ORACLE
PEOPLESOFT

Oracle's PeopleTools PeopleBook

PeopleTools 8.52: SQR Language Reference for PeopleSoft

June 2013

ORACLE

PeopleTools 8.52: SQR Language Reference for PeopleSoft
SKU pt8.52tsql-b0613

Trademark Notice

License Restrictions Warranty/Consequential Damages Disclaimer

Warranty Disclaimer

Restricted Rights Notice

Hazardous Applications Notice

Third Party Content, Products, and Services Disclaimer

Contents

Appendix B

Oracle's PeopleSoft Enterprise SQR Language Reference for PeopleSoft Preface

This PeopleBook is a language reference for Structured Query Reports (SQR) for PeopleSoft.

Oracle's PeopleSoft Enterprise SQR Language Reference for PeopleSoft

Oracle's PeopleSoft Enterprise SQR Language Reference for PeopleSoft PeopleBook describes and demonstrates the structure, command set, and syntax of the SQR language. It also provides a list of the sample SQR programs, an overview of the SQR initialization file, and a detailed list of SQR messages.

This reference PeopleBook is intended for SQR and SQL developers who must report on data from a wide range of enterprise data sources. Before using this reference, familiarize yourself with the data sources from which you are reporting and the connectivity between those data sources and your operating system.

See Also

PeopleTools 8.52: SQR for PeopleSoft Developers, "Getting Started with SQR for PeopleSoft"

PeopleBooks and the PeopleSoft Online Library

A companion PeopleBook called *PeopleBooks and the PeopleSoft Online Library* contains general information, including:

- Understanding the PeopleSoft online library and related documentation.
- How to send PeopleSoft documentation comments and suggestions to Oracle.
- How to access hosted PeopleBooks, downloadable HTML PeopleBooks, and downloadable PDF PeopleBooks as well as documentation updates.
- Understanding PeopleBook structure.
- Typographical conventions and visual cues used in PeopleBooks.
- ISO country codes and currency codes.
- PeopleBooks that are common across multiple applications.
- Common elements used in PeopleBooks.
- Navigating the PeopleBooks interface and searching the PeopleSoft online library.

- Displaying and printing screen shots and graphics in PeopleBooks.
- How to manage the locally installed PeopleSoft online library, including web site folders.
- Understanding documentation integration and how to integrate customized documentation into the library.
- Application abbreviations found in application fields.

You can find *PeopleBooks and the PeopleSoft Online Library* in the online PeopleBooks Library for your PeopleTools release.

Chapter 1

Understanding SQR for PeopleSoft

This chapter discusses:

- SQR for PeopleSoft tools.
- The Structured Query Report (SQR) language.
- SQR command line.
- SQR data elements.
- Sample reports.

SQR for PeopleSoft Tools

SQR for PeopleSoft is a powerful enterprise reporting system that provides direct access to multiple data sources. The SQR for PeopleSoft tools enable you to create clear, professional reports from complex arrays of information systems.

This PeopleBook describes the following SQR for PeopleSoft tools:

- The SQR language, which is a flexible, fourth-generation reporting language with a lexicon of more than 110 commands.

  The procedural design of SQR enables you to easily develop, implement, and distribute complex reports.
- SQR Execute, which enables you to run previously compiled SQR programs.
- SQR Print, which enables you to configure reports for most printers.
- SQR Samples, a library of SQR programs and output that provides a framework for creating configured reports.

The SQR Language

This section provides an overview of the SQR language and discusses:

- SQR program structure.
- SQR syntax conventions.
- Rules for entering SQR commands.

Understanding the SQR Language

SQR is a specialized programming language for accessing, manipulating, and reporting enterprise data. With SQR, you build complex procedures that perform multiple calls to multiple data sources and implement nested, hierarchical, or object-oriented program logic.

SQR provides several important benefits:

- Flexibility and scalability.
- Comprehensive facilities for combined report and data processing.
- Multiple platform availability.
- Multiple data source compatibility.

With SQR, you design reports by defining the page size, headers, footers, and layout. SQR enables you to generate a variety of output types, such as complex tabular reports, multiple page reports, and form letters. You can display data in columns; produce special formats, such as mailing labels; and create HTML, PDF, or configured output for laser printers and phototypesetters.

The high-level programming capabilities of SQR enable you to add procedural logic and to control data source calls. You can use SQR to write other types of applications, such as those for database manipulation and maintenance, table loading and unloading, and interactive querying and displaying.

SQR Program Structure

SQR for PeopleSoft processes source code from a standard text file and generates a report. The text file containing source code comprises a set of sections that you delimit with BEGIN-section and END-section commands. The following examples show the general structure of SQR:

- The SETUP section describes overall characteristics of the report.

```
BEGIN-SETUP
 {setup commands}...
END-SETUP
```

- The HEADING and FOOTING sections specify what information is printed in the header and footer on each page of the report.

```
BEGIN-HEADING {heading_lines}
 {heading commands}...
END-HEADING
BEGIN-FOOTING {footing_lines}
 {footing commands}...
END-FOOTING
```

- The PROGRAM section runs the procedures in the report.

```
BEGIN-PROGRAM
 {commands}...
END-PROGRAM
```

- The PROCEDURE section performs the tasks to produce the report.

```
BEGIN-PROCEDURE {procedure_name}
 {procedure commands}...
END-PROCEDURE
```

SQR Syntax Conventions

The following table describes the SQR syntax conventions:

Syntax Convention	*Description*
{ }	Braces enclose required items.
[ ]	Square brackets enclose optional items.
...	Ellipses indicate that the preceding parameter can be repeated.
|	A vertical bar separates alternatives inside brackets, braces, or parentheses.
!	An exclamation point begins a single-line comment that extends to the end of the line. Each comment line must begin with an exclamation point.
'	A single quote starts and ends a literal text constant or any argument with more than one word. **Important!** If you are copying code directly from the sample programs, change the slanted quotes to regular quotes as shown here, otherwise you will receive an error message.
,	A comma separates multiple arguments.
( )	Parentheses must enclose an argument or element.
UPPERCASE	SQR commands and arguments are uppercase within the text, but lowercase in the sample programs. (Note that these commands are *not* case-sensitive.)
Variable	Information and values that you must supply appear in *variable* style.

Syntax Convention	*Description*
hyphen versus underscore	Many SQR commands, such as BEGIN-PROGRAM, contain a hyphen, whereas procedure and variable names contain an underscore. Procedure and variable names can contain either hyphens or underscores, but you should use underscores in procedure and variable names to distinguish them from SQR commands. This practice also prevents confusion when you mix variable names and numbers in an expression in which hyphens could be mistaken for minus signs.

Rules for Entering SQR Commands

Use these command rules as you develop SQR programs:

- You can enter SQR commands in either uppercase or lowercase; they are *not* case-sensitive.

  Many SQR programmers use uppercase for SQR commands, but SQR ignores case as it compiles source code.
- You must separate command names and arguments by at least one space or tab character.
- You must begin each command on a new line; however, you can develop commands that extend beyond one line.
- You can break a line in any position between words *except* inside a quoted string.
- You can use a hyphen (-) at the end of a line to indicate that it continues on the next line; however, SQR ignores hyphens and carriage returns in commands.
- You must begin each comment line with an exclamation point (!).

Note. To display the exclamation point (!) or single quote (') symbols in a report, type the symbol twice to indicate that it is text. For example, DON'T is typed DON''T. This rule does not apply in the document paragraph of form-letter reports.

SQR Command Line

SQR for PeopleSoft comprises SQR, SQR Execute, and SQR Print. Each has a command-line interface.

To begin running SQR, enter the following command:

```
SQR [program][connectivity][flags...][args...][@file...]
```

Note. The executable name for SQR is SQR (SQRW for Microsoft Windows). The executable name for SQR Execute is SQRT (SQRWT for Microsoft Windows). The executable for SQR Print is SQRP (SQRWP for Microsoft Windows).

See Chapter 4, "Invoking SQR Execute," page 269 and Chapter 5, "Using SQR Print," page 277.

SQR Command-Line Arguments

The following table describes the SQR command-line arguments:

Argument	*Description*
program	Name of the text file containing the source code. The default file type or extension is .sqr. If you enter this value as a question mark (?) or omit it, SQR for PeopleSoft prompts you for the report program name. On UNIX/Linux-based systems, if your shell uses the question mark as a wildcard character, you must precede it with a backslash (\).

connectivity	Information that SQR for PeopleSoft needs to connect to the database. If you enter this value as a question mark (?) or omit it, SQR for PeopleSoft prompts you for the information.
	IBM DB2: Subsystem name and Structured Query Language (SQL) authorization ID.
	`Ssname/SQLid`
	Microsoft Windows or IBM DB2: The name of the database, your username, and the password for the database.
	`[Database] / [Username] / [Password]`
	Informix: Name of the database.
	`Database[/username/password]`
	Open Database Connectivity (ODBC): The name that you give to the ODBC driver when you set up the driver, and your username and password for the database. This port has been certified against IBM DB2 and Microsoft SQL Server.
	`Data_Source_Name/[Username]/[Password⇒ ]`
	Oracle: Your username and password for the database, and an optional connection string for the database (for example, @sales.2cme.com).
	`[Username]/[Password[@Database]]`
	Sybase: Your username and password for the database.
	`Username/[Password]`

flags	See Chapter 1, "Understanding SQR for PeopleSoft," SQR Command-Line Flags, page 6.

Argument	*Description*
args...	Arguments that SQR for PeopleSoft uses while the program is running. The ASK and INPUT commands use these arguments rather than prompting the user. You must enter arguments on the command line in the sequence that the program expects—first all ASK arguments, in order, followed by INPUT arguments.
@file...	File containing program arguments—one argument per line. Arguments that are listed in the file are processed one at a time: first all ASK arguments, in order, followed by INPUT arguments. You can specify the command-line arguments *(program,connectivity,* and *args)* in this file for non-Microsoft Windows platforms.

SQR Command-Line Flags

SQR supports a number of command-line flags. Each flag begins with a hyphen (-). When a flag takes an argument, the argument must follow the flag with no intervening space.

The following table describes the SQR command-line flags:

Flag	*Description*
-A	Appends the output to an existing output file carrying the same name as the source of the output. If the file does not exist, a new one is created. This flag is useful when you want to run the same report more than once but want only a single output file.
-B*nn*	(Oracle, ODBC, and Sybase CT-Lib) Indicates the number of rows to buffer each time SQR for PeopleSoft retrieves data from the database. The default is 10 rows. Regardless of the setting, all rows are retrieved. When used on the command line, -B controls the setting for all BEGIN-SELECT commands. Inside a program, each BEGIN-SELECT command can also have its own -B flag for further optimization.

Flag	*Description*
-BURST:{*xx*}	-BURST:T generates the table of contents file only. -BURST:S generates the report output according to the symbolic table of contents entries that are set in the program with the *level* argument of the TOC-ENTRY command. In -BURST:S[{l}], {l} is the level on which to burst. The -BURST:S setting is equivalent to -BURST:S1. See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Generating and Publishing HTML from an SQR Program," Bursting Reports. **Note.** -BURST:P and -BURST:S require -PRINTER:EH or -PRINTER:HT. The page range selection feature of -BURST:P requires -PRINTER:HT. -BURST:T requires -PRINTER:HT.
-C	(Microsoft Windows) Specifies that the Cancel dialog box appears while the program runs so that you can easily stop the program.
-CB	(Microsoft Windows, Callable SQR) Forces the communication box to use.
-D*nn*	(Non-Microsoft Windows) Displays the report output on the terminal while it is being written to the output file. The value for *nn* is the maximum number of lines to display before pausing. If no number is entered after -D, the display scrolls continuously. **Note.** The printer type must be LP; otherwise, the display is ignored. If the program is producing more than one report, the display is for the first report only.
-DBdatabase	(Sybase) Forces the SQR program to use the specified database, which overrides any USE command in the SQR program.
-DEBUG [*xxx*]	See Chapter 2, "SQR Command Reference," #DEBUG, page 71.
-DNT: {*xx*}	See Chapter 2, "SQR Command Reference," DECLARE-VARIABLE, page 108.

Flag	*Description*
-E[*file*]	Directs error messages to the named file or to the default file program.err. If no errors occur, no file is created.
-EH_APPLETS:*dir*	Specifies the directory location of the enhanced HTML applets. If you include an applet, SQR for PeopleSoft must know where it resides. SQR for PeopleSoft usually checks for the applet in a default directory; the default directory for these applets is IMAGES. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_BQD	Generates a {report}.bqd file from the report data. This flag also associates a query format file (BQD) icon with {report}.bqd in the navigation bar. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_BQD:*file*	Associates the BQD icon with the specified file. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_BROWSER:*xx*	Generates HTML, determines the browser, and displays HTML. When this flag is set to ALL, SQR for PeopleSoft generates frame.html, which contains JavaScript to determine the browser on the user's machine (that is, the person reading the report, not the person writing it). When this flag is set to BASIC, SQR for PeopleSoft generates HTML that is suitable for all browsers. When this flag is set to IE, SQR for PeopleSoft generates HTML that is designed for Microsoft Internet Explorer. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_CSV	Generates a {report}.csv file from the report data. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.

Flag	*Description*
-EH_CSV:*file*	Associates the CSV icon with the specified file. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_CSVONLY	Creates a .csv file, but does not create an HTML file. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_FULLHTML:*xx*	Specifies the level of the generated enhanced HTML code. This can be 30, 32, or 40. **Note.** For upward compatibility, a value of TRUE is equivalent to 40 and FALSE is 30.
-EH_Icons:dir	Specifies the directory in which the HTML should find the referenced icons. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_IMAGES:dir	Specifies the directory path for the .gif files that are used by the navigation bar. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_KEEP	Copies (does not move) the files when used in conjunction with -EH_ZIP. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_LANGUAGE:*xx*	Sets the language that is used for the HTML navigation bar. You can specify English, French, German, Portuguese, Spanish, Japanese, Simplified Chinese, or Korean. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.

Flag	*Description*
-EH_PDF	Associates a PDF icon with {report}.pdf in the navigation bar. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_Scale:{*nn*}	Sets the scaling factor to a value from 50 to 200. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_XML:*file*	Associates the XML icon with the specified file. **Note.** This flag is applicable only when you specify the -PRINTER:EH or -PRINTER:EP flag.
-EH_ZIP[:file]	Moves the generated files to the specified file or {report}.zip if {file} is not specified. **Note.** This flag is applicable only when you specify the -PRINTER:EH or the -PRINTER:EP flag.

-F[*file* | *directory*]	Overrides the default output file name, program.lis. The default action places the program.lis file in the same directory as the program.sqr file. To use the current directory, specify -F without an argument. To change the name of the output file, specify -F with the new name. If the new name does not specify a directory, the file is created in the current directory. The output file is not created until data is actually printed on the page. If no data is printed, no output file is created.
	
	Specify these file names and directories for different operating systems:
	
	UNIX/Linux
	
	Directory character is /
	
	`-F$HOME/reports/`
	IBM MVS
	
	Directory character is (
	
	`-FDSN:SQR.REPORTS(`

-GPRINT=YES | NO	(IBM MVS) -GPRINT=YES includes control characters in the first column of each record of the SQR report output file. -GPRINT=NO omits the control characters.

Flag	*Description*
-Idir_list	Specifies the list of directories that SQR for PeopleSoft searches when processing the #INCLUDE directive when the include file does not exist in the current directory and no path is specified for the file. The directory names must be separated by commas (,) or semicolons (;). For UNIX/Linux-based systems, if your shell uses semicolons as command delimiters, you must precede each semicolon with a backslash (\). Always append the directory character to the end of each directory name. See the -F flag for a list of directory characters, sorted by operating system.
-ID	(Non-Microsoft Windows) Displays the copyright banner on the console.
-KEEP	See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Printing with SQR."
-LL{s | d}{c | i}	Specifies the following LOAD-LOOKUP values: • s: SQR • d: DB • c: Case sensitive • i: Case insensitive See Chapter 2, "SQR Command Reference," LOAD-LOOKUP, page 174.
-M*file*	Defines a startup file containing sizes to assign to internal parameters—extremely small, large, or complex reports. M*files* are text files that have individual switches in the INI files that are unique to a run.
-NOLIS	Prevents the creation of .lis files, creating .spfs instead.
-O[*file*]	Directs log messages to the specified file or to program.log if no file is specified. By default, the sqr.log file is created in the current working directory.

Flag	*Description*
-olim	Displays the SQR resources that are used by the SQR report. • Limit is the programmatic limit, the maximum amount of memory that can be allocated to the variable. • Defined is the value defined in the pssqr.ini file; this is a user-defined value. • Actual is the actual amount of memory/space used by the variable at runtime.
-P	(IBM MVS) Suppresses printer control characters from column 1.
-PB	(Informix) Causes column data to preserve trailing blanks.

