INFORMIX-4GL

Quick Syntax
INFORMIX-4GL
Quick Syntax

Introduction  5
Syntax Conventions  6

Basics  9
Data Types  10
4GL Arithmetic Operators  10
Boolean Operators  10
4GL Relational Operators  10
Global Constants and Variables  11
Built-In Functions  11
Operators  11
Library Functions  12
Display Attributes  15
Command-Line Syntax  16

4GL Statements  19
Types of SQL Statements  20
Other Types of 4GL Statements  21
4GL Statement Segments  37

4GL Forms  47
Form Specification Syntax  48
Attributes  50

Reports  55
Report Specification Syntax  56
Report Execution Statements  59

SQL Statements  61
SQL Segments  91
Stored Procedure Language Statements  105
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLCA Record</td>
<td>113</td>
</tr>
<tr>
<td>Interactive Debugger Commands</td>
<td>117</td>
</tr>
<tr>
<td>Command-Line Syntax</td>
<td>119</td>
</tr>
<tr>
<td>4GL Interactive Debugger Command Segments</td>
<td>126</td>
</tr>
<tr>
<td>Environment Variables</td>
<td>129</td>
</tr>
<tr>
<td>Default Key Assignments</td>
<td>133</td>
</tr>
</tbody>
</table>
Introduction
This Guide presents a quick reference to the material listed in the following table. For a full discussion of each topic, refer to the corresponding documentation.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Reference Documentation</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data types</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>Built-in functions and operators</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 4</td>
</tr>
<tr>
<td>4GL statements</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 3</td>
</tr>
<tr>
<td>SQL statements</td>
<td>Informix Guide to SQL: Reference</td>
<td>Chapter 6</td>
</tr>
<tr>
<td></td>
<td>Informix Guide to SQL: Syntax, Version 6.0</td>
<td></td>
</tr>
<tr>
<td>4.1 servers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.0 servers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stored Procedure Language (SPL)</td>
<td>Informix Guide to SQL: Syntax, Version 6.0</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>6.0 servers:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4GL forms</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 5</td>
</tr>
<tr>
<td>Reports</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 6</td>
</tr>
<tr>
<td>SQLCA record structure</td>
<td>INFORMIX-4GL Reference</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>NewEra Debugger commands</td>
<td>Guide to the INFORMIX-4GL Interactive Debugger</td>
<td>Chapter 9</td>
</tr>
<tr>
<td>Environment variables</td>
<td>INFORMIX-4GL Reference</td>
<td>Appendix D</td>
</tr>
</tbody>
</table>

This Guide shows syntax that must be prepared before you can include it in a 4GL program. You must prepare any SQL statement introduced later than the 4.1 server release. These statements are indicated in this manual by the following icon:

6.0

To use these statements, such as CREATE TRIGGER, you must:

1. Store the SQL statement as a character string.
2. Set up the statement for execution by means of the PREPARE statement (see page 79).
3. Process the statement by means of the EXECUTE statement (see page 75).

Syntax diagrams describe the format of SQL, SPL, and 4GL statements and Debugger commands, including alternative forms of them, required and optional parts of them, and so forth. Syntax diagrams have their own conventions, which are defined in detail and illustrated in this section.

Each syntax diagram displays the sequences of required and optional elements that are valid in a statement or command. Briefly:

- All keywords are shown in uppercase letters for ease of identification, though you need not enter them that way.
- Words for which you must supply values are in italics.
- All boldface characters are literals.
Each diagram begins at the upper left with a keyword and ends at the upper right with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid form of the statement. Except for separators in loops (see page 8), which the path approaches counterclockwise from the right, the path always approaches elements from the left and continues to the right.

Along a path, you may encounter the following elements:

**KEYWORD**  You must spell a word in uppercase letters exactly as shown; however, you can use either uppercase or lowercase letters when you enter it.

`(;+)`  All other characters are literal symbols that you must enter exactly as shown.

' '  Single and double quotes are literal symbols that you must enter as shown.

**variable**  A word in italics represents a value that you must supply. The nature of the value is explained fully in the appropriate reference manual.

A reference in a box represents a subdiagram on the same page (if no page number is supplied) or on a specified page. Imagine that the subdiagram is spliced into the diagram at this point. (A synonym for “subdiagram” is “segment.”)

If the term “(subset)” appears in the box below the name of the segment being referenced, you should refer to the appropriate reference manual for further clarification.

The aspect ratio of the box is not significant. That is, the same segment can be represented by boxes of different shapes, as in these symbols for the Named Value segment.

A code in an icon is a signal warning you that this path is valid only for certain database servers or under certain conditions. The codes indicate the products or conditions that support the path. The following codes are used:

- **6.0**  Path requires the statement to be prepared (by using PREPARE).  
- **SE**  Path is valid only for INFORMIX-SE.  
- **OL**  Path is valid only for INFORMIX-OnLine Dynamic Server.  
- **+**  Path is an Informix extension to ANSI standard SQL.  
- **NLS**  Path is valid only if you are using NLS.
Syntax Conventions

A shaded option is the default. Even if you do not explicitly type the option, it will be in effect unless you choose another option.

Syntax enclosed in a pair of arrows indicates that this is a subdiagram.

The vertical line is a terminator and indicates that the statement is complete.

A branch below the main line indicates an optional path.

A loop indicates a path that can be repeated. Punctuation included in the loop indicates the separator symbol for items in a list; when no symbol is shown in the loop, as in the example below, a blank space or LINEFEED is the separator.

A gate (__) in an option indicates that you can only use that option the number of times indicated, even though it is within a larger loop.

The grey labels and arrows in the following illustration identify the elements of a syntax diagram for the INITIALIZE statement of 4GL.

---

Elements of a syntax diagram
Basics
Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Kind of Values Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY OF type</td>
<td>Arrays of values of any other single data type.</td>
</tr>
<tr>
<td>^BYTE</td>
<td>Any kind of binary data.</td>
</tr>
<tr>
<td>CHAR</td>
<td>Character strings of up to 32,767 ASCII characters.</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>(This keyword is a synonym for CHAR.)</td>
</tr>
<tr>
<td>DATE</td>
<td>Points in time, specified as calendar dates.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Points in time, specified as calendar dates and time-of-day.</td>
</tr>
<tr>
<td>DEC</td>
<td>(This keyword is a synonym for DECIMAL.)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Fixed point numbers, of a specified scale and precision.</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>(These keywords are a synonym for FLOAT.)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Floating-point numbers, of up to 32-digit precision.</td>
</tr>
<tr>
<td>INT</td>
<td>(This keyword is a synonym for INTEGER.)</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Whole numbers, from -2,147,483,647 to +2,147,483,647.</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>Spans of time in years and months, or else in smaller time units.</td>
</tr>
<tr>
<td>MONEY</td>
<td>Currency amounts, with definable scale and precision.</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>(This keyword is a synonym for DECIMAL.)</td>
</tr>
<tr>
<td>REAL</td>
<td>(This keyword is a synonym for SMALLFLOAT.)</td>
</tr>
<tr>
<td>RECORD</td>
<td>Ordered sets of values, of any combination of 4GL data types.</td>
</tr>
<tr>
<td>SERIAL</td>
<td>Same as INTEGER. Automatically assigned by the engine.</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>Floating-point numbers, of up to 16-digit precision.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Whole numbers, from -32,767 to +32,767.</td>
</tr>
<tr>
<td>TEXT</td>
<td>Character strings of any length.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>Character strings of varying length, no greater than 255.</td>
</tr>
</tbody>
</table>

^INFORMIX-OnLine Dynamic Server or other 4GL statements only.  SQL only.

