Guide

Target group
All users of SESAM/SQL.
Contents
The message manual contains information relating to the structure and invocation of messages for the SESAM/SQL server database system and the distribution component SESAM/SQL-DCN. The SQL status codes and CALL DML status codes are also listed in full here.

[8] **SESAM/SQL-Server** (BS2000/OSD)
Migrating SESAM Databases and Applications to SESAM/SQL-Server
User Guide

Target group
Users of SESAM/SQL-Server.
Contents
This manual gives an overview of the new concepts and functions. Its primary subject is, however, the difference between the previous and the new SESAM/SQL version(s). It contains all the information a user may require to migrate to SESAM/SQL-Server V2.0.

[9] **SESAM/SQL-Server** (BS2000/OSD)
Performance
User Guide

Target group
Experienced users of SESAM/SQL.
Contents
The manual covers how to recognize bottlenecks in the behavior of SESAM/SQL and how to remedy this behavior.

[10] **openUTM V5.2**
Concepts and Functions
User Guide

Target group
Anyone who wants information about the functionality and performance capability of openUTM.
Contents
The manual contains a general description of all the functions and features of openUTM, plus introductory information designed to help first-time users of openUTM.

[11] **openUTM** (BS2000/OSD)
Generating and Handling Applications
User Guide

Target group
This manual is intended for application planners, technical programmers, administrators and users of UTM applications.
Contents
The manual describes the generation of UTM applications with distributed processing, the tools available with openUTM for this purpose, and the UTM objects created in the course of generation. It also contains all the information necessary for structuring, operating and monitoring a productive UTM application.

[12] **UTM** (TRANSDATA)
Programming Applications
User's Guide

Target group
Programmers of UTM applications
Contents
– Language-independent description of the KDCS program interface
– Structure of UTM programs
– KDCS calls
– Testing UTM applications
– All the information required by programmers of UTM applications
Applications
BS2000 transaction processing

[13] **openUTM V3.4A, openUTM-D V3.4A** (BS2000/OSD)
New Interfaces and Functions
Supplements to the UTM Manuals for V3.3A/3.4A
User Guide

Target group
Organizers, planners, programmers and administrators of UTM applications.
Contents
This is a supplement to the manuals for UTM V3.3A/UTM-D V2.0A. It contains a description of the X/Open interfaces CPI-C and XATMI under UTM, and the new functions in UTM V3.3B and UTM V3.4.

[14] **FHS** (TRANSDATA)
User Guide

Target group
Programmers
Contents
Program interfaces of FHS for TIAM, DCAM and UTM applications. Generation, application and management of formats.

[15] **CRTE V2.5A (BS2000/OSD)**
Common Runtime Environment
User Guide

Target group
This manual addresses all programmers and system administrators in a BS2000 environment.
Contents
It describes the common runtime environment for COBOL85, COBOL2000, C and C++ objects and for "language mixes":
– CRTE components
– ILCS program communication interface
– linkage examples

[16] **COBOL2000** (BS2000/OSD)
COBOL Compiler
Reference Manual

Target group
COBOL users in BS2000/OSD
Contents
– COBOL glossary
– Introduction to Standard COBOL
– Description of the full language set of the COBOL2000 compiler:
formats, rules and examples illustrating the COBOL ANS85 language elements of the "High" language subset, the Fujitsu Siemens-specific extensions and the extensions defined by the forthcoming COBOL standard, specifically the object orientation.

[17] **COBOL2000** (BS2000/OSD))
COBOL Compiler
User's Guide
Target group
COBOL users of BS2000/OSD
Contents
– Using the COBOL2000 compiler
– Linking, loading and starting of COBOL programs
– Debugging aids
– File processing with COBOL programs
– Checkpointing and restart
– Program linkage
– COBOL2000 and POSIX
– Useful software for COBOL users
– Messages of the COBOL2000 system

[18] **BS2000/OSD-BC**
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.

[19] **BINDER**
Binder in BS2000/OSD
User Guide

Target group
Software developers
Contents
The manual describes the BINDER functions, including examples. The reference section contains a description of the BINDER statements and BINDER macro.

[20] **BLSSERV**
Dynamic Binder Loader / Starter in BS2000/OSD
User Guide

Target group
This manual is intended for software developers and experienced BS2000/OSD users
Contents
It describes the functions, subroutine interface and XS support of the dynamic binder loader DBL as a component of the BLSSERV subsystem, plus the method used for calling it.

Other related literature

International Organization for Standardization (ISO):
Database Language SQL
ISO/IEC 9075:2003
Short title: SQL2 standard

An introduction to Database Systems,
C.J.Date
Addison Wesley, 2003

Index