Flag	*Description*

-PRINTER:*xx*	Uses printer type *xx* when creating output files. The *xx* represents:
	EH: Enhanced HTML
	`-PRINTER:EH`
	EP: Enhanced HTML or PDF
	`-PRINTER:EP`
	HP: HP LaserJet
	`-PRINTER:HP`
	HT: HTML 2.0
	`-PRINTER:HT`
	LP: Line printer
	`-PRINTER:LP`
	PD: PDF
	`-PRINTER:PD`
	PS: PostScript
	`-PRINTER:PS`
	WP: Microsoft Windows
	`-PRINTER:WP`
	LP, HP, and PS produce .lis files. EH and HT produce .htm files. HT produces version 2.0 HTML files with the report content inside <PRE></PRE> tags. EH produces reports in which content is fully formatted with version 3.0 or 3.2 HTML tags. On Microsoft Windows systems, WP sends the output to the default Microsoft Windows printer. To specify a Microsoft Windows printer that is not the default, enter -PRINTER:WP:{printer name}, where {printer name} is the name that is assigned to your printer. For example, to send output to a Microsoft Windows printer named New Printer, use -PRINTER:WP:NewPrinter. If the printer name has spaces, enclose the entire argument in quotes. To create an .spf file also, use -KEEP.

-RS	Saves the program in a runtime file. The program is scanned, compiled, and checked for correct syntax. Queries are validated and compiled. The executable version is saved in a file named program.sqt. **Note.** SQR for PeopleSoft does not prompt ASK variables after compilation.

Flag	*Description*
-RT	Uses the runtime file that is saved with the -RS flag. This skips all syntax and query checking, and processing begins immediately. **Note.** SQR for PeopleSoft does not prompt ASK variables after compilation.
-S	Requests that the status of all cursors be displayed at the end of the report run. Status includes the text of each SQL statement, the number of times each was compiled and run, and the total number of rows that were selected. The output appears directly on the screen. This information can be used for debugging SQL statements, enhancing performance, and tuning.
-T*nn*	Specifies that you want to test your report for *nn* pages. To save time during testing, SQR for PeopleSoft ignores all ORDER BY clauses in SELECT statements. If the program is producing more than one report, SQR for PeopleSoft stops after producing the specified number of pages defined for the first report.

Flag	*Description*
-T{B}	(Microsoft Windows, IBM DB2, Sybase CT-Lib, and ODBC) Trims trailing blanks from database character columns. If the TB flag is set in DB2 database environment, SQR trims the all-blanks fields (fields that contain only spaces) to NULL values in the SQR buffers. Using the TB flag on IBM MVS and DB2 has no effect. IBM MVS and DB2 prevent SQR for PeopleSoft from removing trailing blanks from database character columns. **Note.** The -TB flag has an effect only if SQR is connecting to a DB2, Sybase CT-Lib, or ODBC (MSS) database. Confusingly, the behavior of the -TB command-line flag varies depending on the platform. If you are using one of the previously mentioned databases and are running SQR on z/OS, the -TB flag will act in the following way: If you do not use the -TB flag, trailing blanks are trimmed. If you do use the -TB flag, trailing blanks are not trimmed. If you are running SQR on any other platform, the behavior of -TB is the opposite. That is: If you do not use the -TB flag, trailing blanks are not trimmed. If you do use the -TB flag, trailing blanks are trimmed.
-T{Z}	(IBM MVS and DB2) Prevents SQR for PeopleSoft from removing trailing zeros from the decimal portion of numeric columns.
-Vserver	(Sybase) Uses the named server.
-XB	(Non-Microsoft Windows) Suppresses the SQR banner and the *SQR.... End of Run* message.
-XC	(Callable SQR) Suppresses the database commit when the report has finished running.
-XCB	(Microsoft Windows) Does not use the communication box. Requests for input are made in Microsoft Windows dialog boxes.

Flag	*Description*
-XI	Prevents user interaction during a program run. If an ASK or INPUT command requires user input, an error message is generated and the program ends.
-XL	Prevents SQR for PeopleSoft from signing in to the database. Programs that you run in this mode cannot contain SQL statements. -XL enables you to run SQR for PeopleSoft without accessing the database. You still must supply at least an empty slash on the command line as a placeholder for the connectivity information. For example: `sqr myprog / -xl`
-XLFF	Prevents a trailing form feed.
-XMB	(Microsoft Windows) Disables the error message display so that you can run a program without interruption from error message boxes. Error messages are sent to an .err file. See the -E flag.
-XNAV	Prevents SQR for PeopleSoft from creating the navigation bar in .htm files that are generated with -PRINTER:HT. This occurs when only a single .htm file is produced. Multiple .htm files that are generated from a single report always contain the navigation bar.
-XP	(Sybase) Prevents SQR for PeopleSoft from creating temporary stored procedures. See Chapter 2, "SQR Command Reference," BEGIN-SELECT, page 51.
-XTB	Preserves the trailing blanks in a .lis file.
-XTOC	Suppresses the table of contents for the report. This flag is ignored when you specify either -PRINTER:EH or -PRINTER:HT.
-ZIF{*file*}	Sets the full path and name of the SQR initialization file, sqr.ini.
-ZIV	Invokes the SPF Viewer after generating the program.spf file. This flag implicitly invokes the -KEEP flag to create program.spf. In the case of multiple output files, only the first report file is passed to the viewer.

Flag	*Description*
-ZMF{*file*}	Specifies the full path and name of the SQR error message file, sqrerr.dat.

SQR Data Elements

Each SQR data element begins with a special character that denotes the type of data element.

This section discusses:

- Columns
- Variables
- Literals

Columns

Columns are fields that are defined in the database.

The ampersand character (&) begins a database column or expression name. It can be any type of column, such as character, number, or date. Columns that are defined in a query are declared automatically, except for dynamic columns and database or aggregate functions.

Variables

Variables are storage places for text or numbers that you define and manipulate. Variables begin with special characters:

- $ begins a text or date variable.
- # begins a numeric variable.
- % begins a list variable.
- @ begins a variable name for a marker location.

  Marker locations identify positions to begin printing in a BEGIN-DOCUMENT paragraph.

Variable Rules

The following rules govern the use of variables in SQR:

- Variables can be almost any name of almost any length—for example, $state_name or #total_cost.

- Do not use an underscore (_) or colon (:) as the first character of a two-variable name.

  See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Break Logic," Using Hyphens and Underscores.

- Variable names are *not* case-sensitive.

  That is, you can use a name in uppercase on one line and lowercase on the next; both refer to the same variable.

- SQR for PeopleSoft initializes variables to null (text and date) or zero (numeric).
- A command can grow to whatever length the memory of your computer can accommodate.
- Numeric variables can be one of three types: float, integer, or decimal.

  See Chapter 2, "SQR Command Reference," DECLARE-VARIABLE, page 108.

- Variables and columns are known globally throughout a report, except when used in a local procedure (one with arguments or declared with the LOCAL argument), in which case they are known in that procedure only.

  See Chapter 2, "SQR Command Reference," BEGIN-PROCEDURE, page 48.

SQR Reserved Variables

When you create multiple reports, the variables apply to the current report. SQR for PeopleSoft reserves a library of predefined variables for general use.

The following table describes the SQR-reserved variables:

Variable	*Description*
#current-column	Current column on the page.
$current-date	Current date and time on the local machine when SQR for PeopleSoft starts running the process.
#current-line	Current line on the page. This value is the physical line on the page, not the line in the report body. Line numbers are referenced in PRINT and other SQR commands that are used for positioning the data on the page. Optional page headers and footers, which are defined with BEGIN-HEADING and BEGIN-FOOTING commands, have their own line sequences. Line 2 of the heading is different from line 2 of the report body or footing.
#end-file	See Chapter 2, "SQR Command Reference," READ, page 223.

Variable	*Description*
#page-count	Current page number.
#return-status	Value to return to the operating system when SQR for PeopleSoft exits. This can be set in the report. *#return-status* is initialized to the *success* return value for the operating system.
#sql-count	Count of the rows that are affected by a SELECT paragraph (INSERT, UPDATE, or DELETE). This is equivalent to ROWCOUNT in Oracle and Sybase.
$sql-error	Text message from the database explaining an error. This variable is rewritten when a new error is encountered.
#sql-status	The value of *#sql-status* is set whenever a BEGIN-SELECT command is run. Normally this variable is checked from inside an ON-ERROR procedure, so its value describes the error condition (whereas the *$sql-error* variable contains the error message). The actual meaning of *#sql-status* is database-dependent. Therefore, consult the proper database manual to fully interpret its meaning.
$sqr-encoding-console *{sqr-encoding-console}*	Name of encoding for character data that is written to the log file or console.
$sqr-encoding-database *{sqr-encoding-database}*	Character data that is retrieved from and inserted into the database.
$sqr-encoding-file-input *{sqr-encoding-file-input}*	Name of encoding for character data that is read from files that are used with the OPEN command.
$sqr-encoding-file-output *{sqr-encoding-file-output}*	Name of encoding for character data that is written to files that are used with the OPEN command.
$sqr-encoding-report-output *{sqr-encoding-report-output}*	Report that is generated by SQR for PeopleSoft (for example, a .lis file or a PostScript file).
$sqr-encoding-source *{sqr-encoding-source}*	Name of encoding for SQR source files and include files.

Variable	*Description*
$sqr-database *{sqr-database}*	Database type for which SQR was compiled. Values are ORACLE, DB2, ODBC, SYBASE, and INFORMIX.
$sqr-dbcs *{sqr-dbcs}*	Specifies whether SQR for PeopleSoft recognizes double-byte character strings. Values are YES and NO.
$sqr-encoding *{sqr-encoding}*	Name of the default encoding as defined by the ENCODING environment variable when SQR for PeopleSoft is invoked.
$sqr-hostname *{sqr-hostname}*	Name of the computer on which SQR for PeopleSoft is currently running.
$sqr-locale	Name of the current locale that is being used. A plus symbol (+) at the end of the name indicates that an argument that is used in the locale has changed.
#sqr-max-lines	Maximum number of lines, as determined by the layout. When a new report is selected, this variable is automatically updated to reflect the new layout.
#sqr-max-columns	Maximum number of columns, as determined by the layout. When a new report is selected, this variable is automatically updated to reflect the new layout.
#sqr-pid	Process ID of the current SQR process. *#sqr-pid* is unique for each run of SQR. This variable is useful for creating unique, temporary names.
$sqr-platform, *{sqr-platform}*	The hardware or operating system type for which SQR was compiled. Values are MVS, Windows, and UNIX/Linux.
$sqr-program	Name of the SQR process file.
$sqr-ver	Text string that is shown with the -ID flag, SQR version.
$username	Database username that is specified on the command line.

Variable	*Description*
$sqr-report	Name of the report output file. $sqr-report reflects the actual name of the file to use, as specified by the -F flag or NEW-REPORT command.

List Variables

List variables contain an ordered collection of SQR variables and are nonrecursive—that is, you cannot nest lists inside lists.

Indicate list variables with the percent symbol (%). Create list variables with the LET command and a list of variables. For example:

```
let %list1 = list (num_var1|str_var1, num_var2|str_var2,...)
```

Working with list variables includes the following tasks:

- Defining a list variable:

  You can use a list variable to hold multiple rows of information. Before you assign a list variable, define it by using the following syntax:

  ```
  let %listname=list(col_var|num_var|str_var|str_lit|num_lit[,...])
  ```

  or

  ```
  let %listname[num_lit]=list(NUMBER|DATE|TEXT$colname
  |'.colname'[,...])
  ```

- Assigning a list variable:

  Assign a list variable by using the following syntax:

  ```
  let %listname|%listname[num_var|num_lit]=list(col_var|str_var
  |num_var|str_lit|num_lit[,...])
  ```

- Accessing a list variable:

  Access a list variable by using the following syntax:

  ```
  let str_var|num_var=%listname[num_var|num_lit].#colname
  ```

List Variable Arguments

The value between the brackets indicates either the number of rows in the list for the definition case or the row in the list to modify or assign.

If no brackets are present, you do not need to predefine; assign the types based on the given variable types. For multirow lists, the assignment must be compatible with the types that are given in the definition.

A NUMBER field has the same characteristics as an undeclared #var. The underlying storage depends on the contents, and the DEFAULT-NUMERIC setting applies.

The usual SQR rules for variable assignment apply to list access. Assignment is prohibited only between date and numeric types. Assignment of a numeric column to a string variable returns the string representation of the numeric value; assignment of a date variable to a string variable returns the default-edit-mask representation of the date.

Literals

Literals are text or numeric constants:

- A single quote begins and ends a text literal. For example:

  ```
  'Hello'
  ```

- Numerals 0−9 begin numeric literals.

Numerals that include digits with an optional decimal point and leading sign are acceptable numeric literals, for example, -543.21. Numeric literals can also be expressed in scientific form, for example, 1.2E5.

Sample Reports

For an overview of how an SQR report looks, view the sample reports that are stored in the SQR for PeopleSoft directory <PS_HOME>\bin\sqr\<database_platform>\SAMPLE (or SAMPLEW, for Windows). You can modify these reports to meet your needs.

Chapter 2

SQR Command Reference

This chapter describes and demonstrates each command in the SQR lexicon.

SQR Command Overview

The commands in this section follow the conventions listed in the section SQR Syntax Conventions in the previous chapter and use the abbreviations described in the following table:

Warning! If you are copying code directly from the examples in the PDF file, make sure that you change the slanted quotes to regular quotes or you will receive an error message.

Abbreviation	*Description*	*Example*
txt_col	Text column retrievable from a database.	&address
num_col	Numeric column retrievable from a database.	&price
date_col	Date or datetime column retrievable from a database.	&date1
txt_var	String variable defined in a program.	$your_name
num_var	Numeric variable defined in a program.	#total_cost
date_var	A variable explicitly defined as a date variable.	$date1
any_lit	A literal of any type.	'abc' 12
any_var	A variable of any type.	$string #number $date

Abbreviation	*Description*	*Example*
any_col	A column of any type.	&string &number &date
txt_lit	Text literal defined in a program.	'Company Confidential'
num_lit	Numeric literal defined in a program.	12345.67
int_lit	Integer literal defined in a program.	12345
nn	Integer literal used as an argument to a command.	123
position	The position qualifier, which consists of the line, column, and length specification. The minimum position, (), means to use the current line and column position on the page for the length of the field being printed.	(5,10,30)

SQR Commands

The following sections discuss the SQR commands in alphabetical order.

ADD

Syntax

```
ADD{src_num_lit | _var |
_col} TO dst_num_var
[ROUND=nn]
```

Description

Adds one number to another.

The source value is added to the destination variable and the result is placed in the destination. The source is always first and the destination is always second.

When dealing with money-related values, use decimal variables rather than float variables. Float variables are stored as double-precision floating point numbers, and small inaccuracies can occur when a program is adding many numbers in succession. These inaccuracies can appear due to the way floating point numbers are represented by different hardware and software implementations and also due to inaccuracies that can be introduced when a program is converting between floating point and decimal.

Parameters

Parameter	*Description*
src_num_lit | *_var* | *_col*	Source number literal, variable, or column.
dst_num_var	A numeric destination variable that contains the result after execution.
ROUND	Rounds the result to the specified number of digits to the right of the decimal point. For float variables, this value can be from 0 to 15. For decimal variables, this value can be from 0 to the precision of the variable. For integer variables, this argument is not appropriate.

Example

To add 10 to #counter:

```
add  #counter  to  #new_count
add  &price  to  #total  round=2
```

See Also

The LET command for information about complex arithmetic expressions.

ALTER-COLOR-MAP

Syntax

```
ALTER-COLOR-MAP
      NAME = {color_name_lit | _var | _col}
VALUE = ({color_name_lit | _var | _col} | {rgb})
```

Description

Dynamically alters a defined color.

The ALTER-COLOR-MAP command is allowed wherever the PRINT command is allowed. This command enables you to dynamically alter a defined color. You cannot use this command to define a new color.

Parameters

Parameter	*Description*
NAME	Defines the name of the color that you want to alter. For example, *light blue.*
VALUE	Defines the RGB value of the color that you want to alter, for example, (193, 233, 230).
{color_name_lit | _var | _col}	The color_name is composed of alphanumeric characters (A–Z, 0–9), the underscore (_) character, and the hyphen (-) character. It must start with an alpha (A–Z) character and is not case-sensitive. The name 'none' is reserved and cannot be assigned a value. A name in the format (RGBredgreenblue) cannot be assigned a value. The name 'default' is reserved and can be assigned a value. 'Default' is used during execution when a referenced color is not defined in the runtime environment.
{rgb}	*red_lit* | *_var* | *_col*,*green_lit* | *_var* | *_col*,*blue_lit* | *_var* | *_col* where each component is a value in the range of 000 to 255. In the BEGIN-SETUP section, only literal values are allowed.