4GL Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator Symbol</th>
<th>Operator Name</th>
<th>Name of Result</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>power</td>
<td>12</td>
</tr>
<tr>
<td>mod</td>
<td>modulus</td>
<td>integer remainder</td>
<td>12</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>product</td>
<td>11</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>quotient</td>
<td>11</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
<td>sum</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>difference</td>
<td>10</td>
</tr>
</tbody>
</table>

4GL Boolean Operators

AND

OR

NOT

4GL Relational Operators

<table>
<thead>
<tr>
<th>Operator Symbol</th>
<th>Operator Name</th>
<th>Operator Symbol</th>
<th>Operator Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>!= or &lt;&gt;</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Not greater than</td>
<td>&gt;=</td>
<td>Not less than</td>
</tr>
<tr>
<td>= or ==</td>
<td>Equal to</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
</tbody>
</table>
Global Constants and Variables

**Global Constants and Variables**

- FALSE
- INT_FLAG
- NOTFOUND
- NULL
- SQLCODE
- STATUS
- TRUE
- QUIT_FLAG

**Built-In Functions**

- ARG_VAL(int-expr)
- ARR_COUNT()
- ARR_CURR()
- †AVG(int-field)
- †COUNT(*)
- DOWNSHIFT(char-expr)
- ERR_GET(int-expr)
- ERR_PRINT(int-expr)
- ERR.Quit(int-expr)
- ARG_VAL(int-expr)
- ERRORLOG(char-expr)
- NUM_ARGS()
- FGL_DRAWBOX(height, width, line, left-offset, color)
- SCR_LINE()
- FGL_GETENV(char-expr)
- SET_COUNT(int-expr)
- FGL_KEYVAL(char-expr)
- SHOWHELP(int-expr)
- FGL_LASTKEY()
- SQLEXIT()
- LENGTH(char-expr)
- STARTLOG(char-expr)
- MAX(int-field)
- SUM(int-field)
- MIN(int-field)
- UPSHIFT(char-expr)

†Valid only in REPORT blocks or in some SQL statements. Also, may be preceded by GROUP.

**Color numbers and their meanings that can be used in FGL_DRAWBOX( ) are:**

<table>
<thead>
<tr>
<th>Number</th>
<th>Color</th>
<th>Number</th>
<th>Color</th>
<th>Number</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>White</td>
<td>3</td>
<td>Red</td>
<td>6</td>
<td>Blue</td>
</tr>
<tr>
<td>1</td>
<td>Yellow</td>
<td>4</td>
<td>Cyan</td>
<td>7</td>
<td>Black</td>
</tr>
<tr>
<td>2</td>
<td>Magenta</td>
<td>5</td>
<td>Green</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operators**

- ASCII int-expr
- char-expr CLIPPED
- COLUMN integer
- CURRENT
- CURRENT qualifier
- DATE
- DATE (date-expression)
- DAY(date-expression)
- EXTEND(time-value)
- EXTEND(time-value, qualifier)
- FIELD_TOUCHED(field-list)
- GET_FLDBUF(field-list)
- expression IS NOT NULL
- expression IS NULL
- INFIELD(field)
- LENGTH(char-expr)
- †LINENO
- MDY(int-expr, int-expr, int-expr)
- MONTH(date-expression)
- ORD(string-expr)
- †PAGENO
- DATE
- TIME
- TODAY
- int-expr SPACE
- int-expr SPACES
- int-expr UNITS time-keyword
- expression USING format-string
- WEEKDAY (date-expression)
- †char-expr WORDWRAP
- YEAR (date-expression)
- †Valid only in REPORT blocks.
Library Functions

Calling C Functions from 4GL

Popping Numbers
extern void popint(int *iv)
extern void popshort(short *siv)
extern void poplong(long *liv)
extern void popflo(float *fv)
extern void popdub(double *dfv)
extern void popdec(dec_t *decv)

Popping Characters
extern void popquote(char *qv, int len)
extern void popvchar(char *qv, int len)

Popping Dates and Times
extern void popdate(long *datv)
extern void popdtime(dtime_t *dtv, int qual)
extern void popinv(intrvl_t *iv, int qual)

Popping BYTE and TEXT
extern void poplocator(loc_t **blob)

Returning Values
extern void retint(int iv)
extern void retshort(short siv)
extern void retlong(long lv)
extern void retflo(float fv)
extern void retdub(double dfv)
extern void retdec(dec_t *decv)

extern void retquote(char *str0)
extern void retvchar(char *vc)

extern void retdate(long date)
extern void retdtime(dtime_t *dtv)
extern void retinv(intrvl_t *inv)

Pushing Values
extern void pushint(int iv)
extern void pushshort(short siv)
extern void pushlong(long liv)
extern void pushflo(float fv)
extern void pushdub(double dfv)
extern void pushdec(dec_t *decv, unsigned decp)
Calling 4GL Functions from C

fgl_start(filename, argc, argv)
char *filename;
int argc;
char *argv[];
initialize resources for the 4GL environment

fgl_call(funcname, nparams)
char *funcname;
int nparams;
call the 4GL function

fgl_exitfm()
reset terminal to character mode

fgl_end()
free 4GL resources

decvasc(cp, len, np)
char *cp;
int len;
dec_t *np;
convert C char type to DECIMAL type

dectoasc(np, cp, len, right)
dec_t *np;
char *cp;
int len;
int right;
convert DECIMAL type to C char type

decvint(integer, np)
int integer;
dec_t *np;
convert C int type to DECIMAL type

dectoint(np, ip)
dec_t *np;
int *ip;
convert DECIMAL type to C int type

decvlong(lng, np)
long lng;
dec_t *np;
convert C long type to DECIMAL type

dectolong(np, lngp)
dec_t *np;
long *lngp;
convert DECIMAL type to C long type

decvflt(flt, np)
float flt;
dec_t *np;
convert C float type to DECIMAL type

dectoflt(np, fltp)
dec_t *np;
float *fltp;
convert DECIMAL type to C float type

Decimal Functions
Decimal Functions

```c
deccvdbl(dbl, np)
   convert C double type to DECIMAL type
   double dbl;
   dec_t *np;

dectodb(np, dblp)
   convert DECIMAL type to C double type
   dec_t *np;
   double *dblp;

decadd(n1, n2, result)
   add two decimal numbers
   (result = n1 + n2)
   dec_t *n1;
   dec_t *n2;
   dec_t *result;

decsub(n1, n2, result)
   subtract two decimal numbers
   (result = n1 - n2)
   dec_t *n1;
   dec_t *n2;
   dec_t *result;

decmul(n1, n2, result)
   multiply two decimal numbers
   (result = n1 * n2)
   dec_t *n1;
   dec_t *n2;
   dec_t *result;

decdiv(n1, n2, result)
   divide two decimal numbers
   (result = n1 / n2)
   dec_t *n1;
   dec_t *n2;
   dec_t *result;

int deccmp(n1, n2)
   compare two decimal numbers
   dec_t *n1;
   dec_t *n2;

deccopy(n1, n2)
   copy a decimal number
   dec_t *n1;
   dec_t *n2;

char *dececvt(np, ndigit, decpt, sign)
   convert decimal value to ASCII string
   dec_t *np;
   int ndigit;
   int *decpt;
   int *sign;

char *decfcvt(np, ndigit, decpt, sign)
   convert decimal value to ASCII string
   dec_t *np;
   int ndigit;
   int *decpt;
   int *sign;
```
The ATTRIBUTE clause is used in these 4GL statements:

<table>
<thead>
<tr>
<th>Construct</th>
<th>Display Form</th>
<th>Input Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRUCT</td>
<td>DISPLAY</td>
<td>INPUT ARRAY</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>ERROR</td>
<td>MESSAGE</td>
</tr>
<tr>
<td>DISPLAY ARRAY</td>
<td>INPUT</td>
<td>PROMPT</td>
</tr>
</tbody>
</table>