The default colors implicitly installed with SQR include:

- black=(0,0,0)
- white=(255,255,255)
- gray=(128,128,128)
- silver=(192,192,192)
- red=(255,0,0)
- green=(0,255,0)
- blue=(0,0,255)
- yellow=(255,255,0)
- purple=(128,0,128)
- olive=(128,128,0)
- navy=(0,0,128)
- aqua=(0,255,255)
- lime=(0,128,0)
- maroon=(128,0,0)
- teal=(0,128,128)
- fuchsia=(255,0,255)

Example

The following example illustrates the ALTER-COLOR-MAP command:

```
begin-setup
   declare-color-map
   light_blue = (193, 222, 229)
   end-declare
end-setup

begin-program
   alter-color-map name = 'light_blue' value = (193, 233, 230)

   print 'Yellow Submarine' ()
      foreground = ('yellow')
      background = ('light_blue')

   get-color print-text-foreground = ($print-foreground)
   set-color print-text-foreground = ('purple')
   print 'Barney' (+1,1)
   set-color print-text-foreground = ($print-foreground)
end-program
```

See Also

DECLARE-COLOR-MAP, SET-COLOR, GET-COLOR

ALTER-LOCALE

Syntax

```
ALTER-LOCALE
[LOCALE={txt_lit _var|DEFAULT|SYSTEM}]
[NUMBER-EDIT-MASK={txt_lit|_var|DEFAULT|SYSTEM}]
[MONEY-EDIT-MASK={txt_lit|_var|DEFAULT|SYSTEM}]
[DATE-EDIT-MASK={txt_lit|_var|DEFAULT|SYSTEM}]
[INPUT-DATE-EDIT-MASK={txt_lit|_var|DEFAULT|SYSTEM}]
[MONEY-SIGN={txt_lit|_var|DEFAULT|SYSTEM}]
[MONEY-SIGN-LOCATION={txt_var|DEFAULT|SYSTEM|LEFT
|RIGHT}]
[THOUSAND-SEPARATOR={txt_lit|_var|DEFAULT|SYSTEM}]
[DECIMAL-SEPARATOR={txt_lit|_var|DEFAULT|SYSTEM}]
[DATE-SEPARATOR={txt_lit|_var|DEFAULT|SYSTEM}]
[TIME-SEPARATOR={txt_lit|_var|DEFAULT|SYSTEM}]
[EDIT-OPTION-NA={txt_lit|_var|DEFAULT|SYSTEM}]
[EDIT-OPTION-AM={txt_lit|_var|DEFAULT|SYSTEM}]
[EDIT-OPTION-PM={txt_lit|_var|DEFAULT|SYSTEM}]
[EDIT-OPTION-BC={txt_lit|_var|DEFAULT|SYSTEM}]
[EDIT-OPTION-AD={txt_lit|_var|DEFAULT|SYSTEM}]
[DAY-OF-WEEK-CASE={txt_var|DEFAULT|SYSTEM|UPPER|LOWER
|EDIT|NO-CHANGE}]
[DAY-OF-WEEK-FULL=({txt_lit1|_var1}...{txt_lit7
|_var7})]
[DAY-OF-WEEK-SHORT=({txt_lit1|_var1}...{txt_lit7
|_var7})]
[MONTHS-CASE={txt_var|DEFAULT|SYSTEM|UPPER|LOWER|EDIT
|NO-CHANGE}]
[MONTHS-FULL=({txt_lit1|_var1}...{txt_lit12|
_var12})]
[MONTHS-SHORT=({txt_lit1|_var1}...{txt_lit12|_var12})]
```

Description

Selects a locale or changes locale parameters used for printing date, numeric, and money data and for data accepted by the INPUT command. A locale is a set of preferences for language, currency, and presentation of charts and numbers.

The SYSTEM locale represents the behavior of older versions of SQR prior to Version 4.0. When you install SQR for PeopleSoft Version 4.0 or later, the default locale is set to SYSTEM. This provides upwards compatibility for older SQR programs. This table describes the SYSTEM locale settings:

Keyword	***Value***
NUMBER-EDIT-MASK	The PRINT command prints two digits to the right of the decimal point and left-justifies the number in the field. The MOVE, SHOW, and DISPLAY commands format the number with six digits to the right of the decimal point and left-justify the number.

Keyword	*Value*
MONEY-EDIT-MASK	SQR uses the same default as the NUMBER-EDIT-MASK keyword.
DATE-EDIT-MASK	SQR uses the default database date format. See the Date Time section for more details.
INPUT-DATE-EDIT-MASK	SQR uses a default date edit mask with the INPUT command. See the Sample Date Edit Masks table for a listing of the date edit mask.
MONEY-SIGN	'$'
MONEY-SIGN-LOCATION	LEFT
THOUSAND-SEPARATOR	','
DECIMAL-SEPARATOR	'.'
DATE-SEPARATOR	'/'
TIME-SEPARATOR	':'
EDIT-OPTION-NA	'NA'
EDIT-OPTION-AM	'am'
EDIT-OPTION-PM	'pm'
EDIT-OPTION-BC	'bc'
EDIT-OPTION-AD	'ad'
DAY-OF-WEEK-CASE	EDIT
DAY-OF-WEEK-FULL	('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday')
DAY-OF-WEEK-SHORT	('Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat')

Keyword	*Value*
MONTHS-CASE	EDIT
MONTHS-FULL	('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')
MONTHS-SHORT	('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')

Parameters

The following table lists and describes the parameters:

Note. Many of the settings can have a value of DEFAULT or SYSTEM. For a given setting, specifying DEFAULT retrieves the value from the corresponding setting of the *default* locale as identified in the Default-Settings section of the sqr.ini file. Similarly, specifying SYSTEM retrieves the value from the corresponding setting of the *system* locale. You can alter the *system* locale using the ALTER-LOCALE command; however, you cannot define it in the sqr.ini file.

Parameter	*Description*
LOCALE	Specifies the name of the locale to use. This name must be defined in the sqr.ini file. If this field is omitted, then the current locale is used. The locale name is not case-sensitive and is limited to the following character set: A–Z, 0–9, underscore, and hyphen. The current locale can be determined by printing the reserved variable $sqr-locale.
NUMBER-EDIT-MASK	Specifies the numeric edit mask to use with the keyword NUMBER in a PRINT, MOVE, SHOW, or DISPLAY command.
MONEY-EDIT-MASK	Specifies the numeric edit mask to use with the keyword MONEY in a PRINT, MOVE, SHOW, or DISPLAY command.
DATE-EDIT-MASK	The default date edit mask to use with the keyword DATE in the PRINT, MOVE, SHOW, or DISPLAY command, or the LET function datetostr() or strtodate().
INPUT-DATE-EDIT-MASK	The default date format to use with the INPUT command when TYPE=DATE is specified with the command or the input variable is a date variable. For information about edit masks, see *PRINT*.
MONEY-SIGN	Specifies the characters that replace the $ or other currency symbol used in edit masks.
MONEY-SIGN-LOCATION	Specifies where to place the MONEY-SIGN characters. Valid values are LEFT and RIGHT.

Parameter	*Description*
THOUSAND-SEPARATOR	Specifies the character to replace the ',' edit character.
DECIMAL-SEPARATOR	Specifies the character to replace the '.' edit character.
DATE-SEPARATOR	Specifies the character to replace the '/' character.
TIME-SEPARATOR	Specifies the character to replace the ':' character.
EDIT-OPTION-NA	Specifies the characters to use with the 'na' option.
EDIT-OPTION-AM	Specifies the characters to replace 'AM'.
EDIT-OPTION-PM	Specifies the characters to replace 'PM'.
EDIT-OPTION-BC	Specifies the characters to replace 'BC'.
EDIT-OPTION-AD	Specifies the characters to replace 'AD'.
DAY-OF-WEEK-CASE	Specifies how the case for the DAY-OF-WEEK-FULL or DAY-OF-WEEK-SHORT entries is affected when used with the format codes 'DAY' or 'DY'. Valid values are UPPER, LOWER, EDIT, and NO-CHANGE. UPPER and LOWER force the output to either all uppercase or lowercase, ignoring the case of the format code in the edit mask. Use EDIT to follow the case specified with the format code in the edit mask. Use NO-CHANGE to ignore the case of the format code and output the day of week explicitly listed in the DAY-OF-WEEK-FULL or DAY-OF-WEEK-SHORT entries.
DAY-OF-WEEK-FULL	Specifies the full names for the days of the week. SQR considers the first day of the week to be Sunday. You must specify all seven days.
DAY-OF-WEEK-SHORT	Specifies the abbreviated names for the days of the week. SQR considers the first day of the week to be Sunday. You must specify all seven abbreviations.
MONTHS-CASE	Specifies how the case for the MONTHS-FULL or MONTHS-SHORT entries is affected when used with the format code 'MONTH' or 'MON'. Valid values are UPPER, LOWER, EDIT, and NO-CHANGE. UPPER and LOWER force the output to either all uppercase or lowercase, ignoring the case of the format code in the edit mask. Use EDIT to follow the case specified with the format code in the edit mask. Use NO-CHANGE to ignore the case of the format code and format the month explicitly listed in the MONTHS-FULL or MONTHS-SHORT entries.
MONTHS-FULL	Specifies the full names for the months of the year. SQR for PeopleSoft considers the first month of the year to be January. You must specify all 12 months.
MONTHS-SHORT	Specifies the abbreviated names for the months of the year. SQR for PeopleSoft considers the first month of the year to be January. You must specify all 12 abbreviations.

Example

The following example illustrates the ALTER-LOCALE command:

```
!
! The following program segments will illustrate the various
! ALTER-LOCALE features.
!
begin-setup
 declare-variable
  date $date $date1 $date2 $date3
 end-declare
end-setup

 !
 ! Set default masks
 !
 alter-locale
  number-edit-mask = '9,999,999.99'
  money-edit-mask  = '$999,999,999.99'
  date-edit-mask   = 'Mon DD, YYYY'

 let #value = 123456
 let $edit = 'Mon DD YYYY HH:MI:SS'
 let $date = strtodate('Jan 01 2004 11:22:33', $edit)
 show 'With NUMBER option    #Value = ' #value number
 show 'With MONEY  option    #Value = ' #value money
 show 'Without NUMBER option #Value = ' #value
 show 'With DATE option      $Date  = ' $date date
 show 'Without DATE option   $Date  = ' $date
```

Produces the following output:

```
With NUMBER option    #Value =   123,456.00
With MONEY  option    #Value = $    123,456.00
Without NUMBER option #Value = 123456.000000
With DATE option      $Date  = Jan 01, 2004
Without DATE option   $Date  = 01-JAN-04

 !
 ! Reset locale to SQR defaults and assign a multi-character
 ! money-sign.
 !
 alter-Locale
  locale = 'System'
  money-sign = 'AU$'                ! Australian dollars

 let #value = 123456
 show #value edit '$999,999,999,999.99'
 show #value edit '$$$$,$$$$999,999.99'
```

Produces the following output:

```
AU$      123,456.00
       AU$123,456.00

 !
 ! Move the money-sign to the right side of the value.  Note
 ! the leading space.
 !
 alter-locale
  money-sign = ' AU$'               ! Australian dollars
  money-sign-location = right

 let #value = 123456
 show #value edit '$999,999,999,999.99'
 show #value edit '$$$$,$$$$999,999.99'
```

Produces the following output:

```
        123,456.00 AU$
        123,456.00 AU$

 !
 ! Reset locale to SQR defaults and flip the thousand and
 ! decimal separator characters.
 !
 alter-locale
  locale = 'System'
  thousand-separator = '.'
  decimal-separator = ','

 let #value = 123456
 show #value edit '999,999,999,999.99'
```

Produces the following output:

```
123.456,00

 !
 ! Reset locale to SQR defaults and change the date and time
 ! separators
 !
 alter-locale
  locale = 'System'
  date-separator = '-'
  time-separator = '.'

 let $edit = 'Mon/DD/YYYY HH:MI:SS'
 let $date = strtodate('Jan/01/2004 11:22:33', $edit)
 show $date edit :$edit
```

Produces the following output:

```
Jan-01-2004 11.22.33

 !
 ! Reset locale to SQR defaults and change the text used with
 ! the edit options 'na', 'am', 'pm', 'bc, 'ad'
 !
 alter-locale
  locale = 'System'
  edit-option-na = 'not/applicable'
  edit-option-am = 'a.m.'
  edit-option-pm = 'p.m.'
  edit-option-bc = 'b.c.'
  edit-option-ad = 'a.d.'

 let $value = ''
 let $edit = 'Mon DD YYYY HH:MI'
 let $date1 = strtodate('Jan 01 2004 11:59', $edit)
 let $date2 = strtodate('Feb 28 2004 12:01', $edit)
 show $value edit '999,999,999,999.99Na'
 show $date1 edit 'Mon DD YYYY HH:MI:SS PM'
 show $date2 edit 'Mon DD YYYY HH:MI:SS pm'
```

Produces the following output:

```
    Not/Applicable
Jan 01 2004 11:59:00 A.M.
Feb 28 2004 12:01:00 p.m.

 !
 ! Input some dates using the 'system' locale and
 ! output using other locales from the SQR.INI file.
 !
 alter-locale
  locale = 'System'
 let $date1 = strtodate('Jan 01 2004', 'Mon DD YYYY')
 let $date2 = strtodate('Feb 28 2004', 'Mon DD YYYY')
 let $date3 = strtodate('Mar 15 2004', 'Mon DD YYYY')
 show 'System:'
 show
 show $date1 edit 'Month DD YYYY' ' is ' $date1 edit 'Day'
 show $date2 edit 'Month DD YYYY' ' is ' $date2 edit 'Day'
 show $date3 edit 'Month DD YYYY' ' is ' $date3 edit 'Day'
 alter-locale
  locale = 'German'
 show
 show 'German:'
 show
 show $date1 edit 'DD Month YYYY' ' ist ' $date1 edit 'Day'
 show $date2 edit 'DD Month YYYY' ' ist ' $date2 edit 'Day'
 show $date3 edit 'DD Month YYYY' ' ist ' $date3 edit 'Day'
 alter-locale
  locale = 'Spanish'
 show
 show 'Spanish:'
 show
 show $date1 edit 'DD Month YYYY' ' es ' $date1 edit 'Day'
 show $date2 edit 'DD Month YYYY' ' es ' $date2 edit 'Day'
 show $date3 edit 'DD Month YYYY' ' es ' $date3 edit 'Day'
```

Produces the following output:

```
System:

January 01 2004 is Thursday
February 28 2004 is Saturday
March 15 2004 is Monday

German:

01 Januar 2004 ist Donnerstag
28 Februar 2004 ist Samstag
15 März 2004 ist Montag

Spanish:

01 enero 2004 es jueves
28 febrero 2004 es sábado
15 marzo 2004 es lunes
```

See Also

DISPLAY, LET, MOVE, PRINT, SHOW

Chapter 7, "Using the PSSQR.INI File and the PSSQR Command Line," page 301

ALTER-PRINTER

Syntax

```
ALTER-PRINTER
[POINT-SIZE={point_size_num_lit|_var}]
[FONT-TYPE={font_type|txt_var}]
[SYMBOL-SET={symbol_set_id|txt_var}]
[FONT={font_int_lit|_var}]
[PITCH={pitch_num_lit|_var}]
```

Description

Alters printer parameters at runtime.

You can place the ALTER-PRINTER command in any part of an SQR program except the SETUP section.

ALTER-PRINTER attempts to change the attributes of the *current* printer for the *current* report. If an attribute does not apply to the *current* printer, it is ignored. For example, ALTER-PRINTER is ignored if it specifies proportional fonts for a report printed on a line printer. When your program is creating multiple reports and the printer is shared by another report, the attributes are changed for that report as well.

Parameters

Parameter	***Description***
POINT-SIZE	Specifies the new font point size.

Parameter	*Description*
FONT-TYPE	Specifies the new font type. Values are PROPORTIONAL or FIXED.
SYMBOL-SET	Specifies the new symbol set identifier.
FONT	Specifies the new font as a number. (For example, font=3 for Courier and font=4 for Helvetica.)
PITCH	Specifies the new pitch in characters per inch. See the DECLARE-PRINTER arguments table for information about these arguments.

Example

Change the font and symbol set for the current printer:

```
alter-printer

font=4   ! Helvetica

symbol-set=12U   ! PC-850 Multilingual
```

If the output prints to a PostScript printer, the SYMBOL-SET argument is ignored; however, if the .spf file is kept (-KEEP) and later printed on an HP LaserJet, the symbol set 12U can be used.

See Also

The DECLARE-PRINTER command and the -KEEP command-line flag.

ALTER-REPORT

Syntax

```
ALTER-REPORT
 [HEADING={heading_name_txt_lit|_var|_col}]
 [HEADING-SIZE={heading_size_int_lit|_var|_col}]
 [FOOTING={footing_name_txt_lit|_var|_col}]
 [FOOTING-SIZE={heading_size_int_lit|_var|_col}]
```

Description

Alters some of the report-specific functionality.

This command enables you to dynamically change the heading or footing sections that are active for the current report. You can also change how much space the heading or footing sections occupy.

If the HEADING or FOOTING value is set to NONE, the section is disabled for the current report.

If the HEADING or FOOTING value is set to DEFAULT, the section reverts to the setting that was in effect when the report was initiated.

If no HEADING or FOOTING value is set, the HEADING-SIZE or FOOTING-SIZE values affect the HEADING/FOOTING currently being used.

If the ALTER-REPORT command was not invoked from within a BEGIN-HEADING or BEGIN-FOOTING section and the page has not been written to, the assignment takes effect immediately; otherwise, it takes effect for the next page.

Parameters

Parameter	*Description*
HEADING	Specifies the name of the BEGIN-HEADING section to use.
HEADING-SIZE	Specifies the amount of space the BEGIN-HEADING section occupies on the page.
FOOTING	Specifies the name of the BEGIN-FOOTING section to use.
FOOTING-SIZE	Specifies the amount of space the BEGIN-FOOTING section occupies on the page

Example

The following example illustrates the ALTER-REPORT command:

```
begin-footing 2 name=confidental
  print 'Company Confidential' (1,1,0) center
  page-number (2,37,0)
end-footing

begin-footing 2       name=proprietary
  print 'Company Proprietary' (1,1,0) center
  page-number (2,37,0)
end-footing

begin-report
 alter-report
   footing = 'Proprietary'
   footing-size = 6                    ! Increase depth

.
.
.
end-report
```

See Also

The BEGIN-FOOTING and BEGIN-HEADING commands in this section.

ARRAY-ADD, ARRAY-DIVIDE, ARRAY-MULTIPLY, ARRAY-SUBTRACT

Syntax

```
ARRAY-ADD{src_num_lit|_var|_col}...TO
dst_array_name (element_lit|_var|_col)[field
[(occurs_lit|_var|_col)]]...
ARRAY-DIVIDE{src_num_lit|_var|_col}...INTO
dst_array_name (element_int_lit|_var|_col)[field
[(occurs_lit|_var|_col)]]...
ARRAY-MULTIPLY{src_num_lit|_var|_col}...TIMES
dst_array_name(element_int_lit|_var|_col)[field
[(occurs_lit|_var|_col)]]...
ARRAY-SUBTRACT{src_num_lit|_var|_col}...FROM
dst_array_name (element_int_lit|_var|_col)[field
[(occurs_lit|_var|_col)]]...
```

Description

These four commands perform arithmetic operations on one or more elements in an array.

The following information applies to the array arithmetic commands:

- The array must first be created with the CREATE-ARRAY command.
- The four array arithmetic commands perform on one or more source numbers, placing the result into the corresponding field in the array.
- Array element and field occurrence numbers can be numeric literals (123) or numeric variables (#j) and can be from zero (0) to one less than the size of the array.
- If fields are not listed, the results are placed into consecutively defined fields in the array. If fields are listed, results are placed into those fields at the specified occurrence of the field. If an occurrence is not specified, the zeroth (0) occurrence is used.
- All fields must be of type number, decimal, float, or integer. They cannot be of type date, char, or text.
- If division by zero is attempted, a warning message appears, the result field is unchanged, and SQR continues running.

Parameters

Parameter	***Description***
src_num_lit|_var|_col	Source values are added to, divided into, multiplied by, or subtracted from the respective destination array fields. All variables must be numeric in type.
dst_array_name (element_int_lit|_var|_col) [field [(occurs_lit|_var|_col)]]	Destination array fields contain the results after the operation. All variables must be numeric in type.

Example

The following example adds &salary and #comm to the first two fields defined in the emps array. The #j'th element of the array is used:

```
array-add &salary #comm to emps(#j)
```

The following example subtracts #lost, #count, and 1 from the fields loses, total, and sequence of the #j2'th element of the stats array:

```
array-subtract #lost #count 1 from stats(#j2) loses total sequence
```

The following example multiplies occurrences 0 through 2 of the field p in the #i'th element of the percentages array by 2:

```
array-multiply 2 2 2 times percentages(#i) p(0) p(1) p(2)
```

The following example divides the #i2'th occurrence of the salesman field of the #j'th element of the commissions array by 100:

```
array-divide 100 into commissions(#j) salesman(#i2)
```

The following example uses the ARRAY-ADD command in an SQR program:

```
begin-setup
! declare arrays
create-array name=emps size=1              ! one row needed for this example
   field=Salary:number=35000               ! initialize to 35,000
   field=Comm:number=5000                  ! initialize to 5,000
end-setup

begin-program
do Main
end-program

begin-procedure Main local
! Show original contents of the arrays, then the modified arrays
! array-add
! retrieve values from the only row of array "emps"
get #sal #com FROM emps(0) Salary Comm
print 'Array-Add'         (+1, 1)

print 'Add 1000 to each column'  (+1, 1)
print 'Salary'  (+1, 3) bold underline
print 'Comm'  (,25) bold underline

print #sal   (+1, 1) money
print #com   (,22) money

let #salary = 1000
let #commission = 1000
let #j = 0  ! address the array row with variable "#j"
! Add 1000 (in variables) to each column of row 0 (the 1st and only row)
array-add #salary  #commission  TO emps(#j)
! retrieved the new "added" values
get #sal #com FROM emps(0) Salary Comm
print #sal  (+1,1) money
print #com  (,22) money
end-procedure
```

See Also

The CREATE-ARRAY command for information about creating an array.

The CLEAR-ARRAY command for information about clearing or initializing an array.

The GET, PUT, and LET commands for information about using arrays.

ASK

Syntax

ASK *substitution_variable* [*prompt*]

Description

Retrieves values for a compile-time substitution variable. The retrieval can be by user input or command-line arguments, or as entries in the @file on the command line.

The value of the substitution variable replaces the reference variable in the program. Variables are referenced by enclosing the variable name in braces, for example, '{state_name}'. When the substitution variable is text or date, enclose the brackets with single quotes. Substitutions are made when the program is compiled and are saved in the .sqt file. Each variable can be referenced multiple times.

ASK is used only in the SETUP section and must appear prior to any substitution variable references.

You cannot break the ASK command across program lines.

Parameters

Parameter	*Description*
Substitution_variable	The variable to be used as the substitution variable.
Prompt	An optional, literal text string to be displayed as a prompt if the value for the substitution variable is not entered on the command line or in an argument file.

Example

In the following example, state takes the value entered by the user in response to the prompt *Enter state for this report:*

```
begin-setup
  ask state 'Enter state for this report'
end-setup
  ...
begin-select
name, city, state, zip
from customers where state = '{state}'
end-select
```

See Also

The INPUT command for information about input at runtime.

PeopleTools 8.52: SQR for PeopleSoft Developers, "Compiling Programs and Using SQR Execute"

BEGIN-DOCUMENT

Syntax

```
BEGIN-DOCUMENT position
END-DOCUMENT
```

Description

Begins a document paragraph. A document paragraph enables you to write free-form text to create form letters, invoices, and so on.

You can reference database columns, SQR variables, and document markers within a document. Their locations in the document determine where they print on the page. You should not use tabs inside a document paragraph. To indent text or fields, use the spacebar. Note also that if the variables being printed inside a document paragraph are variable in length, you might need to manipulate the variable outside the document paragraph.

Note. A document must be run before you can reference its document markers. Because documents can be printed at relative positions on the page, the actual location of a document marker may not be known by SQR until the document itself has been run.

Parameters

Parameter	*Description*
position	The location on the page where the document begins. The position can be fixed or relative to the current position. See the POSITION command for a description and examples of the *position* parameter.

Example

The following example illustrates the BEGIN-DOCUMENT command

```
begin-document (1,1)
.b
Dear $firstname
  ...
end-document
```

See Also

END-DOCUMENT

PeopleTools 8.52: SQR for PeopleSoft Developers, "Creating Form Letters"

BEGIN-EXECUTE

Syntax

```
BEGIN-EXECUTE
[CONNECTION=uq_txt_lit]
[ON-ERROR=procedure[(arg1[,argi]...)]]
[RSV=num_var]
[STATUS=list_var|num_var|txt_var]
[SCHEMA=txt_lit|txt_var]
[PROCEDURE=txt_lit|txt_var
[PARAMETERS=(arg1[IN|INOUT|NULL[,argi[IN|INOUT]]...]])]
|COMMAND=txt_lit|txt_var
|GETDATA=txt_lit|txt_var]
[BEGIN-SELECT[BEFORE=sqr_procedure[(arg1[,argi]...]])]]
[AFTER=sqr_procedure[(arg1[,argi]...]])]]]
col-name type=char|text|number|date[edit-mask][on-break]...
FROM ROWSETS=({m,-n,n-m,m-|all})
|FROM PARAMETER=txt_lit|txt_var
END-SELECT]
END-EXECUTE
```

Description

Begins a new construct. In a BEGIN-EXECUTE paragraph, the syntax of BEGIN-SELECT varies as shown in the following syntax:

Parameters

Parameter	*Description*
CONNECTION	Identifies a name previously specified with the DECLARE-CONNECTION construct. If you do not specify a name, SQR Server uses the default connection. The default connection is defined by the command-line entries for datasource (DSN), username (USER), and password (PASSWORD). Name is not case-sensitive.
ON-ERROR	Declares the procedure to run if an error occurs.
RSV	Row Set Variable. A global SQR variable containing the row set being retrieved.

Parameter	*Description*
STATUS	Identifies a list or scalar variable that receives the status of the stored procedure.
SCHEMA	Identifies the location in the datasource of the object being queried.
PROCEDURE	The name of the datasource-stored procedure to be run. The name may include spaces. If the datasource is SAP R/3, this procedure is a BAPI.
PARAMETER_LIST	Scalar variables, list variables, or both of the form list_var | num_lit | txt_lit | txt_var | num_var | any_col. If you do not specify the keywords IN or INOUT, the default value is IN. Specify all parameters in order; leaving any parameters unnamed causes a syntax error. To ignore a parameter, fill its position with the keyword NULL. This results in a null value for that parameter position.
COMMAND	A text string that you pass to the datasource without modification by SQR. This string can include embedded SQR variables.
BEFORE/AFTER	Names an SQR procedure to be run before or after the row set. The procedure is not performed unless at least one row is returned from the specified row sets.
FROM ROWSET	Special case addition to the BEGIN-SELECT syntax. Available for use with all datasource types, including SAP R/3 and JDBC. Names the row sets from which to retrieve the column variables. If you specify more than one row set, use identical column name/type signatures. Row set numbers must be sequential from left to right within the parentheses, and they must not overlap as in this example: (1–3, 2–4). Numeric literals or #variables are allowed.
FROM PARAMETER	Special case addition to the BEGIN-SELECT syntax. Available only for SAP R/3 datasources. Use only with the PROCEDURE keyword. This argument names an output parameter containing one or more rows from which the column variables are to be retrieved.
PROPERTIES = (*txt_var* | *strlit* = *txt_var* | *strlit* | *num_var* | *num_lit* | *any_col*, …)	Specifies a set of keyword-value pairs that represent modifications to be made to the properties of the datasource (specified by the CONNECTION = statement). An arbitrary number of such pairs can be specified.

Note. This is a similar concept to the PARAMETERS = statement in DECLARE-CONNECTION, with the minor difference that the properties specified here alter the flow of returned information, as opposed to just setting login properties. Can be used in conjunction with any data-access model (Procedure, Command, Getdata). An application of this statement would be in the MDB setting, where it might be used to specify such things as Level, Generation, or Include-Column, for example, PROPERTIES = ('SetColumn' = 5)

Example

The following example illustrates the BEGIN-EXECUTE command

```
begin-setup
   declare-variable
      date $when_ordered
      text $ship_method
      integer #theRow
      integer #theStatus
      integer #howMany
   end-declare
end-setup

input #howMany type=integer
input $pword
let %parm1 = list($when_ordered, $ship_method, #howMany)

declare-connection SAPR3
user=scott
parameters=clientno=5;node=starfish;
end-declare

alter-connection
   name=SAPR3
   password=$pword

Begin-Execute
   connection=SAPR3
      rsv=#theRow
         status=#theStatus
            on-error=it_failed(#theStatus)
            procedure='CreditHistory version 5'
                parameters=(%parm1,'recalculate')

   print 'proc ran OK, status is ' (+1,1)
   print #theStatus (,+5) edit 999

Begin-Select before=do_eject after=cleanup
city &col=char (1,1) on-break level=1 after=city-tot
keyval type=number (1,+1)
rcvd type=date (0,+2)
from Rowsets=(1)
End-Select

End-Execute
```

See Also

EXECUTE

BEGIN-FOOTING

Syntax

```
BEGIN-FOOTING footing_lines_int_lit
[FOR-REPORTS=(ALL|report_name1[,report_namei]...)]
[FOR-TOCS=(ALL|toc_name1[,toc_namei]...)]
[NAME={footing_name}]
END-FOOTING
```

Description

Begins the FOOTING section.

The FOOTING section defines and controls the information to be printed at the bottom of each page.

You must define the *report_name* in a DECLARE-REPORT paragraph. If you do not use DECLARE-REPORT, the footing is applied to all reports. You can also specify FOR-REPORTS=(ALL). Note that the parentheses are required.

You can specify more than one BEGIN-FOOTING section; however, only one can exist for each report. A BEGIN-FOOTING section with FOR-REPORTS=(ALL) can be followed by other BEGIN-FOOTING sections for specific reports, which override the ALL setting.

You must define the *toc_name* in a DECLARE-TOC paragraph. You can also specify FOR-TOCS=(ALL). Note that the parentheses are required.

You can specify more than one BEGIN-FOOTING section; however, only one section can exist for each table of contents. A BEGIN-FOOTING section with FOR-TOCS=(ALL) can be followed by other BEGIN-FOOTING sections for a specific table of contents, which override the ALL setting.

The BEGIN-FOOTING section can be shared between reports and tables of contents.

You can print outside the footing area of the report—that is, in the body area—from the footing, but you cannot print in the footing area from the body.

Parameters

Parameter	***Description***
footing_lines_int_lit	The number of lines to be reserved at the bottom of each page.
FOR-REPORTS	Specifies the reports to which this footing applies. This argument is required only for a program with multiple reports. If you are writing a program that produces a single report, you can ignore this argument.
FOR-TOCS	Specifies the table of contents to which this heading applies.
NAME	Specifies the name to be associated with this footing section. Use this parameter with the ALTER-REPORT command. The name cannot be NONE or DEFAULT.

Example

The following example illustrates the BEGIN-FOOTING command

```
begin-footing 2 for-reports=(customer, summary)
  print 'Company Confidential' (1,1,0) center
  page-number (2,37,0)
end-footing
begin-footing 2                       ! For all reports
  print 'Division Report' (1,1,0) center
  page-number (2,37,0)
end-footing
begin-footing 2 for-tocs=(all)
  print 'Table of Contents' (2,1)
  let $page = roman(#page-count)      ! ROMAN numerals
  print $page (,64)
end-footing
```

See Also

The ALTER-REPORT command for information about dynamic headings and footings.

The DECLARE-LAYOUT command for information about page layout.

The DECLARE-REPORT command for information about programs with multiple reports.

The DECLARE-TOC command for information about the table of contents.

The END-FOOTING command.

BEGIN-HEADING

Syntax

```
BEGIN-HEADING heading_lines_int_lit
[FOR-REPORTS=(ALL|
report_name1[, report_namei]...)]
[FOR-TOCS=(ALL|toc_name1[, toc_namei]...)]
[NAME={footing_name}]
END-HEADING
```

Description

Begins a HEADING section.

The HEADING section defines and controls information to be printed at the top of each page.

You must define the *report_name* in a DECLARE-REPORT paragraph. If you do not use DECLARE-REPORT, the heading is applied to all reports. You can also specify FOR-REPORTS=(ALL). Note that the parentheses are required.

You can specify more than one BEGIN-HEADING section; however, only one can exist for each report. A BEGIN-HEADING section with FOR-REPORTS=(ALL) can be specified followed by other BEGIN-HEADING sections for specific reports, which override the ALL setting.

You must define the *toc_name* in a DECLARE-TOC paragraph. You can also specify FOR-TOCS=(ALL). Note that the parentheses are required.

You can specify more than one BEGIN-HEADING section; however, only one section can exist for each table of contents. A BEGIN-HEADING section with FOR-TOCS=(ALL) can be specified, followed by other BEGIN-HEADING sections for specific tables of contents, which override the ALL setting.

The BEGIN-HEADING section can be shared between reports and a table of contents.

You can print outside the heading area of the report—that is, in the body area—from the heading, but you cannot print in the heading area from the body.

Parameters

Parameter	*Description*
heading_lines_int_lit	The number of lines to be reserved at the top of each page.
FOR-REPORTS	Specifies the reports to which this heading applies. This is required only for a program with multiple reports. If you are writing a program that produces a single report, you can ignore this argument.
FOR-TOCS	Specifies the table of contents to which this heading applies.
NAME	Specifies the name to be associated with this heading section. This option cannot be used if FOR-REPORTS or FOR-TOCS is also specified. Use this parameter with the ALTER-REPORT command. The name cannot be NONE or DEFAULT.

Example

The following example illustrates the BEGIN-HEADING command

```
begin-heading 2                              ! Use 2 lines for
print $current-date (1,1) edit MM/DD/YY      ! heading,
  print 'Sales for the Month of ' (1,30)     ! 2nd is blank.
  print $month ()
end-heading
begin-heading 2 for-tocs=(all)
  print 'Table of Contents' (1,1) bold center
end-heading
```

See Also

The ALTER-REPORT command for information about dynamic headings/footings.

The DECLARE-LAYOUT command for information about page layout.

The DECLARE-REPORT command for information about programs with multiple reports.

The DECLARE-TOC command for information about Table of Contents.

The END-HEADING command.

BEGIN-PROCEDURE

Syntax

```
BEGIN-PROCEDURE procedure_name [LOCAL|(arg1
[, argi]...)]
END-PROCEDURE
```

Description

Begins a procedure. A procedure is one of the most powerful parts of the SQR language. It enables modularized functions and provides standard execution control.

The procedure name must be unique. The name is referenced in DO commands. Procedures contain other commands and paragraphs (for example, SELECT, SQL, DOCUMENT).

By default, procedures are global. That is, variables or columns defined within a procedure are known and can be referenced outside of the procedure.

A procedure is local when the word LOCAL appears after the procedure name or when the procedure is declared with arguments. That is, variables declared within the procedure are available only within the procedure, even when the same variable name is used elsewhere in the program. In addition, any query defined in a local procedure has local database, column-variable names assigned that do not conflict with similarly named columns defined in queries in other procedures.