For all ATTRIBUTE clauses and field attributes the following table shows the effects of the color attributes on a monochrome monitor, as well as the effects of the intensity attributes on a color monitor:

<table>
<thead>
<tr>
<th>Color Attribute</th>
<th>Monochrome Display</th>
<th>Intensity Attribute</th>
<th>Color Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHITE</td>
<td>NORMAL</td>
<td>NORMAL</td>
<td>WHITE</td>
</tr>
<tr>
<td>YELLOW</td>
<td>BOLD</td>
<td>BOLD</td>
<td>RED</td>
</tr>
<tr>
<td>MAGENTA</td>
<td>BOLD</td>
<td>DIM</td>
<td>BLUE</td>
</tr>
<tr>
<td>RED</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CYAN</td>
<td>DIM</td>
<td>DIM</td>
<td></td>
</tr>
<tr>
<td>GREEN</td>
<td>DIM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLUE</td>
<td>DIM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLACK</td>
<td>DIM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Command-Line Syntax

```
-ansi -e -a -anyerr

-source.4gl

-esqlc.ec

-source.c

-obj.o

-yourlib

-o outfile

-args

-V

-c4gl

-fgioso.c

-cfile

-ec

-o newfglgo

-V cfile

-V

-fglgo

-a -anyerr

-argument

-fglio

-V

-fglpc

-ansi -a -anyerr

-p pathname

-V

-finderr

-msg_num

-V

-fglio

-c4gl
```

16 Command-Line Syntax
form4gl

-m lines -c characters

-V

-d form-name
database-name
table-name

.mkmessage in file out file

Basics

4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
4GL Statements
INFORMIX-4GL supports the SQL language, but it is sometimes convenient to distinguish between SQL statements and other 4GL statements:

- SQL statements operate on tables in the database.
- Other 4GL statements operate on variables in memory.

The SQL statements of 4GL can be divided into these functional categories.

**Note:** Not all of these SQL statements listed on this and the next page are directly supported by 4GL. If the statement or any part of its syntax is preceded by a 6.0 icon in its syntax diagram later in this chapter, the statement must be prepared (by using the PREPARE statement). Preparing SQL statements is described in Chapter 3 of the INFORMIX-4GL Reference.

### Types of SQL Statements

#### SQL Data Definition Statements
- ALTER INDEX
- CREATE TABLE
- DROP TABLE
- ALTER TABLE
- CREATE VIEW
- DROP VIEW
- CLOSE DATABASE
- DATABASE
- RENAME COLUMN
- CREATE DATABASE
- DROP DATABASE
- RENAME TABLE
- CREATE INDEX
- DROP INDEX
- CREATE SYNONYM
- DROP SYNONYM

#### SQL Data Manipulation Statements
- INSERT
- LOAD
- UNLOAD
- DELETE
- SELECT
- UPDATE

#### SQL Cursor Manipulation Statements
- CLOSE
- FETCH
- OPEN
- DECLARE
- FLUSH
- PUT

#### SQL Query Optimization Information Statements
- SET EXPLAIN
- SET OPTIMIZATION
- UPDATE STATISTICS

#### SQL Data Access Statements
- GRANT
- REVOKE
- SET LOCK MODE
- LOCK TABLE
- SET ISOLATION
- UNLOCK TABLE
- START DATABASE

#### SQL Data Integrity Statements
- BEGIN WORK
- DROP AUDIT
- ROLLFORWARD DATABASE
- COMMIT WORK
- RECOVER TABLE
- SET CONSTRAINTS
- CREATE AUDIT
- REPAIR TABLE
- SET LOG
- ROLLBACK WORK
- START DATABASE

**Note:** The data integrity statements marked with the symbol are supported only by the INFORMIX-SE engine. Statements marked can only be used with the INFORMIX-OnLine Dynamic Server engine.
Other Types of 4GL Statements

SQL Dynamic Management Statements
EXECUTE PREPARE
FREE

4GL Definition and Declaration Statements
DEFINE MAIN
FUNCTION REPORT

4GL Program Flow Control Statements
CALL FINISH REPORT OUTPUT TO REPORT
CASE FOR RETURN
CONTINUE FOREACH START REPORT
DATABASE GOTO WHILE
END IF
EXIT LABEL

4GL Compiler Directives
DATABASE GLOBALS
DEFER WHENEVER

4GL Storage Manipulation Statements
INITIALIZE LOCATE
LET VALIDATE

4GL Screen Interaction Statements
CLEAR DISPLAY FORM OPEN WINDOW
CLOSE FORM ERROR OPTIONS
CLOSE WINDOW INPUT PROMPT
CONSTRUCT INPUT ARRAY SCROLL
CURRENT WINDOW MENU SLEEP
DISPLAY MESSAGE
DISPLAY ARRAY OPEN FORM

4GL Report Execution Statements
NEED PRINT
PAUSE SKIP

Most 4GL statements are not sensitive to whether the SE or the OnLine engine supports the application. Only the OnLine engine, however, can store values in BYTE, TEXT, or VARCHAR columns, or can accept database: or database@system: as qualifiers to names of tables, views, or synonyms.
CALL

CALL function, returning

RETURNING Receiving Variable

4GL Expression

p. 39

Case I: (single criterion)

CASE

WHEN 4GL Expression (subset) p. 39

statement

EXIT CASE

END CASE

Case II: (multiple criteria)

WHEN 4GL Boolean Expression p. 40

statement

EXIT CASE

OTHERWISE Block

OTHERWISE

statement

EXIT CASE

CLEAR

FORM

WINDOW

window

SCREEN

Field Clause p. 46

CLOSE FORM form

CLOSE WINDOW window
CONSTRUCT

ATTRIBUTE Clause p. 37

HELP number

END CONSTRUCT

CONSTRUCT Variable Clause

variable ON

Column List FROM

Field Clause p. 46

BY NAME variable ON

Column List

Column List

Table Qualifier p. 46

table.

Table Qualifier p. 46

table.

CONSTRUCT Form Management Block

BEFORE CONSTRUCT

statement

NEXT FIELD

PREVIOUS

NEXT

CONTINUE CONSTRUCT

EXIT CONSTRUCT

CONTINUE keyword

CURRENT WINDOW IS

window

SCREEN

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
Case I: (output in the Line mode overlay)

Case II: (in a specified line of the current window)

Case III: (in a screen form)

DISPLAY ARRAY record array TO screen array.

CLIPPED USING "format string" ASCII number

ATTRIBUTE Clause p. 37 ON KEY Block p. 25 END DISPLAY

DISPLAY Value BY NAME variable COLUMN left-offset

DISPLAY Value AT line, left-offset

DISPLAY Value TO Field Clause p. 46 ATTRIBUTE Clause p. 37

DISPLAY Value

Data Type Declaration p. 37
These are valid keywords in the END statement:

- CASE
- CONSTRUCT
- DISPLAY
- FOR
- IF
- FOREACH
- FUNCTION
- INPUT
- PROMPT
- REPORT
- MAIN
- MENU
- WHILE

Case I: (terminating a program)

Case II: (terminating a statement)

These are valid keywords in Case II of the EXIT statement:

- CASE
- CONSTRUCT
- DISPLAY
- FOR
- IF
- FOREACH
- FUNCTION
- INPUT
- PROMPT
- MENU
- Reports
- SQL
- SQLCA
- Debugger
- Variables
- Keys
FOR counter = start TO finish
  STEP increment
  CONTINUE FOR
  EXIT FOR
END FOR

FOREACH cursor
  INTO Variable List
  CONTINUE FOREACH
  EXIT FOREACH
END FOREACH

Variable List

FUNCTION function
  ( argument )
  DEFINE Statement p. 24
  RETURN Statement p. 35
END FUNCTION

Case I: (declaring and exporting variables)

Case II: (importing variables)

"filename"
GOTO label name

IF 4GL Boolean Expression p. 40 THEN statement END IF
ELSE statement

INITIALIZE Variable List LIKE table column TO NULL

Variable List

variable

record first THROUGH last

THRU

Array [Integer Expression p. 41]

INPUT Binding Clause p. 28 ATTRIBUTE Clause p. 37 HELP number INPUT Form Management Block END INPUT

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
Acceptable values of key (in lowercase or uppercase letters) for the ON KEY block are:

- ACCEPT
- DELETE
- DOWN
- ESC or ESCAPE
- Fi through F64
- CONTROL-char (except A, D, H, I, J, L, M, R, or X)
- NEXT or NEXTPAGE
- RIGHT
- INTERRUPT
- PREVIOUS or PREVPAGE
- TAB
- F1 through F64
- INPUT
- CONTINUE
- EXIT
- NEXT
- PREVIOUS
- RIGHT
- LEFT
- UP
- F1 through F64
- CONTROL-char (except A, D, H, I, J, L, M, R, or X)

Built-in functions that access field buffers and keystroke buffers:

**Built-In Function**

FIELD_TOUCHED(field) Returns TRUE when the user has made a change to screen field.

GET_FLDBUF(field-list) Returns the character values of the contents of one or more fields.

FGL_LASTKEY() Returns an INTEGER value corresponding to the most recent keystroke.

INFIELD(field) Returns TRUE if field is the name of the current screen field.
For acceptable values of `key`, see p. 28. For built-in functions that access field buffers and keystroke buffers, see p. 28.

Built-in functions that keep track of the relative states of the screen cursor, the program array, and the screen array:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR_CURR( )</td>
<td>Returns the number of the current record of the program array.</td>
</tr>
<tr>
<td>ARR_COUNT( )</td>
<td>Returns the current number of records in the program array.</td>
</tr>
<tr>
<td>SCR_LINE( )</td>
<td>Returns the number of the current line within the screen array.</td>
</tr>
<tr>
<td>SET_COUNT(filled-rows)</td>
<td>Sets the initial value of ARR_COUNT( ) to filled-rows.</td>
</tr>
</tbody>
</table>

For acceptable values of `key`, see p. 28. For built-in functions that access field buffers and keystroke buffers, see p. 28.
Attribute Default Setting

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Default foreground color on your terminal</td>
</tr>
<tr>
<td>REVERSE</td>
<td>No reverse video</td>
</tr>
<tr>
<td>BORDER</td>
<td>No border</td>
</tr>
<tr>
<td>PROMPT LINE line value</td>
<td>FIRST (=1)</td>
</tr>
<tr>
<td>MESSAGE LINE line value</td>
<td>FIRST + 1 (=2)</td>
</tr>
<tr>
<td>MENU LINE line value</td>
<td>FIRST (=1)</td>
</tr>
<tr>
<td>FORM LINE line value</td>
<td>FIRST + 2 (=3)</td>
</tr>
<tr>
<td>COMMENT LINE line value</td>
<td>LAST - 1 (for the 4GL screen)</td>
</tr>
<tr>
<td></td>
<td>LAST (for all other 4GL windows)</td>
</tr>
</tbody>
</table>

Reserved Line Position

```
FIRST + integer
```

```
LAST - integer
```
Clause | Default
---|---
COMMENT | LAST - 1 for the 4GL screen
LINE | LAST for all other 4GL windows
ERROR LINE | LAST line of the 4GL screen
FORM LINE | FIRST + 2 or line 3 of the current 4GL window
MENU LINE | FIRST line of the 4GL window
MESSAGE LINE | FIRST + 1 or line 2 of the current 4GL window
PROMPT LINE | FIRST line of the 4GL window
ACCEPT KEY | ESCAPE
DELETE KEY | F2
INSERT KEY | F1
NEXT KEY | F3
PREVIOUS KEY | F4
HELP KEY | CONTROL-W
HELP FILE | None
For values for `key`, see p. 28.
ATTRIBUTE Clause

4GL Statement Segments

ATTRIBUTE Clause

ATTRIBUTE

REVERSE

BLACK
BLUE
CYAN
GREEN
MAGENTA
RED
WHITE
YELLOW

BLINK
UNDERLINE

4GL Data Type

LIKE

table . column

Table Qualifier
p. 46

4GL Data Type

4GL Simple Data Type
p. 38

ARRAY Data Type
p. 38

RECORD Data Type
p. 39

TEXT
BYTE

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
4GL Simple Data Type

**Integer (Int)**
- `INTEGER`
- `INT`

**Second Level Subtypes**
- `SMALLINT`
- `DECIMAL` with `precision` and `scale`
- `NUMERIC` with `precision` and `scale`
- `MONEY`
- `FLOAT`
- `DOUBLE PRECISION`
- `REAL`
- `DATE`
- `DATETIME`
- `INTERVAL`

**Third Level Subtypes**
- `CHARACTER` with `size`
- `VARCHAR` with `maximum size` and `reserve size`
- `MONEY` with `size`, `precision`, and `scale`
- `SMALLFLOAT`
- `REAL`
- `DATE`
- `DATETIME`
- `INTERVAL`

**Array Data Type**
- `ARRAY` with `size`

**Fourth Level Subtypes**
- `4GL Simple Data Type`

**Record Data Type**
- `RECORD`
- `BYTE`
- `TEXT`
4GL Boolean Expression

- AND
- OR
- NOT

4GL Expression

Boolean Comparison

Function Call

TRUE
FALSE

String Comparison

Set Membership Test

NULL Test

Relational Comparison

4GL Expression

Relational Comparison:
- <
- >=
- <=
- >=
- !=

4GL Expression

String Comparison

Character Expression

M A T C H E S criterion

ESCAPE "char"

NOT
LIKE
MATCHES
Wildcard Effect
*  Matches a string of zero or more characters.
?  Matches any single character.
[ ] Matches any of the enclosed characters.
-  Between characters in brackets means a range in the ASCII collating sequence.
^  As the first character in the brackets, matches any character that is not listed.
\ Treats the next character as a literal.

LIKE
Wildcard Effect
%  Matches a string of zero or more characters.
_  Matches any single character.
\  Treats the next character as a literal.
Case I: Time units smaller than MONTH

- DAY
- HOUR
- MINUTE
- SECOND
- FRACTION

Case II: Time units greater than DAY

- YEAR
- MONTH

Interval (Numeric Time Interval p. 46) INTERVAL Qualifier
Case I: Time units smaller than MONTH

Case II: Time units greater than DAY

Field Clause

Table Qualifier

THRU Notation

Numeric Time Interval
4GL Forms
Form Specification Syntax

DATABASE Section

DATABASE

SCREEN

TABLES Section p. 49

ATTRIBUTES Section p. 49

INSTRUCTIONS Section p. 49

DATABASE Section

Database Reference

FORMONLY

WITHOUT NULL INPUT

database

database @server

"///server/database"

"pathname/database@server"

"///server/pathname/database"

SCREEN Section

SCREEN

Screen Layout

SIZE lines

BY characters

END

Screen Layout

[ field-tag ]

[ field-tag | field-tag ]

character
Values for display mode consists of zero or one color and zero or more intensities:

<table>
<thead>
<tr>
<th>Color Keywords</th>
<th>Intensity Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLACK</td>
<td>MAGENTA</td>
</tr>
<tr>
<td>BLUE</td>
<td>RED</td>
</tr>
<tr>
<td>CYAN</td>
<td>WHITE</td>
</tr>
<tr>
<td>GREEN</td>
<td>YELLOW</td>
</tr>
<tr>
<td></td>
<td>REVERSE</td>
</tr>
<tr>
<td></td>
<td>LEFT</td>
</tr>
<tr>
<td></td>
<td>BLINK</td>
</tr>
<tr>
<td></td>
<td>UNDERLINE</td>
</tr>
</tbody>
</table>
For DATE data types, \textit{format-string} consists of:

<table>
<thead>
<tr>
<th>Special Characters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm</td>
<td>2-digit representation of the month</td>
</tr>
<tr>
<td>mmm</td>
<td>3-letter abbreviation of the month</td>
</tr>
<tr>
<td>dd</td>
<td>2-digit representation of the day of the month</td>
</tr>
<tr>
<td>ddd</td>
<td>3-letter abbreviation of the day of the week</td>
</tr>
<tr>
<td>yy</td>
<td>2-digit representation of the year, discarding the leading digits</td>
</tr>
<tr>
<td>yyyy</td>
<td>4-digit representation of the year</td>
</tr>
</tbody>
</table>

All other characters are literals.