SQR procedures can be called recursively. However, unlike C or Pascal, SQR maintains only one copy of the local variables and they are persistent.

Arguments passed by a DO command to a procedure must match in number:

- Database text or date columns, string variables, and literals can be passed to procedure string arguments. If you are passing a date string to a date argument, the date string must be in the format specified by the SQR_DB_DATE_FORMAT setting, or a database-dependent format, or the database-independent format SYYYYMMDD[HH24[MI[SS[NNNNNN]]]].

  See the Default Database Formats table in the PRINT command description.

- Database numeric columns, numeric variables, and numeric literals can be passed to procedure numeric arguments.

- Numeric variables (DECIMAL, INTEGER, FLOAT) can be passed to procedure numeric arguments without regard to the argument type of the procedure. SQR automatically converts the numeric values upon entering and leaving the procedure as required.

- Date variables or columns can be passed to procedure date or string arguments. When a date variable or column is passed to a string argument, the date is converted to a string according to the following rules:
  - For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

    If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.
  - For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

    If this has not been set, SQR uses the format listed in the Default Database Formats table.
  - For TIME columns, the format specified by the SQR_DB_TIME_ONLY_FORMAT setting is used.

    If this has not been set, SQR uses the format listed in the TIME Column Formats table.

To reference or declare global variables from a local procedure, add a leading underscore to the variable name, after the initial $, #, or &. (Example: #_amount)

Note. All the SQR-reserved variables, such as *#sql-status* and *$sql-error*, are global variables. Within a local procedure, they must be referenced by the leading underscore: *#_sql-status* or *$_sql-error*.

Parameters

Parameter	***Description***
procedure_name	Specifies a unique name for this procedure. Procedure names are not case-sensitive.
LOCAL	Specifies that this is a local procedure.
arg1 [, argi]...	Specifies the arguments to be passed to or returned from the procedure. Arguments can be string variables ($arg), numeric variables (#arg), or date variables ($arg). If you want to return a value passed back to the calling DO command, place a colon (:) before the variable name. The arguments of the BEGIN-PROCEDURE and DO commands must match in number, order, and type.

Example

The following example shows a procedure, **main,** that also runs the procedure **print_list** for each row returned from the Select statement. No parameters are passed to **print_list**:

```
begin-procedure main
begin-select
name
address
phone
  do print_list
from custlist order by name
end-select
end-procedure              ! main
```

In the following example, five arguments are passed to the **Calc** procedure:

```
do Calc (&tax, 'OH', &county_name, 12, #amount)

begin-procedure Calc(#rate, $state, $county, #months, :#answer)
.
.
.
let #answer = ...
end-procedure
```

In the preceding example, the value for *:#answer* is returned to *#amount* in the calling DO command.

The following example references global variables:

```
begin-procedure print-it ($a, $b)
print  $_deptname (+2,5,20)           ! $deptname is
print  $a         (,+1)               ! declared outside
print  $b         (,+1)               ! this procedure
end-procedure
```

See Also

DO, END-PROCEDURE

The Default Database Formats table in the Chapter 2, "SQR Command Reference," PRINT, page 193 command description.

BEGIN-PROGRAM

Syntax

```
BEGIN-PROGRAM
END-PROGRAM
```

Description

Begins the PROGRAM section of an SQR program.

After processing any commands in the SETUP section, SQR starts program execution at the BEGIN-PROGRAM section. The PROGRAM section typically contains a list of DO commands, though other commands can be used. This is the only required section in an SQR program.

Example

The following example illustrates the BEGIN-PROGRAM command:

```
begin-program
   do startup
   do main
   do finish
end-program
```


See Also

BEGIN-REPORT, BEGIN-SETUP, END-PROGRAM

BEGIN-SELECT

Syntax

```
BEGIN-SELECT[DISTINCT][-Cnn][-Bnn][-XP][-NR][-SORTnn]
[-LOCK{RR|CS|RO|RL|XX}][-DBdatabase]
[-DBconnectionstring]
[LOOPS=nn][ON-ERROR=procedure[(arg1[,argi]...)]]
{column} [&synonym]
{expression &synonym}
{[$columnname] &synonym = (char | number | date)}
[SQR commands]
FROM {table,...|[table:$tablename]}
                [additional SQL]
                [$variable]
END-SELECT
```

Description

Begins a SELECT paragraph. A SELECT paragraph is the principal means of retrieving data from the database and printing it in a report. A SELECT paragraph must be inside a PROCEDURE or BEGIN-PROGRAM section.

Note that SELECT * FROM is not a valid SQR SQL statement. BEGIN-SELECT can be placed inside a BEGIN-PROGRAM section.

Parameters

The table describes the parameters:

Note. The arguments can span multiple lines; however, the first character position cannot be used unless the continuation character terminated the previous line. If the first character position is used with arguments spanning multiple lines, the argument will be misconstrued as a Select column.

Parameter	*Description*
DISTINCT	Specifies that duplicate rows be eliminated from your query.
-C*nn*	(Oracle) Sets the context area size (buffer size for query) to larger or smaller than the default. This option is rarely needed.

Parameter	*Description*
-B*nn*	(Oracle, ODBC, Sybase CT-Lib) Sets the number of rows to retrieve at one time. This is for performance purposes only. Regardless of this setting, all rows are selected. The default, without using -B, is 10 rows. An overall setting for a program can be indicated on the SQR command line with -B, which can be overridden by a separate -B flag on each BEGIN-SELECT command.
-XP	(Sybase) Prevents the creation of a stored procedure for the SELECT paragraph. When -XP is specified, SQR generates a new SQL statement using the current value of any bind variables each time the BEGIN-SELECT is is carried out. This disables the potential performance optimization created by stored procedures. Use this flag if you change the variables frequently during execution and you do not want SQR to automatically create a stored procedure. You can also use -XP if the username/password to your program does not have permission to create stored procedures. If you do not change variables frequently during execution, the use of stored procedures may optimize your program's performance. In that case, do not use this argument. Note also that -XP is available as a command-line flag. -XP improves performance when you use bind variables and dynamic query variables in the same query. Each time the dynamic query variable changes in value, a new stored procedure is created. If the dynamic query variable changes often and the query contains bind variables, you create many stored procedures if you do not use -XP.
-DB*connectionstring*	(ODBC) Specifies the ODBC connection string for this SELECT paragraph only. A connection string has the following syntax: DSN=data_source_name[;keyword=value[;keyword=value [;...]]] This option enables you to combine data from multiple databases in one program. For example, a connection string for an Oracle database named ora8 might look like this: `'DSN=ora8;UID=scott;PWD=tiger'` where DSN, UID, and PWD are keywords common to all drivers (representing name, user ID, and password, respectively). Connection string options are always separated by a semicolon (;). Other driver-specific options can be added to the connection string by driver-defined keywords. See your ODBC driver documentation for available options.
LOOPS	Specifies the number of rows to retrieve. After the specified number has been processed, the SELECT loop exits.
ON-ERROR	Declares a procedure to run if an error occurs due to incorrect SQL syntax. Error trapping should be used with dynamic query variables. SELECT paragraphs without dynamic variables are checked for errors before the program is processed and therefore do not require a special error procedure. Optionally, you can specify arguments to be passed to the ON-ERROR procedure. Arguments can be any variable, column, or literal.

Example

In this example, duplicate rows are not selected for the city, state, and zip columns because of the **distinct** keyword. The numbers within parentheses accompanying city, state, and zip define the column positions of these rows. Column names cannot have spaces in front of them.

See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Column Variables."

```
begin-select distinct
city      (1,1,30)
state     (0,+2,2)
zip       (1,+3,6)
from custlist order by city
end-select
```

In this example, the first two columns may be present when the statement is compiled. The column cust_id is declared to be a number. A runtime error occurs if the database table, as identified by the variable *$table_name,* declares it to be something other than a number.

```
begin-select             loops=100
[$col_var_char]        &col1=char
[$col_var_num]           &col2=number
cust_id                &id=number
from [$table_name]
[$where clause]
[$order_by_clause]
end-select
```

In this example, the embedded SQR command `Do Print_Row` is carried out once for each row.

```
begin-select distinct
city      (1,1,30)
state     (0,+2,2)
zip       (1,+3,6)
    Do Print_Row
from custlist order by city
end-select
```

See Also

PeopleTools 8.52: SQR for PeopleSoft Developers, "Selecting Data from the Database" and *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Dynamic SQL and Error Checking"

END-SELECT, EXIT-SELECT

BEGIN-SETUP

Syntax

```
BEGIN-SETUP
END-SETUP
```

Description

Begins a SETUP section. This optional section is processed prior to the BEGIN-PROGRAM, BEGIN-HEADING, and BEGIN-FOOTING sections.

The SETUP section should be the first section in the program.

The SETUP section contains commands that determine the overall characteristics of the program. The commands used in the SETUP section cannot be used elsewhere unless specified. The SETUP section can include the following commands:

```
ASK
BEGIN-SQL
```

(The BEGIN-SQL command can also be used in BEGIN-PROCEDURE paragraphs.)

```
CREATE-ARRAY
```

(The CREATE-ARRAY command can also be used in the other sections of an SQR program.)

```
DECLARE-CHART
DECLARE-IMAGE
DECLARE-LAYOUT
DECLARE-PRINTER
DECLARE-PROCEDURE
DECLARE-REPORT
DECLARE-VARIABLE
```

(The DECLARE-VARIABLE command can also be used in LOCAL procedures.)

```
DECLARE-TOC
LOAD-LOOKUP
```

(The LOAD-LOOKUP command can also be used in the other sections of an SQR program.)

```
USE
```

(Sybase and Microsoft SQL Server only.)

Example

The following example illustrates the BEGIN-SETUP command:

```
begin-setup
   declare-layout customer_list
      paper-size=(8.5, 11)
      left-margin=1.0
      right-margin=1.0
   end-declare
end-setup
```

See Also

ASK, BEGIN-SQL, CREATE-ARRAY, LOAD-LOOKUP, USE

BEGIN-SQL

Syntax

```
BEGIN-SQL[-Cnn][-XP][-NR][-SORTnn]
[-LOCK{RR|CS|RO|RL|XX}]
[-DBdatabase][-DBconnectionstring]
   [ON-ERROR=procedure[(arg1[,argi]]...)]! In the SETUP section
   |[ON-ERROR={STOP|WARN|SKIP}](insetup)! Outside the SETUP section
END-SQL
```

Description

Begins an SQL paragraph. This paragraph can reside in a BEGIN-PROCEDURE, BEGIN-SETUP, or BEGIN-PROGRAM section.

BEGIN-SQL starts all SQL statements except SELECT, which has its own BEGIN-SELECT paragraph. If a single paragraph contains more than one SQL statement, each statement except the last must be terminated by a semicolon (;).

If a single paragraph contains more than one SQL statement, and the -C flag is used, all are assigned the same context area size or logical connection number.

Only non-SELECT statements can be used (except SELECT INTO for Sybase and Microsoft SQL Server). Columns and variables can be referenced in the SQL statements.

Stored Procedures

For Oracle, stored procedures are implemented by PL/SQL in the BEGIN-SQL paragraph. For Sybase and Microsoft SQL Server, SQR supports stored procedures with the EXECUTE command.

For some databases, such as Oracle, using DDL statements within a BEGIN-SQL paragraph causes a commit of outstanding inserts, updates, and deletes and releases cursors. For this reason, ensure that these are done in the proper order or the results will be unpredictable.

Oracle PL/SQL

For Oracle, PL/SQL is supported in a BEGIN-SQL paragraph. This requires an additional semicolon at the end of each PL/SQL statement.

For Oracle PL/SQL:


```
begin-sql
declare
   varpl varchar2 (25);;
   var2 number (8,2);;
begin
varpl :='abcdefg';;
$v1 :=varpl;;
$v2 :='1230894asd';;
var2 :=1234.56;;
#v :=var2;;
end;;
end-sql
```

For Oracle stored procedures:

```
begin-sql
begin
#dept_number :=get_dept_no($dept_name);;
end;;
end-sql
```

Parameters

Parameter	***Description***
-C*nn*	(Oracle) Sets the context area size (buffer size for query) to larger or smaller than the default. This option is rarely needed.
-XP	(Sybase) Prevents the creation of a stored procedure for the SQL paragraph. When -XP is specified, SQR generates a new SQL statement using the current value of the bind variables each time the BEGIN-SQL is carried out. This disables the performance optimization created by stored procedures. Use this flag if you change the variables frequently during execution and you do not want SQR to automatically create stored procedures. You can also use it if your program does not have permission to create stored procedures. If you do not change variables frequently during execution, the use of stored procedures optimizes the performance of the program. In that case, do not use this argument. -XP improves performance when you use bind variables and dynamic query variables in the same query. Each time the dynamic query variable changes in value, a new stored procedure is created. If the dynamic query variable changes often and the query contains bind variables, you create many stored procedures if you do not use -XP.

Parameter	*Description*

-DB*connectionstring*	(ODBC) Specifies the ODBC connection string for this SQL paragraph only. A connection string has the following syntax:
	`DSN=data_source_name[;keyword=value[;keyword=value[;...]]]`
	This option enables you to combine data from multiple databases in one program. For example, a connection string for an Oracle database named ora8 might look like this:
	`'DSN=ora8;UID=scott;PWD=tiger'`
	where DSN, UID, and PWD are keywords common to all drivers (representing name, user ID, and password, respectively). Connection string options are always separated by a semicolon (;). Other driver-specific options can be added to the connection string with driver-defined keywords. See your ODBC driver documentation for available options.

ON-ERROR	Declares a procedure to run if an error occurs due to incorrect SQL syntax except when the statement is run in a BEGIN-SETUP section. By default, SQR reports any error and then halts; if an error procedure is declared, you can trap errors, report or log them, and continue processing. The procedure is invoked when an error occurs in any SQL statement in the paragraph. After the error procedure ends, control returns to the next SQL statement. Optionally, you can specify arguments to be passed to the ON-ERROR procedure. Arguments can be any variable, column, or literal. If ON-ERROR is used in the SETUP section, it is a condition flag supporting the following conditions: STOP: Do not run the program. WARN: Run the program but with a warning message. SKIP: Ignore any errors and run the program.

Example

The following example illustrates the BEGIN-SQL command:

```
begin-sql
update orders set invoice_num = #next_invoice_num

where order_num = &order_num
end-sql

begin sql
delete orders

where order_num = &order_num;
insert into orders values ($customer_name, #order_num,...)
end-sql
```

See Also

PeopleTools 8.52: SQR for PeopleSoft Developers, "Using Dynamic SQL and Error Checking" and *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Additional SQL Statements with SQR"

END-SQL, BEGIN-PROCEDURE, EXECUTE

The -S command-line flag.

BREAK

Syntax

BREAK

Description

Causes an exit from within an EVALUATE or WHILE command. Execution then continues to the command immediately following the END-WHILE or END-EVALUATE.

This command is used inside a WHILE ... END-WHILE loop or within an EVALUATE command.

See Also

WHILE, EVALUATE

CALL, CALL SYSTEM

Syntax

```
CALL subroutine USING  {src_txt_lit|_var|_col}|{ src_num_lit|_var|_col} {
dst_txt_var|_num_var}  [param]
```

To issue operating system commands from within an SQR program, use the following syntax:

```
CALL SYSTEM USING commandstatus [ WAIT | NOWAIT ]
```

Description

Issues an operating system command or calls a subroutine that you have written in another language, such as C or COBOL, and passes the specified parameters.

You can write your own subroutines to perform tasks that are awkward in SQR. Subroutines can be written in any language.

Warning! Oracle recommends that the UCALL function not use any database calls because it may cause erroneous results.

Used in an SQR program, CALL has the following format:

```
CALL your_sub USING source destination [param_literal]
CALL SYSTEM USING command status [WAIT|NOWAIT]
```

The CALL SYSTEM is a special subroutine that is provided as part of SQR to enable the program to issue operating system commands. Its arguments, *command*, *status*, and WAIT|NOWAIT are described subsequently.

The values of the source and destination variables and the parameter's literal value are passed to your subroutine. Upon return from the subroutine, a value is placed in the destination variable.

You must write the subroutine and call it in one of the supplied UCALL routines. Optionally, you could rewrite UCALL in another language instead.

The source file UCALL.C contains sample subroutines written in C. The UCALL function takes the following arguments:

Argument	***Description***	***How Passed***
callname	Name of the subroutine.	By reference with a maximum of 31 characters, null terminated.
strsrc	Source string.	By reference with a maximum of 255 characters, null terminated.
strdes	Destination string.	By reference with a maximum of 255 characters.
dblsrc	Source double floating point.	By reference.
dbldes	Destination double floating point.	By reference.
param	Subroutine parameter string. It must be a literal.	By reference with a maximum of 80 characters, null terminated.

When you use the CALL command, your arguments are processed in the following way:

- Calling arguments are copied into the variables depending on the type of argument. Strings are placed into *strsrc* and numerics are placed into *dblsrc*.
- Return values are placed into *strdes* or *dbldes* depending on whether your destination argument for CALL is a string or numeric variable.

The destination arguments can also be used to pass values to your subroutine.

To access your subroutine, add a reference to it in UCALL and pass along the arguments that you need.

You must relink SQR to CALL after compiling a user-defined function that becomes another SQR function.