For DECIMAL, SMALLFLOAT, or FLOAT data types, \textit{format-string} consists of pound signs (\#) to represent digits and a decimal point. If you are using NLS, the period is a placeholder for the decimal separator and the comma is a placeholder for the thousands separator.
A format-string can include these three special symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Any letter</td>
</tr>
<tr>
<td>#</td>
<td>Any digit</td>
</tr>
<tr>
<td>X</td>
<td>Any character</td>
</tr>
</tbody>
</table>

PICTURE Attribute

\[ \text{PICTURE} = "\text{format-string}" \]

PROGRAM Attribute

\[ \text{PROGRAM} = "\text{command}" \]

REQUIRED Attribute

\[ \text{REQUIRED} \]

REVERSE Attribute

\[ \text{REVERSE} \]

UPSHIFT Attribute

\[ \text{UPSHIFT} \]

VALIDATE LIKE Attribute

\[ \text{VALIDATE LIKE} \quad \text{table} . \quad \text{column} \]

VERIFY Attribute

\[ \text{VERIFY} \]
WORDWRAP Attribute

WORDWRAP

COMPRESS

NONCOMPRESS
ORDER BY Section

LEFT MARGIN size (default = 5 characters)

TOP MARGIN size (default = 3 lines)

PAGE LENGTH size (default = 66 lines)

BOTTOM MARGIN size (default = 3 lines)

RIGHT MARGIN size (default = 132 lines)
(for default reports or PRINT WORDWRAP only)

ORDER BY Section

ORDER

EXTERNAL

BY

argument

ASC

DESC

ORDER BY Section

Basics

4GL

Forms

Reports

SQL

SQLCA

Debugger

Variables

Keys
Following is the execution sequence of report control blocks.

BEFORE GROUP OF a  {1}
BEFORE GROUP OF b  {2}
BEFORE GROUP OF c  {3}
ON EVERY ROW  {4}
AFTER GROUP OF c  {3}
AFTER GROUP OF b  {2}
AFTER GROUP OF a  {1}
Report Execution Statements

NEED lines LINES

PAUSE "string"
SQL Statements
The 4GL source compiler does not recognize SQL statements identified in this Guide by the icon nor SQL statements containing a clause identified by the icon. To compile 4GL source code containing such statements, you must do the following:

1. Store the 6.0 SQL statement as a character string.
2. Set up the statement for execution by means of the PREPARE statement (see p. 79).
3. Process the statement by means of the EXECUTE statement (see p. 75).
BEGIN WORK

ADD CONSTRAINT Clause

ADD CONSTRAINT

Constraint Definition p. 68

DROP CONSTRAINT

DROP CONSTRAINT

Constraint Name p. 92

MODIFY NEXT SIZE kbytes

MODIFY NEXT SIZE

LOCK MODE

LOCK MODE

PAGE

ROW

BEGIN WORK

CLOSE

Cursor Name p. 92
CREATE INDEX

CREATE

UNIQUE
DISTINCT
CLUSTER

INDEX

ON Clause

ON

Table Name
p. 92

Synonym Name
p. 92

ON Clause

column name

ASC
DESC

FILLFACTOR

percent

CREATE PROCEDURE FROM

" filename "

Named Value
p. 103

CREATE

SYNONYM

FOR

Table Name
p. 92

Synonym Name
p. 92

View Name
p. 92

PUBLIC
PRIVATE
CREATE TABLE

**Column-Level Constraint Definition**
- **UNIQUE**
- **DISTINCT**
- **PRIMARY KEY**
- **REFERENCES**
  - Clause p. 69
- **CHECK**
  - Clause p. 69

**Temp-Table Column Constraint**
- **UNIQUE**
- **DISTINCT**
- **PRIMARY KEY**
- **CHECK**
  - Clause p. 69

**Constraint Definition**
- **UNIQUE**
- **DISTINCT**
- **PRIMARY KEY**
- **FOREIGN KEY**
  - column
- **REFERENCES**
  - Clause p. 69
- **CHECK**
  - Clause p. 69

**Constraint Name** p. 92
CREATE TRIGGER

LOCK MODE Clause

LOCK MODE

PAGE

ROW

6.0

CREATE TRIGGER

Trigger Name

INSERT — ON

Table Name p. 92

Action Clause p. 71

Insert REFERENCING Clause p. 71

Action Clause Referencing p. 72

DELETE — ON

Table Name p. 92

Action Clause p. 71

Delete REFERENCING Clause p. 71

Action Clause Referencing p. 72

UPDATE Clause p. 71

Table Name p. 92

Action Clause p. 71

Update REFERENCING Clause p. 72

Action Clause Referencing p. 72

Trigger Name

owner.

Identifier p. 100
CREATE TRIGGER

UPDATE Clause

UPDATE

OF column name

Action Clause

BEFORE Triggered Action List p. 72

FOR EACH ROW Triggered Action List p. 72

FOR EACH ROW Triggered Action List p. 72

AFTER Triggered Action List p. 72

AFTER Triggered Action List p. 72

AFTER Triggered Action List p. 72

INSERT REFERENCING Clause

REFERENCING NEW correlation name AS

DELETE REFERENCING Clause

REFERENCING OLD correlation name AS

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
A procedure that returns no values must be executed by using PREPARE and EXECUTE. A procedure that returns values must be handled by using PREPARE and DECLARE, and then either a FOREACH loop or OPEN, FETCH, or CLOSE.
The 4GL compiler treats the name of the object to be freed in the order shown in the diagram. In other words, the compiler looks first for a TEXT or BYTE variable having the given name; if one exists, that is the object that is freed. If no TEXT or BYTE variable having that name exists, the compiler then looks for a prepared statement or a cursor having that name and frees that.

When a TEXT or BYTE variable has the same name as a prepared statement or cursor, you cannot free resources allocated to the prepared statement or to the cursor.
Table-Level Privileges

INSERT INTO

VALUES Clause

VALUES Clause

SELECT Statement (Subset)

EXECUTE PROCEDURE

VALUES Clause

SELECT Statement (Subset)

EXECUTE PROCEDURE

Table-Name p. 92

Table-Name p. 92

View Name p. 92

View Name p. 92

Synonym Name p. 92

Synonym Name p. 92

Procedure Name p. 92

Procedure Name p. 92

Argument p. 78

Argument p. 78
INSERT INTO

- Argument
- SPL parameter name =
- SELECT Statement (singleton)
- VALUES Clause
- VALUES ( Named Value
  - Constant Expression
  - Record Reference
  - Record Reference
  - Record Reference
  - NULL
  - Record Reference
  - Record Reference
  - Record Reference
  - Record Reference
  - THRU
  - member
  - member
  - member
  - member
  - member

- SPL Expression
- p. 112
- p. 97
- p. 104
- p. 103
- p. 82
Table-Level Privileges

- ALL
- PRIVILEGES
  - INSERT
  - DELETE
  - SELECT
  - UPDATE
  - INDEX
  - ALTER
  - REFERENCES

Database-Level Privileges

- CONNECT
- RESOURCE
- DBA

ROLLBACK WORK

SE
+ 

ROLLFORWARD DATABASE

Database Name p. 93
SELECT

FROM Clause p. 83

UNION

UNION ALL

SELECT

INTO TEMP Clause p. 86

INTO Clause p. 83

GROUP BY Clause p. 85

HAVING Clause p. 85

WHERE Clause p. 84

ORDER BY Clause p. 86

SELECT Clause

SELECT Item p. 83

DISTINCT

ALL

UNIQUE
SELECT

Join

Relational Operator

column name

column name

Table Name p. 92
View Name p. 92
Synonym Name p. 92

Table Name p. 92
View Name p. 92
Synonym Name p. 92

GROUP BY

GROUP BY

Table Name p. 92
View Name p. 92
Synonym Name p. 92

HAVING

Condition p. 91

HAVING Clause

GROUP BY Clause

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
ORDER BY Clause

ORDER BY

column name

ASC

DESC

Table Name

p. 92

table alias

View Name

p. 92

Synonym Name

p. 92

select number

display label

rowid

INTO TEMP Clause

INTO TEMP temp table name

WITH NO LOG

6.0

SET CONSTRAINTS

ALL

IMMEDIATE

DEFERRED

Constraint Name

p. 92

6.0

SET DEBUG FILE TO

" filename "

Named Value

p. 103

character expression

WITH APPEND
UPDATE STATISTICS

FOR PROCEDURE

Procedure Name

FOR TABLE

Table Specification

DROP DISTRIBUTIONS

LOW

FOR TABLE

Table Specification

MEDIUM

FOR TABLE

Table Specification

RESOLUTION percent

HIGH

FOR TABLE

Table Specification

RESOLUTION percent

Table Specification

Table Name

p. 92

Synonym Name

p. 92

WHENEVER

NOT FOUND

SQLERROR

ERROR

ANY

WARNING

SQLWARNING

CONTINUE

GOTO

label

GO TO

'label'

STOP

CALL function name

6.0
SQL Segments

Condition

AND

OR

NOT

Comparison Condition

IN Subquery  p. 92

EXISTS Subquery  p. 92

ALL/ANY/SOME Subquery  p. 92

Comparison Condition

SQL Expression  p. 96

Relational Operator  p. 104

SQL Expression  p. 96

SQL Expression  p. 96

BETWEEN

NOT

SQL Expression  p. 96

AND

SQL Expression  p. 96

NOT

SQL Expression  p. 96

IN

Constant Expression  p. 97

NOT

NULL

IS

NOT

Column Expression  p. 96

LIKE

ESCAPE "character"

Quoted String  p. 104

Named Value  p. 103

Column Expression  p. 96

Basics
4GL
Forms
Reports
SQL
SQLCA
Debugger
Variables
Keys
For SE engines, database identifiers can have up to ten characters in UNIX.

When the identifier for a database is also the name of a 4GL variable, the compiler uses the variable. To override this compiler action, quote the database identifier.
Character Data Type

CHAR
- CHARACTER
  - NCHAR
    - VARCHAR
      - NLS
        - 6.0
      - OL
    - +
  - 6.0
  - NLS

Number Data Type

INTEGER
- INT
- SERIAL
  - +
  - ( start )
  - ( size )
  - ( precision )
  - ( float precision )
  - SMALLINT
- FLOOR
- DOUBLE PRECISION
- SMALLFLOAT
- REAL
- DECIMAL
- DEC
- NUMERIC
  - +
  - ( precision )
  - ( scale )
  - MONEY
    - +
    - ( precision )
    - ( scale )
When you refer to a column whose name is identical to that of a variable, you must prefix the column name with an @ symbol; otherwise the 4GL compiler treats it as a variable.
Algebraic Functions

**ROUND**

- **SQL Expression**
  - p. 96

- Rounding factor

**TRUNC**

- **SQL Expression**
  - p. 96

- Truncating factor

**ABS**

- Numeric expression

**MOD**

- Dividend, divisor

**POW**

- Base, exponent

**ROOT**

- Radicand

**SQRT**

- Integer expression

---

**DBINFO Function**

**DBINFO**

- **SQL**
  - 'DBSPACE'

- Tblspace num

- Expression

- 'sqlca.sqlerrd1'

- 'sqlca.sqlerrd2'

---

**Exponential and Logarithmic Functions**

**EXP**

- Float expression

**LOGN**

- Float expression

**LOG10**

- Float expression
It is important to recognize that Stored Procedure Language (SPL) statements are not part of 4GL. This means that you cannot include these statements within a 4GL program. Doing so causes compile errors.

To create a stored procedure from a 4GL program, do the following:

1. Put the text of the CREATE PROCEDURE statement in a file.
2. Use a PREPARE statement to prepare a CREATE PROCEDURE FROM statement that refers to the text file created in Step 1.
3. Use an EXECUTE statement to execute the prepared statement, which then compiles the stored procedure.

Refer to the Informix Guide to SQL: Reference, Version 6.0 for a full description of the CREATE PROCEDURE statement.

You may explicitly invoke stored procedures from within your 4GL program by preparing and executing the following SQL statements:

- CREATE PROCEDURE FROM
- DROP PROCEDURE
- EXECUTE PROCEDURE
- GRANT
- INSERT INTO
- REVOKE
- SET DEBUG FILE TO
- UPDATE STATISTICS

Refer to the Informix Guide to SQL: Reference, Version 6.0 for a description of working with dynamic SQL.

Also, you may implicitly invoke a stored procedure through a reference to that procedure within the context of an SQL expression. For example, the reference to `avg_price()` in the following SELECT statement implicitly invokes the stored procedure having the name `avg_price`:

```sql
select
    manu_code, unit_price,
    (avg_price(1) - unit_price) variance
from stock
where stock_num = 1
```

Such implicit references to stored procedures do not require the statement to be prepared.
LET

variable name = Procedure Name p. 92

, called variable = SPL Expression p. 112

SPL Expression p. 112

ON EXCEPTION

IN (error number)

SET SQL error variable, ISAM error variable, error data variable

Statement Block p. 108

END EXCEPTION

WITH RESUME

RAISE EXCEPTION SQL error

, ISAM error, error text

RETURN

SPL Expression p. 112

WITH RESUME

SYSTEM "character expression" "character variable"
SQLCA Record
DEFINE SQLCA RECORD
  SQLCODE INTEGER,
  SQLERRM CHAR(71),
  SQLERRP CHAR(8),
  SQLERRD ARRAY [6] OF INTEGER,
  SQLAWARN CHAR (8)
END RECORD

Summary of fields:

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Details of Statement Execution</th>
<th>Special Conditions</th>
</tr>
</thead>
</table>

SQLCODE indicates the result of executing an SQL statement. It is set as follows:

- To zero for a successful execution of most statements.
- To NOTFOUND (defined as 100) for a successfully executed query that returns zero rows or for a FETCH that seeks beyond the end of an active set. (However, in an ANSI-compliant database, when an INSERT INTO/SELECT statement or a DELETE, UPDATE, or SELECT INTO TEMP statement fails to access any rows, the value of SQLCA.SQLCODE is set to NOTFOUND rather than 0.)
- To a negative value for an unsuccessful execution.

INFORMIX-4GL sets the global variable STATUS equal to SQLCODE after each SQL statement. However, any subsequent 4GL statement can reset STATUS.

SQLERRM contains parameters for the error message.

SQLERRP is reserved for future use.