If you have created a new object file, you must add your subroutine to the link command file: in UNIX/Linux it is called SQRMAKE; in Microsoft Windows it is called SQREXT.MAK. (Alternatively, you could add your routine to the bottom of the UCALL source module that is already included in the link).

Your subroutine and calling SQR program are responsible for passing the correct string or numeric variables and optional parameter string to the subroutine. No checking is performed.

Parameters

Parameter	***Description***
subroutine	Specifies the name of your subroutine.
src_txt_lit|_var|_col	Specifies a text column, variable, or literal that is to be input to the called subroutine.
src_num_lit|_var|_col	Specifies a numeric column, variable (decimal, float, or integer), or literal that is to be input to the called subroutine.
dst_txt_var|_num_var	Specifies a text or numeric variable (decimal, float, or integer) into which the called subroutine is to place the return result.
param	Specifies an optional alphanumeric string of characters to be passed as a parameter to the subroutine.
SYSTEM	Specifies that this CALL command issues an operating system command.
command	Specifies the operating system command to carry out. The command can be a quoted string, string variable, or column.
status	Contains the status returned by the operating system. The status must be a numeric variable. The value returned in status is system-dependent as described here: UNIX/Linux: Zero (0) indicates success. Any other value is the system error code. PC/ Microsoft Windows: A value less than 32 indicates an error.
WAIT|NOWAIT	(Microsoft Windows only): WAIT specifies that SQR suspend its execution until the CALL SYSTEM command has finished processing. NOWAIT specifies that SQR start the CALL SYSTEM command but continue its own processing while that command is in progress. For Microsoft Windows, the default is NOWAIT. On UNIX\Linux operating systems, the behavior is always WAIT.

Example

For example, if your program runs under UNIX and you want to make a copy of a file, you can use the following code in your program:


```
!Executing a UNIX command from an SQR program
Let $Command_String='cp /usr/tmp/file1.dat /usr/tmp/file2.dat'
Call System Using $Command_String #Status
If #Status<>0
  Show 'Error executing the command in Unix: '$command
End-If
```

See these sample subroutines included in the UCALL source file:

- TODASH shows how strings can be manipulated.
- SQROOT demonstrates how to access numerics.
- SYSTEM invokes a secondary command processor.

The following code calls these subroutines:

```
call todash using $addr $newaddr '/.',                  ! Convert these to
                                                        ! dashes
call sqroot using #n #n2                                ! Put square root of
                                                        ! #n into #n2
call sqroot using &hnvr #j                              ! Hnvr is numeric
                                                        ! database column
call system using 'dir' #s                              ! Get directory listing
```

The following example uses the SYSTEM argument to issue an operating system command. Some operating systems enable you to invoke a secondary command processor to enter one or more commands and then return to SQR.

```
! Unix (Type 'exit' to return to SQR)
!
let $shell = getenv('SHELL')
if isblank($shell)
  let $shell = '/bin/sh'
end-if
call system using $shell #unix_status

!Windows (Type 'exit' to return to SQR)
!
let $comspec = getenv('COMSPEC')
let $cmd = comspec || '/c' ||$comspec || ' /k'
call system using $cmd #win_status wait
```

The following step-by-step example shows how to add a user-defined subroutine to SQR so that it can be invoked from SQR with the CALL command. For this example, the C function initcap, which makes the first letter of a string uppercase, is added. The function accepts two parameters. The first parameter is the string to which the initcap function is applied. The second is the resultant string.

To add the initcap function to SQR, you need to make the following modifications to the UCALL.C file that was provided with SQR:

1. Add the prototype for the initcap function:

   ```
   static void initcap CC_ARGS((char *, char *));
   ```

2. Modify the UCALL routine in the UCALL.C file.

   Specifically, add an else if statement at the end of the if statement to check for the initcap function:

```
void ucall CC_ARGL((callname, strsrc, strdes, dblsrc, dbldes, params))
...

    /* If other subroutines, add "else if..." statement for each */
    else if (strcmp(callname,"initcap") == 0)
        initcap(strsrc, strdes);
    else
        sq999("Unknown CALLed subroutine:  %s\n", callname);
    return;
    }
```

3. At the end of the UCALL.C file, add the initcap routine listed in the following example.

   The routine name must be lowercase; however, in your SQR program, it can be referenced by using either uppercase or lowercase.

```
static void initcap CC_ARGL((strsrc, strdes))
CC_ARG(char *, strsrc)              /* Pointer to source string      */
CC_LARG(char *, strdes)             /* Pointer to destination string */
   {
   int nIndex;
   int nToUpCase;
   char cChar;

   nToUpCase = 1;
   for (nIndex = 0; cChar = strsrc[nIndex]; nIndex++)
      {
      if (isalnum(cChar))
         {
         if (nToUpCase)
            strdes[nIndex] = islower(cChar) ? toupper(cChar) : cChar;
         else
            strdes[nIndex] = isupper(cChar) ? tolower(cChar) : cChar;
         nToUpCase = 0;
         }
      else
         {
         nToUpCase = 1;
         strdes[nIndex] = cChar;
         }
      }
   strdes[nIndex] = '\0';
   }
```

Note. The CC_ARG macros are defined in the UCALL.C source module. The macros enable the programmer to define a fully prototyped function without concern for whether the C compiler supports the feature.

After these modifications, recompile UCALL.C and relink SQR. See the programming manual for your particular machine for details.

Finally, the following example shows a simple SQR program that uses the initcap function:

```
begin-program
   input $name 'Enter the first name '! Get the first name from the user
   lowercase $name                    ! Set the first name to all lowercase
   call initcap using $name $capname  ! Now set the first character to uppercase
   input $last 'Enter the last name ' ! Get the last name from the user
   lowercase $last                    ! Set the last name to all lowercase
   call initcap using $last $caplast  ! Now set the first character to uppercase
   .
   .
   .
```

See Also

The LET command for information about user-defined functions using UFUNC.C that can be used in the context of an expression and that can either or both pass and return any number of arguments.

CLEAR-ARRAY

Syntax

`CLEAR-ARRAY NAME=`*`array_name`*

Description

Resets each field of an array to its initial value.

The CLEAR-ARRAY command resets each field of the named array to the initial value specified for that field in the CREATE-ARRAY command. If no initial value was specified, numeric fields are reset to zero, text fields are reset to null, and date fields are reset to null. CLEAR-ARRAY also releases all memory that was used by the specified array and returns it to its pristine state.

Parameters

Parameter	*Description*
NAME	Specifies the name of the array to be cleared.

Example

The following example illustrates the CLEAR-ARRAY command:

```
clear-array  name=custs
```

See Also

CREATE-ARRAY

CLOSE

Syntax

```
CLOSE {filenum_lit|_var_col}
```

Description

Closes a file, specified by its file number.

Closes a flat file that was previously opened with the OPEN command.

Parameters

Parameter	*Description*
filenum_lit|_var_col	Specifies the number assigned to the file in the OPEN command.

Example

The following example illustrates the CLOSE command:

```
close 5
close #j
```

See Also

OPEN, READ, WRITE

COLUMNS

Syntax

```
COLUMNS {int_lit|_var|_col}[int_lit|_var|_col]...
```

Description

Defines logical columns to be used for PRINT commands.

COLUMNS defines the leftmost position of one or more columns within the current page layout. It sets the first column as current.

You can use COLUMNS for printing data either down the page or across the page, depending on how you use the NEXT-COLUMN and USE-COLUMN commands.

The COLUMNS command applies only to the current report. If you want to print columns in more than one report, you must specify the COLUMNS command for each report.

The USE-COLUMN 0 deselects columns. See USE-COLUMN.

Parameters

Parameter	*Description*
int_lit|_var|_col	Specifies the left margin position of each column.

See Also

NEXT-COLUMN, NEXT-LISTING, NEW-PAGE, USE-COLUMN, USE-REPORT.

COMMIT

Syntax

```
COMMIT
```

Description

Causes a database commit.

COMMIT is useful when you are doing many inserts, updates, or deletes in an SQL paragraph. A database commit releases the locks on the records that have been inserted, updated, or deleted. Used with some databases, it also has other effects. For this reason, it should not be used within the scope of an active SELECT paragraph or results will be unpredictable.

When the application finishes, a commit is performed automatically unless a ROLLBACK was done or, for callable SQR, the -XC flag was set.

Other commands or options, such as the CONNECT command and the use of DDL statements for some databases with a BEGIN-SQL paragraph, can also cause the database to do a commit.

COMMIT is an SQR command and *should not* be used within an SQL paragraph. If COMMIT is used in an SQL paragraph, results will be unpredictable.

Note. The COMMIT command can be used with SQR servers for Oracle, DB2, Informix, and ODBC. For Sybase and Microsoft SQL Server, use BEGIN TRANSACTION and COMMIT TRANSACTION within SQL paragraphs as in the following code segment.

Example

The following example illustrates the COMMIT command:

```
add 1 to #updates_done
if #updates_done > 50
    commit
    move 0 to #updates_done
end-if
```

For Sybase:

```
...      ! Begin Transaction occurred previously
begin-sql
   insert into custlog values (&cust_num, &update_date)
end-sql
add 1 to #inserts
if #inserts >= 50
   begin-sql
      commit transaction;            ! Commit every 50 rows
      begin transaction              ! Begin next transaction
   end-sql
   move 0 to #inserts
end-if

...      ! One more Commit Transaction is needed
```

Warning! Any data being changed by a current transaction is locked by the database and cannot be retrieved in a SELECT paragraph until the transaction is completed by a COMMIT or ROLLBACK statement (or COMMIT TRANSACTION or ROLLBACK TRANSACTION statement for Sybase or Microsoft SQL Server).

CONCAT

Syntax

```
CONCAT {src_any_lit|_var|_col} WITH
dst_txt_var[[:$]edit_mask]
```

Description

Concatenates a variable, column, or literal with a string variable.

The contents of the source field are appended to the end of the destination field.

CONCAT can optionally edit the source field before appending it. You can change edit masks dynamically by placing them in a string variable and referencing the variable name preceded by a colon (:).

Also, the source can be a date variable or column. If an edit mask is not specified, the date is converted to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, the first database-dependent format listed in the Default Database Formats table is used.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT.

  If a format has not been set, the format listed in the Default Database Formats table is used.
- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT.

  If a format has not been set, the format listed in the Time Column Formats table is used.

Parameters

Parameter	*Description*
src_any_lit|_var|_col	Specifies the source field to be concatenated with the dst_txt_var field.
dst_txt_var	Contains the result after execution
edit_mask	Specifies an optional edit mask.

Example

The following example illustrates the CONCAT command:

```
concat &zip_plus_4 with $zip '-xxxx'        ! Edit zip plus 4.
concat &descrip with $rec :$desc_edit       ! Edit mask in variable.
concat $date1 with $string                  ! Concatenate a date.
```

See Also

The PRINT command for information about the Default Database Formats table, the Time Column Formats table, and edit masks.

The LET command for string functions.

STRING, UNSTRING

CONNECT

Syntax

```
CONNECT {txt_lit|_var|_col}[ON-ERROR=procedure[(arg1
[, argi]...)]]
```

Description

Logs off the database and logs on under a new username and password.

The new username and password can be stored in a string variable, column, or literal.

Warning! The username and password are not encrypted, so beware of security issues.

After each CONNECT, the reserved variable *$username* is set to the new username.

All database cursors or logons are closed before the CONNECT occurs. You should not issue a CONNECT within a SELECT or an SQL paragraph while a query is actively fetching or manipulating data from the database.

Parameters

Parameter	*Description*
txt_lit|_var|_col	Specifies a username and password for the logon.
ON-ERROR	Specifies a procedure to be run if the logon fails. If no ON-ERROR procedure is specified and the logon fails, SQR halts with an error message.

Note. You can optionally specify arguments to be passed to the ON-ERROR procedure. Arguments can be any variable, column, or literal.

Example

The following example illustrates the CONNECT command:

```
connect  $new-user  on-error=bad-logon($new_user)
connect  'sqr/test'
```

CREATE-ARRAY

Syntax

```
CREATE-ARRAY NAME=array_name SIZE=nn
{FIELD=name:type[:occurs]
[={init_value_txt_lit|_num_lit}]}...
```

Description

Creates an array of fields to store and process data.

You can define arrays to store intermediate results or data retrieved from the database. For example, a SELECT paragraph can retrieve data, store it in an array, and gather statistics at the same time. When the query finishes, a summary could be printed followed by the data previously stored in the array.

SQR creates arrays before a program starts to run. The CREATE-ARRAY command can be used in any section of a program.

Commands to process arrays include:


```
CREATE-ARRAY
CLEAR-ARRAY
GET
PUT
ARRAY-ADD
ARRAY-SUBTRACT
ARRAY-MULTIPLY
ARRAY-DIVIDE
LET
```

The maximum number of arrays in a program is 128; the maximum number of fields per array is 200.

The following code is a representation of an array *emps* with three fields in which the CREATE-ARRAY command defines the array:

```
create-array name=emps size=10
field=name:char='Unknown'
   field=rate:number:2=10.50
   field=phone:char='None'
```

The *name* is a simple field (one occurrence), *rate* has two occurrences, and *phone* is a simple field. Both array elements and field occurrences are referenced beginning with 0 (zero). The *rate* is referenced by rate(0) or rate(1). The *emps* array will contain 10 elements, 0 through 9. All *name* fields are initialized to Unknown, all *phone* fields are initialized to None, and all *rate* fields are initialized to 10.50.

Parameters

Parameter	***Description***
NAME	Names the array. The name is referenced in other array commands.
SIZE	Defines the number of elements in the array.
FIELD	Defines each field or column in the array. Each field must be defined as type: DECIMAL[(p)]: Decimal numbers with an optional precision (p). FLOAT: Double precision floating point numbers. INTEGER: Whole numbers. NUMBER: Uses the DEFAULT-NUMERIC type. See the DECLARE-VARIABLE command. CHAR (or TEXT): Character string. DATE: Same as date variable. You can specify an initialization value for each field. Each field is set to this value when the array is created and when the CLEAR-ARRAY command is carried out. If no initialization value is specified, numeric fields (DECIMAL, FLOAT, INTEGER) are set to zero, character fields are set to null, and date fields are set to null. All occurrences of a multiple occurring field are set to the same value. For dates, the initialization string must be formatted as 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.
OCCURS	Fields can optionally have a number of occurrences (occurs); that is, they can be repeated any number of times.

Example

The following example illustrates the CREATE-ARRAY command:

```
create-array  name=custs size=100
   field=name:char
   field=no:number
   field=state:char
   field=zip:char
   field=contacts:char:5
   field=last-contacted:date
```

See Also

The sample report CUSTOMR4.SQR included with SQR.

DECLARE-VARIABLE, ARRAY-ADD, ARRAY-DIVIDE, ARRAY-MULTIPLY, ARRAY-SUBTRACT, GET, PUT, LET, CLEAR-ARRAY.

The LOAD-LOOKUP command for an alternative way to store database tables in memory.

CREATE-COLOR-PALETTE

Syntax

```
CREATE-COLOR-PALETTE
 NAME = {palette_name_txt_lit}
 COLOR_1 = {rgb_value}
 COLOR_2 = {rgb_value}
 [COLOR_n] = {rgb_value}
```

Description

Create a color palette.

This command enables you to create a palette of colors. The number of palettes that can be defined in a program is not limited. No gaps are permitted in the palette.

Parameters

Parameter	*Description*
NAME	Specifies the name of the color palette.
COLOR_1	Specifies the first color in the palette.
COLOR_2	Specifies the second color in the palette.

Parameter	*Description*
COLOR_n	Specifies the nth color in the palette. You can specify up to 64 colors in the palette.
{*rgb*}	Designates a color reference. This can be expressed as *(r,g,b)*, where *r*, *g*, and *b* are either a numeric literal (0 to 255), a numeric variable, or a numeric column. It can also be expressed as a *(c)*, where *c* is a string literal, column, or variable that is the name of a color.

Example

The following example illustrates the CREATE-COLOR-PALETTE command:

```
begin-report
 create-color-palette
   name = 'funky'
   color_1 = ('blue')
   color_2 = ('red')
   color_3 = ('orange')

  Print-Chart Groovy
   Color-Palette = 'Funky'
 end-report
```

See Also

DECLARE-CHART, PRINT-CHART

#DEBUG

Syntax

#DEBUG[*x...*]*SQR_Command*

Description

Causes the current command to be processed during a debugging session.

A -DEBUG[*xx*] flag in the SQR command line enables conditional compilation of SQR commands. When this flag is used, any command (including other compiler directives) preceded by the word #DEBUG is processed; other commands are ignored.

This is useful for placing DISPLAY, SHOW, PRINT or other commands in your program for testing and for deactivating them when the report goes into production.

The -DEBUG flag can contain a suffix of up to 10 letters or digits. These characters are used to match any letters or digits appended to the #DEBUG preprocess command inside the program. #DEBUG commands with one or more matching suffix characters are processed; other commands are ignored. Commands without any suffix always match.

In addition, for each -DEBUG*xx* letter, a substitution variable is defined. For example, if the flag -DEBUGab is used on the command line, three substitution variables are defined: *debug*,*debuga*, and *debugb*. These variables can be referenced in #IFDEF commands to enable or disable whole sections of code for debugging.

Parameters

Parameter	*Description*
x	Represents any letter or digit.