SQLERRD is an array of six integers:

- SQLERRD[1] is the estimated number of rows returned.
- SQLERRD[2] is the SERIAL value returned or an error code.
- SQLERRD[3] is the number of rows processed.
- SQLERRD[4] is a weighted sum of disk accesses and total rows processed, the estimated CPU cost of the query.
- SQLERRD[5] is the offset of error into the SQL statement.
- SQLERRD[6] is the rowid of the last row processed.
SQLWARN is a character string of length eight whose individual characters signal various warning conditions (as opposed to errors) following the execution of an SQL statement. The characters are blank if no problems or exceptional conditions are detected.

SQLWARN[1] is set to W if one or more of the other warning characters has been set to W. If SQLWARN[1] is blank, you do not have to check the remaining warning characters.

SQLWARN[2] is set to W if one or more data items were truncated to fit into a character variable or if a DATABASE statement selected a database with transactions.

SQLWARN[3] is set to W if an aggregate function (SUM, AVG, MAX, or MIN) encountered a null value in its evaluation or if a DATABASE statement selected an ANSI-compliant database.

SQLWARN[4] is set to W if a DATABASE statement selected an INFORMIX-OnLine Dynamic Server<Default ¶ Fo> database or when the number of items in the select-list of a SELECT clause is not the same as the number of program variables in the INTO clause. In the latter case, the number of values INFORMIX-4GL returns is the smaller of these two numbers.

SQLWARN[5] is set to W if float-to-decimal conversion is used.

SQLWARN[6] is set to W when your program executes an Informix extension to ANSI-compliant standard syntax and the DBANSIWARN environment variable is set or the -ansi option is specified.

SQLWARN[7] is reserved for future use.

SQLWARN[8] is reserved for future use.
Interactive Debugger Commands
The following table lists the **Debugger** commands, options, and accelerators.

<table>
<thead>
<tr>
<th>Command</th>
<th>Option</th>
<th>Shortest Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>ALIAS</td>
<td>al</td>
<td></td>
</tr>
<tr>
<td>APPLICATION</td>
<td>DEVICE</td>
<td>ap</td>
</tr>
<tr>
<td>BREAK</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>ca</td>
<td></td>
</tr>
<tr>
<td>CLEANUP</td>
<td>cl</td>
<td></td>
</tr>
<tr>
<td>CONTINUE</td>
<td>co</td>
<td></td>
</tr>
<tr>
<td>DATABASE</td>
<td>da</td>
<td></td>
</tr>
<tr>
<td>DISABLE</td>
<td>di</td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td>du</td>
<td></td>
</tr>
<tr>
<td>ENABLE</td>
<td>en</td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>ex</td>
<td></td>
</tr>
<tr>
<td>FUNCTIONS</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>GROW</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>HELP</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>LET</td>
<td>le</td>
<td></td>
</tr>
<tr>
<td>LIST</td>
<td>li</td>
<td></td>
</tr>
<tr>
<td>NOBREAK</td>
<td>nob</td>
<td></td>
</tr>
<tr>
<td>NOTRACE</td>
<td>not</td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>READ</td>
<td>re</td>
<td></td>
</tr>
<tr>
<td>RUN</td>
<td>ru</td>
<td></td>
</tr>
<tr>
<td>STEP</td>
<td>s</td>
<td></td>
</tr>
<tr>
<td>TIME DELAY</td>
<td>SOURCE COMMAND</td>
<td>ti</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ti s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ti c</td>
</tr>
</tbody>
</table>
After setting a breakpoint (or tracepoint), the Output text region displays:

```
(2) break show_menu:91 [customer.4gl]
scope function: show_menu
checkpoint scope
```

To: Type:
- Escape ! command
- Interrupt CONTROL-D or Del key
- Redraw CONTROL-R
- Screen CONTROL-P
- Toggle CONTROL-T
- Search for characters / pattern
- ? pattern
These built-in operators can be used in a boolean expression:

- CURRENT
- DATE()
- DATETIME()
- DAY()
- ENTEND()
- INTERVAL()
- MDY()
- MONTH()
- TODAY
- UNITS
- WEEKDAY()
- YEAR()
To select a remote database, use one of the following formats:

Remote Database Engine                                      Format of database name
INFORMIX-OnLine Dynamic Server                              database@servername

or                                                                //servername/database

INFORMIX-SE                                                      //servername/path/database

Checkpoint Identification p. 126
Scope List p. 128
Output Redirection p. 127
Checkpoint Identification p. 126

CLEANUP 121
A pattern is a string of no more than 50 characters and blanks or up to 80 characters if enclosed in quotation marks.

You can use the following wildcard characters within the pattern:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>any single character</td>
</tr>
<tr>
<td>*</td>
<td>zero or more characters</td>
</tr>
<tr>
<td>[characters]</td>
<td>one or more unseparated characters</td>
</tr>
<tr>
<td>[character-character]</td>
<td>characters within the range in ASCII collating sequence</td>
</tr>
</tbody>
</table>
These built-in operators can be used within a 4GL expression:

- `CURRENT`
- `DATE()`
- `DATETIME()`
- `DAY()`
- `MONTH()`
- `YEAR()`
- `ENTEND()`
- `INTERVAL()`
- `MDY()`
- `MONTH()`
- `WEEKDAY()`

To pass arguments to the p-code runner, include them on the Debugger command line.

Example:

```
PROGRAM = "program"
```

Filename:

```
filename.4db
```

Argument:

```
number INTO NOBREAK
```
A reference name must:

- Be unique among other checkpoint names.
- Begin with an alphabetic character.
- Contain only letters, numbers, or the underscore ( _) character.
<table>
<thead>
<tr>
<th>Type</th>
<th>Valid Entities</th>
<th>Triggered When</th>
</tr>
</thead>
<tbody>
<tr>
<td>statement</td>
<td>executable statements within 4GL functions</td>
<td>Execution completes previous statement.</td>
</tr>
<tr>
<td>variable</td>
<td>†active variables</td>
<td>Variable is assigned a different value.</td>
</tr>
<tr>
<td>function</td>
<td>4GL functions (including MAIN), 4GL reports, ESQL/C functions, C functions</td>
<td>Function is called.</td>
</tr>
</tbody>
</table>

†You cannot set a checkpoint on an entire array or record structure.
Environment
Variables
### Environment Variables