Example

The following SQR command line contains the -DEBUG flag with no suffixes:

```
sqr  myprog  sammy/baker  -debug
```

The following SHOW command in the program is carried out if invoked with the previous command line because the -DEBUG flag was used:

```
#debug show 'The total is ' #grand-tot 999,999,999
```

In the following code example, the command line contains the -DEBUG flag with the suffixes a, b, and c:

```
sqr  myprog  sammy/baker  -debugabc
```

In the following code example, the first three #DEBUG commands are compiled, but the fourth, beginning *#debuge*, is not because its suffix does not match any of the suffixes on the -DEBUG flag:

```
#debuga  show 'Now selecting rows...'
#debug   show 'Finished query.'
#debugb  show 'Inserting new row.'
#debuge  show 'Deleting row.'
```

The following code example shows the use of an #IF with a #DEBUG:

```
#debuga  #if {platform}='unix'
#debuga     show 'Platform is UNIX'
#debuga  #endif
```

See Also

The #IF, #IFDEF, and #IFNDEF commands.

DECLARE-CHART

Syntax

```
DECLARE-CHART chart_name
[DATA-LABELS=data_labels_lit]
[COLOR-PALETTE=color_palette_lit]
[ITEM-COLOR=(chart_item_keyword_lit, color_value_lit |(r,g,b)]
[CHART-SIZE=(chart_width_int_lit,chart_depth_int_lit)]
[TITLE=title_txt_lit]
[SUB-TITLE=subtitle_txt_lit]
[FILL=fill_lit]
[3D-EFFECTS=3d_effects_lit]
[BORDER=border_lit]
[POINT-MARKERS=point_markers_lit]
[TYPE=chart_type_lit]
[LEGEND=legend_lit]
[LEGEND-TITLE=legend_title_txt_lit]
[LEGEND-PLACEMENT=legend_placement_lit]
[LEGEND-PRESENTATION=legend_presentation_lit]
[PIE-SEGMENT-QUANTITY-DISPLAY=
pie_segement_quantity_display_lit]
[PIE-SEGMENT-PERCENT-DISPLAY=
pie_segement_percent_display_lit]
[PIE-SEGMENT-EXPLODE=pie_segement_explode_lit]
[X-AXIS-LABEL=x_axis_label_txt_lit]
[X-AXIS-MIN-VALUE={x_axis_min_value_lit|_num_lit}]
[X-AXIS-MAX-VALUE={x_axis_max_value_lit|_num_lit}]
[X-AXIS-SCALE=x_axis_scale_lit]
[X-AXIS-MAJOR-TICK-MARKS=x_axis_major_tick_marks_lit]
[X-AXIS-MINOR-TICK-MARKS=x_axis_minor_tick_marks_lit]
[X-AXIS-MAJOR-INCREMENT=
{x_axis_major_increment_lit|_num_lit}]
[X-AXIS-MINOR-INCREMENT=
x_axis_minor_increment_num_lit]
[X-AXIS-TICK-MARK-PLACEMENT=
x_axis_tick_mark_placement_lit]
[X-AXIS-GRID=x_axis_grid_lit]
[Y-AXIS-LABEL=y_axis_label_lit]
[Y-AXIS-MIN-VALUE={y_axis_min_value_lit|_num_lit}]
[Y-AXIS-MAX-VALUE={y_axis_max_value_lit|_num_lit}]
[Y-AXIS-SCALE=y_axis_scale_lit]
[Y-AXIS-MAJOR-TICK-MARKS=y_axis_major_tick_marks_lit]
[Y-AXIS-MINOR-TICK-MARKS=y_axis_minor_tick_marks_lit]
[Y-AXIS-MAJOR-INCREMENT=
{y_axis_major_increment_lit|_num_lit}]
[Y-AXIS-MINOR-INCREMENT=
y_axis_minor_increment_num_lit]
[Y-AXIS-TICK-MARK-PLACEMENT=
y_axis_tick_mark_placement_lit]
[Y-AXIS-GRID=y_axis_grid_lit]
END-DECLARE
```

Note. If CHART-SIZE is not defined, it must be defined in PRINT-CHART.

Description

Defines the attributes of a chart that can later be displayed using PRINT-CHART.

The DECLARE-CHART command can define the attributes of a chart to be printed as part of a report.

This command can appear only in the SETUP section.

A chart defined with DECLARE-CHART is printed by referencing its name in the PRINT-CHART command. Some or all of the chart attributes can be overridden at runtime with the PRINT-CHART command. As such, DECLARE-CHART is useful when the basic properties of a chart are common to many PRINT-CHART commands.

Note. All DECLARE-CHART attributes can be overridden as part of the PRINT-CHART command. Columns are not supported within the DECLARE-CHART command or the PRINT-CHART command. Attributes that are specified more than once produce a warning, and the first instance is regarded as the actual value. Attributes can be used in any order, with the exception of *chart-name,* which must follow the DECLARE-CHART keyword.

Also, the FILL specification in the DECLARE-PRINTER command can influence the appearance of the chart. The following table lists the final appearance of the chart with a combination of values for PRINTER.COLOR and CHART.FILL options.

CHART.FILL=	***PRINTER.COLOR=Y***	***PRINTER.COLOR=N***
GRAYSCALE	GRAYSCALE	GRAYSCALE
COLOR	COLOR	GRAYSCALE
CROSS-HATCH	COLOR-CROSS-HATCH	CROSSHATCH
NONE	NONE	NONE

Specifying Chart Data Series Colors

Color palettes are used in the new graphics to set the colors of each data point in a data series. You specify the color palette to be used in a business chart by creating an SQR COLOR-PALETTE using the CREATE-COLOR-PALETTE command. The following code demonstrates how to create the color palette:

```
Create-Color-Palette
   Name = 'Test-Palette'
   Color_1 = (100,133,238)
   Color_2 = (0, 0, 255)
   Color_3 = (0,255,0)
   Color_4 = (0,0,255)
   Color_5 = (0,0,0)
```

Users can specify any number of palettes, with up to 64 colors defined in each palette. If more data points are in the data sets than are defined colors in the palette, the palette resets and continues to set the data point colors from Color_1 to Color_n.

After a color palette has been defined, it can be used within the DECLARE-CHART and PRINT-CHART commands to specify the color palette to be used. The following code example demonstrates the use of a color palette:

```
Print-Chart test_Chart
   COLOR-PALETTE = 'Test-Palette'
```

Specifying Chart Item Colors

Users can specify the foreground and background colors of the individual areas within a business chart using ITEM-COLOR = *(rgb-value)* within the DECLARE-CHART and PRINT-CHART commands. The following list shows chart item keywords that are valid for ITEM-COLOR:

- ChartBackground – Background color of entire chart area.
- ChartForeground – Text and Line color of chart area.
- HeaderBackground – Area within the text box specified for the Title and Subtitle.
- HeaderForeground – Text color of the Title and Subtitle.
- FooterBackground – Area within the text box specified for the X Axis label.
- FooterForeground – Text color of the X Axis label.
- LegendBackground – Area within the box defining the legend.
- LegendForeground – Text and outline color of the legend.
- ChartAreaBackground – Area that includes the body of the chart.
- ChartAreaForeground – Text and line colors of the chart area.
- PlotAreaBackground – Area within the X and Y Axis of a chart.
- PlotAreaForeground – Text and line colors of the plot area.

Parameters

Parameter	*Description*
chart_name	A unique name to be used for referencing a chart.
CHART-SIZE	The size of the chart frame in standard SQR coordinate units.

The following DECLARE-CHART Command Arguments table describes other arguments for the DECLARE-CHART command.

Note. Oracle does not currently support setting NewGraphics to Yes. You should not use the DATA-LABELS, COLOR-PALETTE, and ITEM-COLOR attributes listed in the following table because they are valid only when NewGraphics=Yes.

Argument	*Values*	*Description*
DATA-LABELS	Yes|No	If NewGraphics is set to Yes, use this argument to specify whether SQR prints the numeric value above the individual data points. Set to NO to suppress the numeric values.
COLOR-PALETTE	palette_name	If NewGraphics is set to Yes, use this argument to specify the name of the color palette to be used to color the individual data points in each chart (for example, bar, slice, point). A valid SQR color-palette must be defined to use COLOR-PALETTE.
ITEM-COLOR	ChartBackground|ChartForeground HeaderBackground|HeaderForeground FooterBackground|FooterForeground LegendBackground|LegendForeground ChartAreaBackground|ChartAreaForeground PlotAreaBackground	If NewGraphics is set to Yes, use this argument to specify the color of an individual item in a chart. Specify a chart item and a valid (r,g,b) color to set the color of the chart item.
TITLE	NONE | text	Specifies a title for the chart. That text is placed at the top of the chart.
SUB-TITLE	NONE | text	Specifies a subtitle for the chart. That text is placed below the title regardless of whether TITLE is specified.
FILL	GRAYSCALE | COLOR | CROSS-HATCH | NONE	Specifies the type of filling that is applied to the shapes (bars, pie segments, and so on) in the chart. GRAYSCALE varies the density of black dots. COLOR sends color instructions to the current printer. If the current printer does not support color, then color can appear in a GRAYSCALE fashion. CROSSHATCH uses patterns to fill the shapes representing each data set. With NONE, all graph shapes are filled with white.
3D-EFFECTS	YES | NO	Specifies whether the chart depth appears with 3-D effects. If this argument is set to NO, the chart is displayed in the default 2D mode.

Argument	*Values*	*Description*
BORDER	YES | NO	If this argument is set to YES, a border is drawn around the chart. If it is set to NO, no border is displayed around the chart.
POINT-MARKERS	YES | NO	Specifies whether point markers appear on line charts. If this argument is set to YES, point markers appear on line charts. If it is set to NO, point markers do not appear.
TYPE	LINE | PIE | BAR, STACKED-BAR | 100%-BAR | OVERLAPPED-BAR | FLOATING-BAR | HISTOGRAM | AREA | STACKED-AREA | 100%-AREA | XY-SCATTER-PLOT | HIGH-LOW-CLOSE	Specifies the type of chart. See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Business Charts."
LEGEND	YES | NO	Specifies whether to display a legend.
LEGEND-TITLE	NONE | text	Specifies the title for the legend. If this argument is set to NONE, no title is displayed in the legend box.
LEGEND-PLACEMENT	CENTER-RIGHT | CENTER-LEFT | UPPER-RIGHT | UPPER-LEFT | UPPER-CENTER | LOWER-RIGHT | LOWER-LEFT | LOWER-CENTER |	Places the legend in the specified location on the chart. The first portion of the placement parameter (CENTER, UPPER, or LOWER) is the vertical position, and the second portion (RIGHT, LEFT, or CENTER) is the horizontal.
LEGEND-PRESENTATION	INSIDE | OUTSIDE	Specifies where the legend appears on the chart. If this argument is set to INSIDE, the legend is presented inside the area defined by the two axes. If it is set to OUTSIDE, the legend is presented within the chart border, but outside of the region represented by the two axes.
PIE-SEGMENT-QUANTITY-DISPLAY	YES | NO	Specifies whether quantity is presented for each pie segment. If this argument is set to YES, the quantity is presented.
PIE-SEGMENT-PERCENT-DISPLAY	YES | NO	Specifies whether the percent-of-total number is presented for each pie segment. If this argument is set to YES, the percent-of-total figures is presented.

Argument	*Values*	*Description*
PIE-SEGMENT-EXPLODE	NONE | MAX |MIN | USE-3RD-DATA-COLUMN	Controls which pie segments are exploded (selected) within the pie chart. MAX selects the largest segment. MIN selects the smallest segment. USE-3RD-DATA-COLUMN uses the third field in the DATA-ARRAY to determine which pie segments are exploded. This third field should be a CHAR and values of YES or Y indicate that the segment should be exploded.
X-AXIS-LABEL or Y-AXIS-LABEL	NONE | text	Specifies a line of text to be displayed below (or alongside) the tick-mark labels on the axis.
X-AXIS-MIN-VALUE	AUTOSCALE | number	Specifies the minimum value on the axis. If data values exist that are less than X-AXIS-MIN-VALUE, they are not displayed. AUTOSCALE directs SQR to calculate an appropriate minimum value.
Y-AXIS-MIN-VALUE	AUTOSCALE | number	Specifies the minimum value on the axis. If data values exist that are less than Y-AXIS-MIN-VALUE, then they are not displayed. AUTOSCALE directs SQR to calculate an appropriate minimum value.
X-AXIS-MAX-VALUE	AUTOSCALE | number	Specifies the maximum value on the axis. If data values exist that are greater than X-AXIS-MAX-VALUE, they are not displayed. AUTOSCALE directs SQR to calculate an appropriate maximum value.
Y-AXIS-MAX-VALUE	AUTOSCALE | number	Specifies the maximum value on the axis. If data values exist that are greater than Y-AXIS-MAX-VALUE, they are not displayed. AUTOSCALE directs SQR to calculate an appropriate maximum value.
X-AXIS-SCALE or Y-AXIS-SCALE	LOG | LINEAR	Specifies the scale for the axis. LOG specifies a logarithmic scale for the axis. Otherwise, the scale is linear.

Argument	*Values*	*Description*
X-AXIS-MAJOR-TICK- MARKS	YES | NO	Specifies whether to display tick-marks for major increments on the X-axis. If this argument is set to YES, tick-marks appear on the axis between X-AXIS-MIN-VALUE and X-AXIS-MAX-VALUE, according to the X-AXIS-SCALE setting spaced by X-AXIS-MAJOR-INCREMENT.
Y-AXIS-MAJOR-TICK- MARKS	YES | NO	Specifies whether to display tick-marks for major increments on the Y-axis. If this argument is set to YES, tick-marks appear on the axis between Y-AXIS-MIN-VALUE and Y-AXIS-MAX-VALUE, according to the Y-AXIS-SCALE setting spaced by Y-AXIS-MAJOR-INCREMENT.
X-AXIS-MINOR-TICK- MARKS	YES | NO	Specifies whether to display tick-marks for minor increments on the X-axis. If this argument is set to YES, tick-marks appear on the axis between X-AXIS-MIN-VALUE and X-AXIS-MAX-VALUE, according to the X-AXIS-SCALE setting spaced by X-AXIS-MINOR-INCREMENT.
Y-AXIS-MINOR-TICK- MARKS	YES | NO	Specifies whether to display tick-marks for minor increments on the Y-axis. If this argument is set to YES, tick-marks appear on the axis between Y-AXIS-MIN-VALUE and Y-AXIS-MAX-VALUE, according to the Y-AXIS-SCALE setting spaced by Y-AXIS-MINOR-INCREMENT.
X-AXIS-MAJOR-INCREMENT or Y-AXIS-MAJOR-INCREMENT	AUTOSCALE | *number*	Specifies, for SQR, the increment used for spacing the major tick-marks on the axis. AUTOSCALE directs SQR to determine an appropriate increment.
X-AXIS-MINOR-INCREMENT or Y-AXIS-MINOR-INCREMENT	*number*	Specifies, for SQR, the increment used for spacing the minor tick-marks on the axis. These arguments must be set for the X-AXIS- MINOR-TICK-MARKS and the Y-AXIS- MINOR-TICK-MARKS to appear.
X-AXIS-TICK-MARK-PLACEMENT or Y-AXIS-TICK-MARK-PLACEMENT	INSIDE | OUTSIDE | BOTH	Specifies where to place the tick-marks on the axis. INSIDE (or OUTSIDE) directs SQR to place the tick-marks on the inside (or outside) of the axis only. BOTH directs SQR to draw the tick-marks such that they appear on both sides of the axis.

Argument	*Values*	*Description*
X-AXIS-GRID or Y-AXIS-GRID	YES | NO	Specifies whether a grid line is drawn for each major tick-mark on the axis. If this argument is set toYES, a dashed grid line is drawn for each major tick-mark. If it is set to NO, no grid line is drawn on the axis.

Example

This code example declares a basic sales chart using DECLARE-CHART. For each region, the SUB-TITLE, DATA-ARRAY, and other elements are overridden to provide the chart with the specific features desired.

```
begin-setup


declare-chart base_sales_chart
chart-size             = (30, 20 )
title                  = 'Quarterly Sales'
sub-title              = none
fill                   = color
3d-effects             = yes
type                   = stacked-bar
legend-title           = 'Product'
x-axis-grid            = yes
end-declare


end-setup

begin-program


print-chart base_sales_chart
sub-title               = 'Region I'
data-array              = reg1_sales
data-array-row-count    = #rows_reg1
data-array-column-count = 2
y-axis-max-value        = #max_of_all_regions
y-axis-min-value        = #min_of_all_regions
legend                  = no


print-chart base_sales_chart
sub-title               = 'Region II'
data-array              = reg2_sales
data-array-row-count    = #rows_reg2
data-array-column-count = 2
y-axis-max-value        = #max_of_all_regions
y-axis-min-value        = #min_of_all_regions
legend                  = no


end-program

begin-procedure chart_region_sales ($sub, $ary,
                     #rows, #cols,
                     #max_of_all_regions,
                     #min_of_all_regions)



print-chart base_sales_chart (20, 15 )
sub-title                 = $sub
data-array                = all sales
data-array-row-count      = #rows
data-array-column-count   = #cols
data-array-column-labels  = ('Q1', 'Q2', 'Q3', 'Q4' )
y-axis-max-value          = #max_of_all_regions
y-axis-min-value          = #min_of_all_regions
chart-size                = (50, 30)


end-procedure
```

See Also

The PRINT-CHART command.