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBANSIWARN</td>
<td></td>
<td>D-8</td>
</tr>
<tr>
<td>DBDATE</td>
<td></td>
<td>D-9</td>
</tr>
<tr>
<td>DBDELCOMMITER</td>
<td></td>
<td>D-11</td>
</tr>
<tr>
<td>DBEDIT</td>
<td></td>
<td>D-11</td>
</tr>
<tr>
<td>DBFORMAT</td>
<td></td>
<td>D-14</td>
</tr>
<tr>
<td>DLANGLES</td>
<td></td>
<td>D-18</td>
</tr>
<tr>
<td>DBMONEY</td>
<td></td>
<td>D-21</td>
</tr>
<tr>
<td>DBPATH</td>
<td></td>
<td>D-23</td>
</tr>
<tr>
<td>DBPRINT</td>
<td></td>
<td>D-26</td>
</tr>
<tr>
<td>DBREMOTEEXEC</td>
<td><strong>OnLine only</strong></td>
<td>D-27</td>
</tr>
<tr>
<td>DBSPACEEXEC</td>
<td><strong>OnLine only</strong></td>
<td>D-28</td>
</tr>
<tr>
<td>DBEXEC</td>
<td><strong>SE only</strong></td>
<td>D-29</td>
</tr>
<tr>
<td>DBUPSPACE</td>
<td></td>
<td>D-29</td>
</tr>
<tr>
<td>ENVIRORE</td>
<td></td>
<td>D-30</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td></td>
<td>D-30</td>
</tr>
<tr>
<td>INFORMIXCONTIME</td>
<td></td>
<td>D-31</td>
</tr>
<tr>
<td>INFORMIXDIR</td>
<td></td>
<td>D-32</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td></td>
<td>D-33</td>
</tr>
<tr>
<td>INFORMIXSHMBASE</td>
<td><strong>OnLine only</strong></td>
<td>D-33</td>
</tr>
<tr>
<td>INFORMIXSTACKSIZE</td>
<td><strong>OnLine only</strong></td>
<td>D-34</td>
</tr>
<tr>
<td>INFORMIXTERM</td>
<td></td>
<td>D-34</td>
</tr>
<tr>
<td>ONCONFIG</td>
<td><strong>OnLine only</strong></td>
<td>D-36</td>
</tr>
<tr>
<td>PSORT_DBTEMP</td>
<td><strong>OnLine only</strong></td>
<td>D-36</td>
</tr>
<tr>
<td>PSORT_NPROCS</td>
<td><strong>OnLine only</strong></td>
<td>D-37</td>
</tr>
<tr>
<td>SQLEXEC</td>
<td></td>
<td>D-38</td>
</tr>
<tr>
<td>SQLRM</td>
<td></td>
<td>D-38</td>
</tr>
<tr>
<td>SQLRMDIR</td>
<td>SQL APIs only</td>
<td>D-39</td>
</tr>
</tbody>
</table>
### NLS Environment Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLCHAR</td>
<td>E-18</td>
</tr>
<tr>
<td>DBAPICODE</td>
<td>E-23</td>
</tr>
<tr>
<td>DBNLS</td>
<td>E-16</td>
</tr>
<tr>
<td>LANG</td>
<td>E-25</td>
</tr>
<tr>
<td>LC_COLLATE</td>
<td>E-27</td>
</tr>
<tr>
<td>LC_CTYPE</td>
<td>E-29</td>
</tr>
<tr>
<td>LC_MONETARY</td>
<td>E-31</td>
</tr>
<tr>
<td>LC_NUMERIC</td>
<td>E-35</td>
</tr>
</tbody>
</table>

### UNIX Environment Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH</td>
<td>D-40</td>
</tr>
<tr>
<td>TERM</td>
<td>D-41</td>
</tr>
<tr>
<td>TERMCAP</td>
<td>D-41</td>
</tr>
<tr>
<td>TERMINFO</td>
<td>D-42</td>
</tr>
</tbody>
</table>
Default Key Assignments
Logical command keys at runtime and their default assignments:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Purpose of Key</th>
<th>Default Keystroke</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Selects the current menu option in a MENU statement; terminates input during</td>
<td>Escape</td>
</tr>
<tr>
<td></td>
<td>CONSTRUCT, INPUT and INPUT ARRAY; terminates DISPLAY ARRAY.</td>
<td></td>
</tr>
<tr>
<td>Interrupt</td>
<td>Represents the external interrupt signal; available when interrupts are</td>
<td>CONTROL-C</td>
</tr>
<tr>
<td></td>
<td>deferred with the DEFER statement.</td>
<td></td>
</tr>
<tr>
<td>Insert</td>
<td>Requests insertion of a new line during INPUT ARRAY, starting execution of a</td>
<td>F1</td>
</tr>
<tr>
<td></td>
<td>BEFORE INSERT block.</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>Requests deletion of the current line during INPUT ARRAY, starting execution</td>
<td>F2</td>
</tr>
<tr>
<td></td>
<td>of a BEFORE DELETE block.</td>
<td></td>
</tr>
<tr>
<td>Next</td>
<td>Causes scrolling to the next page (group of lines) during DISPLAY ARRAY and</td>
<td>F3</td>
</tr>
<tr>
<td></td>
<td>INPUT ARRAY.</td>
<td></td>
</tr>
<tr>
<td>Previous</td>
<td>Causes scrolling to the previous page (group of lines) during DISPLAY ARRAY</td>
<td>F4</td>
</tr>
<tr>
<td></td>
<td>and INPUT ARRAY.</td>
<td></td>
</tr>
<tr>
<td>Help</td>
<td>Starts the display of the specified HELP message from the current help file.</td>
<td>CONTROL-W</td>
</tr>
<tr>
<td>Quit</td>
<td>Terminates the program unless DEFER QUIT is specified.</td>
<td>CONTROL-\</td>
</tr>
</tbody>
</table>

Effect of special keys on interactive 4GL statements and within 4GL menus:

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Use in CONSTRUCT, INPUT, and INPUT ARRAY</th>
<th>Use in MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL-A</td>
<td>Switches between overtype and insert</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td>modes.</td>
<td></td>
</tr>
<tr>
<td>CONTROL-D</td>
<td>Deletes from the cursor to the end of</td>
<td>None.</td>
</tr>
<tr>
<td></td>
<td>the field.</td>
<td></td>
</tr>
<tr>
<td>CONTROL-H</td>
<td>During text entry, moves the cursor left</td>
<td>Moves highlight to next</td>
</tr>
<tr>
<td>(backspace)</td>
<td>one position (nondestructive backspace).</td>
<td>option left.</td>
</tr>
<tr>
<td>CONTROL-I</td>
<td>Cursor moves to next field; except in a</td>
<td>None.</td>
</tr>
<tr>
<td>or TAB</td>
<td>WORDWRAP field, inserts a tab or skips to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a tab depending on mode.</td>
<td></td>
</tr>
<tr>
<td>CONTROL-J</td>
<td>Cursor moves to next field; except in a</td>
<td>Moves the highlight to the</td>
</tr>
<tr>
<td>(Linefeed)</td>
<td>WORDWRAP field, inserts a newline or</td>
<td>next option right.</td>
</tr>
<tr>
<td></td>
<td>moves down one line depending on mode.</td>
<td></td>
</tr>
<tr>
<td>CONTROL-L</td>
<td>During text entry, moves the cursor right</td>
<td>Moves the highlight to the</td>
</tr>
<tr>
<td></td>
<td>one position.</td>
<td>next option right.</td>
</tr>
</tbody>
</table>
Default Key Assignments

<table>
<thead>
<tr>
<th>Key Name</th>
<th>Use in CONSTRUCT, INPUT, and INPUT ARRAY</th>
<th>Use in MENU</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROL-M or RETURN</td>
<td>Completes entry of the current field. Cursor moves to next field if any; else same as Accept.</td>
<td>Accepts the option that is currently highlighted.</td>
</tr>
<tr>
<td>CONTROL-N</td>
<td>Cursor moves to beginning of current field.</td>
<td>None.</td>
</tr>
<tr>
<td>CONTROL-R</td>
<td>Causes the screen to be redrawn.</td>
<td>Causes the screen to be redrawn.</td>
</tr>
<tr>
<td>CONTROL-X</td>
<td>Deletes the character under the cursor.</td>
<td>None.</td>
</tr>
<tr>
<td>Left Arrow</td>
<td>Same as Backspace.</td>
<td>Same as CONTROL-L.</td>
</tr>
<tr>
<td>Right Arrow</td>
<td>Same as CONTROL-L.</td>
<td>Moves the highlight to the next option left.</td>
</tr>
<tr>
<td>Up Arrow</td>
<td>Moves to previous field; except in a WORDWRAP field moves up one line in field and in an INPUT ARRAY moves to the corresponding field in the previous row.</td>
<td>Moves the highlight to the next option right.</td>
</tr>
<tr>
<td>Down Arrow</td>
<td>Moves to next field; except in a WORDWRAP field moves down one line in field and in an INPUT ARRAY moves to the corresponding field in the next row.</td>
<td></td>
</tr>
</tbody>
</table>