DECLARE-COLOR-MAP

Syntax

In the SETUP section:

```
DECLARE-COLOR-MAP
      color_name = ({rgb})
color_name = ({rgb})
.
.
.
END-DECLARE
```

Description

Defines colors in an SQR report.

The DECLARE-COLOR-MAP command in the BEGIN-SETUP section defines or redefines colors in an SQR report. You can define an endless number of entries.

Parameters

Parameter	*Description*
color_name	A *color_name* is composed of alphanumeric characters (A–Z, 0–9), the underscore (_) character, and the hyphen (-) character. It must start with an alphabetical (A–Z) character and is not case-sensitive. The name **none** is reserved and cannot be assigned a value. A name in the format (RGBredgreenblue) cannot be assigned a value. The name **default** is reserved and cannot be assigned a value. **Default** is used during execution when a referenced color is not defined in the runtime environment.

Parameter	*Description*
{rgb}	*red_lit* | *_var* | *_col, green_lit* | *_var* | *_col, blue_lit* | *_var* | *_col* where each component is a value in the range of 000 to 255. In the BEGIN-SETUP section, only literal values are allowed. The default colors implicitly installed with SQR include: black=(0,0,0) white=(255,255,255) gray=(128,128,128) silver=(192,192,192) red=(255,0,0) green=(0,255,0) blue=(0,0,255) yellow=(255,255,0) purple=(128,0,128) olive=(128,128,0) navy=(0,0,128) aqua=(0,255,255) lime=(0,128,0) maroon=(128,0,0) teal=(0,128,128) fuchsia=(255,0,255)

Example

The following example illustrates the DECLARE-COLOR-MAP command:

```
begin-setup
   declare-color-map
    light_blue = (193, 222, 229)
   end-declare
end-setup
```

See Also

The ALTER-COLOR-MAP, SET-COLOR, and GET-COLOR commands in this section.

DECLARE-CONNECTION

Syntax

In the SETUP section:


```
DECLARE-CONNECTION connection_name_txt_lit
DSN={uq_txt_lit}
[USER={uq_txt_lit}]
[PASSWORD={uq_txt_lit}]
[PARAMETERS=keyword_str=attr_str;[,keyword_str=attr_str ;...]]
END-DECLARE
```

In the body of the report:

```
DECLARE-CONNECTION connection_name
DSN={uq_txt_lit|_var}
[USER={uq_txt_lit|_var}]
[PASSWORD={uq_txt_lit|_var}]
[PARAMETERS=keyword_str=attr_str;[, keyword_str=attr_str;...]]
END-DECLARE
```

Description

Defines the datasource logon parameters prior to logon. Can be used to override the default connection logon parameters.

Parameters

Parameter	***Description***
connection_name	A user-defined name for describing a datasource connection.
USER, PASSWORD	Traditional logon semantics.
PARAMETERS = *keyword_str= attr_str*;	Defines a list of keyword-attribute pairs required by a datasource driver for logon. No syntax restriction exists for these entries apart from the delimiting semicolons (;) and equal signs (=). The keywords must match the logon property names listed for a datasource.
NO-DUPLICATE= TRUE|FALSE (default is FALSE)	This optional keyword prevents SQR from automatically creating additional logins to datasources that are busy handling a previous query. Creating a new login in such cases is the default behavior for SQR, which allows a single CONNECTION declaration to be used in a subquery. The behavior, while allowing dynamic logins as needed, causes problems when you are doing both DDL (BEGIN-SQL) and DML (BEGIN-SELECT) against temporary tables in certain vendors' datasources. In such cases, you must fetch from the temporary table using the same login in which it was created. Here, you should code the CONNECTION as NO-DUPLICATE=TRUE, and then use that connection in both the table creation logic of BEGIN-SQL and the row fetching logic of BEGIN-SELECT.

Example

The following example illustrates the DECLARE-CONNECTION command:

```
declare-connection SAPR3-1
   dsn=SAPR3
   username=guest
   password=guest
end-declare
```

DECLARE-IMAGE

Syntax

```
DECLARE-IMAGE image_name
[TYPE=image_type_lit]
[IMAGE-SIZE=(width_num_lit,height_num_lit)]
[SOURCE=file_name_lit]
END-DECLARE
```

Note. If TYPE, IMAGE-SIZE, and SOURCE are not defined in DECLARE-IMAGE, they must be defined in PRINT-IMAGE.

Description

Declares the type, size, and source of an image to be printed.

The DECLARE-IMAGE command defines and names an image. This image can then be placed in a report at the position specified in the PRINT-IMAGE command.

Note. If the image file is unrecognizable, or has incomplete header information, a box (either shaded, for HP printers, or having a diagonal line through it in the case of postscript) appears where the image is expected.

Parameters

Parameter	***Description***
image_name	Specifies a unique name for referencing the image declaration.
TYPE	Specifies the image type. Types can be EPS-FILE, HPGL-FILE, GIF-FILE, JPEG-FILE, or BMP-FILE (for Microsoft Windows).
IMAGE-SIZE	Specifies the width and height of the image in SQR coordinates.
SOURCE	Specifies the name of a file containing the image. The file must be in the SQRDIR directory or you must specify the full path. **Note.** If the file is not in the SQRDIR directory, the full path or no path should be given. You cannot specify a relative path, because you must know where to run the file from.

Example

The following example illustrates the DECLARE-IMAGE command:

```
declare-image officer-signature
   type           = eps-file
   source         = 'off_sherman.eps'
   image-size     = (40, 5)
end-declare
```

See Also

PRINT-IMAGE.

DECLARE-LAYOUT

Syntax

```
DECLARE-LAYOUT layout_name
[PAPER-SIZE=({paper_width_num_lit[uom],
paper_depth_num_lit[uom]}|{paper_name})]
[FORMFEED=form_feed_lit]
[ORIENTATION=orientation_lit]
[LEFT-MARGIN=left_margin_num_lit[uom]]
[TOP-MARGIN=top_margin_num_lit[uom]]
[RIGHT-MARGIN=right_margin_num_lit[uom]
|LINE-WIDTH=line_width_num_lit[uom]
|MAX-COLUMNS=columns_int_lit]
[BOTTOM-MARGIN=bottom_margin_num_lit[uom]
|PAGE-DEPTH=page_depth_num_lit[uom]
|MAX-LINES=lines_int_lit]
[CHAR-WIDTH=char_width_num_lit[uom]]
[LINE-HEIGHT=line_height_num_lit[uom]]
END-DECLARE
```

Description

Defines the attributes for the layout of an output file.

The DECLARE-LAYOUT command describes the characteristics of a layout to be used for an output file. A layout can be shared by more than one report. If no DECLARE-LAYOUT is defined or if a DECLARE-REPORT does not reference a defined layout, a layout named DEFAULT is created with the default attribute values shown in the DECLARE-LAYOUT Command Arguments table. For an example of how DECLARE-LAYOUT relates to DECLARE-REPORT, see the DECLARE-REPORT examples in this document.

You can define as many layouts as are necessary for the requirements of the application. You can override the DEFAULT layout attributes by defining a layout called DEFAULT in your program. Each layout name must be unique.

SQR maps its line and column positions on the page by using a grid determined by the LINE-HEIGHT and CHAR-WIDTH arguments. That is, SQR calculates the number of columns per row by dividing the LINE-WIDTH by the CHAR-WIDTH and calculates the number of lines by dividing the PAGE-DEPTH by the LINE-HEIGHT. Each printed segment of text is placed on the page using this grid. Because the characters in proportional fonts vary in width, a word or string may be wider than the horizontal space you have allotted, especially in words containing uppercase letters or boldfaced characters. To account for this behavior, you can either move the column position in the PRINT or POSITION statements or indicate a larger CHAR-WIDTH in the DECLARE-LAYOUT command.

The DECLARE-LAYOUT command selects the proper fonts. In addition, the parameter interacts with PAPER-SIZE like this:

- When you do not specify ORIENTATION=LANDSCAPE or the PAPER-SIZE dimensions, SQR creates a page with the dimensions set to 11 inch by 8.5 inch. This results in a page of 100 columns by 45 lines with 0.5 inch margins.
- When you specify PAPER-SIZE=*(paper_name)*, the page orientation is set according to the *paper_name* specified. If you also specify ORIENTATION and the value differs from the PAPER-SIZE value, the ORIENTATION value overrides the PAPER-SIZE value.
- When you specify PAPER-SIZE=*(page_width, page_depth)*, SQR *does not* swap the page width and page depth if ORIENTATION=LANDSCAPE.

Parameters

Parameter	***Description***
layout_name	A unique layout name to be used for referencing the layout and its attributes.
uom (unit of measure)	An optional suffix that denotes the unit of measure to apply to the preceding value.
paper_name	An option of PAPER-SIZE. This name is associated with predefined dimensions.

This table lists valid unit of measure suffixes:

Suffix	***Meaning***	***Definition***
dp	decipoint	0.001388 inch
pt	point	0.01388 inch
mm	millimeter	0.03937 inch
cm	centimeter	0.3937 inch
in	inch	1.0000 inch

This table lists valid paper names for the *paper_name* parameter.

Name	***Width***	***Depth***	***Orientation***
Letter	8.5 in	11 in	Portrait

Name	*Width*	*Depth*	*Orientation*
Legal	8.5 in	14 in	Portrait
A4	8.27 in	11.69 in	Portrait
Executive	7.25 in	10.5 in	Portrait
B5	7.17 in	10.12 in	Portrait
Com-10	4.125 in	9.5 in	Landscape
Monarch	3.875 in	7.5 in	Landscape
DL	4.33 in	8.66 in	Landscape
C5	6.378 in	9.016 in	Landscape

This table describes the arguments of the DECLARE-LAYOUT command:

Argument	*Choice or Default UOM*	*Default Value*	*Description*
PAPER-SIZE	inches	8.5 in, 11 in	Physical size of the page. The first parameter is the width of the page. The second parameter is the depth or length. It may also be a predefined name. (See the table of valid paper names.) Note that when ORIENTATION= LANDSCAPE, the default values are 11 in, 8.5 in.
FORMFEED	YES, NO	YES	Specifies whether formfeeds are to be written at the end of each page.

Argument	*Choice or Default UOM*	*Default Value*	*Description*
ORIENTATION	PORTRAIT, LANDSCAPE	PORTRAIT	Portrait pages are printed vertically. Landscape pages are printed horizontally. Printing in landscape for the printer type HPLASERJET requires landscape fonts.
LEFT-MARGIN	inches	0.5 in	Amount of blank space to leave at the left side of the page.
TOP-MARGIN	inches	0.5 in	Amount of blank space to leave at the top of the page.
RIGHT-MARGIN	inches	0.5 in	Amount of blank space to leave at the right side of the page. If you specify LINE-WIDTH or MAX-COLUMNS, you cannot use this parameter.
LINE-WIDTH	inches	7.5 in	Length of the line. If you specify RIGHT-MARGIN or MAX-COLUMNS, you cannot use this parameter.
MAX-COLUMNS	NA (not applicable)	75	Maximum number of columns in a line. If you specify RIGHT-MARGIN or LINE-WIDTH, you cannot use this parameter.
BOTTOM-MARGIN	inches	0.5 in	Amount of blank space to leave at the bottom of the page. If you specify PAGE-DEPTH or MAX-LINES, you cannot use this parameter.
PAGE-DEPTH	inches	10 in	Depth of the page. If you specify BOTTOM-MARGIN or MAX-LINES, you cannot use this parameter.

Argument	*Choice or Default UOM*	*Default Value*	*Description*
MAX-LINES	NA	60	Maximum number of lines printed on the page. If you specify PAGE-DEPTH or BOTTOM-MARGIN, you cannot use this parameter.
LINE-HEIGHT	points	12 pt	Size of each SQR line on the page. An inch has 72 points. If LINE-HEIGHT is not specified, it follows the value for POINT-SIZE, if specified. The default value of 12 points yields 6 lines per inch. For the printer type LINEPRINTER, this value is used only to calculate the TOP-MARGIN and BOTTOM-MARGIN (for example, not in computing the position on the page).
CHAR-WIDTH	points	7.2 pt	Size of each SQR horizontal character column on the page (for example, the distance between the locations (1, 12) and (1, 13)). For the printer type LINEPRINTER, this value is used only to calculate the TOP-MARGIN and BOTTOM-MARGIN (not for computing the position on the page).

Example

This example illustrates the ability to specify these parameters using a different measurement system, such as metric:

```
!
declare-layout my-layout      ! Results in:

paper-size=(a4)               !   paper-size=(210mm, 297mm)

left-margin=12.7 mm           !   top-margin=12.7mm

right-margin=25.4 mm          !  left-margin=12.7mm
end-declare                   !  right-margin=25.4mm

                              !   bottom-margin=12.7mm

                              !   orientation=portrait

                              !   columns=67

                              !   lines=64
```

This example changes the page dimensions and also changes the left and right margins to be 1 inch:

```
!
declare-layout large-paper    ! Results in:

paper-size=(14, 11)           !   paper-size=(14in, 11in)

left-margin=1                 !   top-margin=0.5in

right-margin=1                !   left-margin=1.0in
end-declare                   !   right-margin=1.0in

                              !   bottom-margin=0.5in

                              !   orientation=portrait

                              !   columns=120

                              !   lines=60
```

This example retains the default page dimensions and changes the left and right margins to be 1 inch:

```
declare-layout default        ! Results in:

left-margin=1                 !   paper-size=(8.5in, 11in)

right-margin=1                !   top-margin=0.5in
end-declare                   !   left-margin=1.0in

                              !   right-margin=1.0in

                              !   bottom-margin=0.5in

                              !   orientation=portrait

                              !   columns=65

                              !   lines=60
```

This example changes the orientation to landscape; the default page dimensions (8.5in and 11in) are swapped, the columns and rows are recalculated, and all other values remain the same:

```
declare-layout default          ! Results in:

orientation=landscape           !   paper-size=(11in, 8.5in)
end-declare                     !   top-margin=0.5in

                                !   left-margin=0.5in

                                !   right-margin=0.5in

                                !   bottom-margin=0.5in

                                !   orientation=landscape

                                !   columns=100

                                !   lines=45
```

This example changes the orientation to landscape; the default page dimensions (8.5in and 11in) are swapped, and the top margin is set to 1 inch:

```
declare-layout my_landscape     ! Results in:

orientation=landscape           !   paper-size=(11in, 8.5in)

top-margin=1                    !   top-margin=1.0in
end-declare                     !   left-margin=0.5in

                                !   right-margin=0.5in

                                !   bottom-margin=0.5in

                                !   orientation=landscape

                                !   columns=100

                                !   lines=43
```

This example illustrates how to specify the page dimensions using one of the predefined names (note that the orientation has also changed because this example is an envelope):

```
declare-layout envelope          ! Results in:

paper-size=(com-10)              !   paper-size=(4.125in, 9.5in)
end-declare                      !   top-margin=0.5in

                                 !   left-margin=0.5in

                                 !   right-margin=0.5in

                                 !   bottom-margin=0.5in

                                 !   orientation=landscape

                                 !   columns=85

                                 !   lines=18
```

See Also

DECLARE-REPORT

DECLARE-PRINTER

Syntax

```
DECLARE-PRINTER printer_name
[FOR-REPORTS=(report_name1[,report_namei]...)]
[TYPE=printer_type_lit]
[INIT-STRING=initialization_string_txt_lit]
[RESET-STRING=reset_string_txt_lit]
[COLOR=color_lit]
[POINT-SIZE=point_size_num_lit]
[FONT-TYPE=font_type_int_lit]
[SYMBOL-SET=symbol_set_id_lit]
[STARTUP-FILE=file_name_txt_lit]
[PITCH=pitch_num_lit]
[FONT=font_int_lit]
[BEFORE-BOLD=before_bold_string_txt_lit]
[AFTER-BOLD=after_bold_string_txt_lit]
END-DECLARE
```

Description

Overrides the printer defaults for the specified printer type.

Each printer has a set of defaults as listed in the DECLARE-PRINTER Command Arguments table. The DECLARE-PRINTER command overrides these defaults.

Use the DECLARE-PRINTER command in the SETUP section to define the characteristics of the printer or printers to be used. If you need to change some of the arguments depending on the runtime environment, you can use the ALTER-PRINTER command in any part of the program except the PROGRAM and SETUP sections.

A program can contain no more than one DECLARE-PRINTER command for each printer type for each report. If you do not provide a printer declaration, the default specifications are used. You can override the default printer attributes by providing a DECLARE-PRINTER specification for each printer. The names are:

- DEFAULT-LP for line printer.
- DEFAULT-HP for HP LaserJet.
- DEFAULT-HT for HTML.
- DEFAULT-PS for PostScript.

This table lists the arguments, provides the possible choices or measure, lists the default values, and describes the arguments.

Argument	*Choice or Measure*	*Default*	*Description*
FOR-REPORTS	NA (Not Applicable)	ALL	The name of the reports that use this printer definition. The default is ALL, for all reports. This argument is required only for a program with multiple reports. If you are writing a program that produces only a single report, you can ignore this argument.
TYPE	LINEPRINTER, POSTSCRIPT, HPLASERJET, HTML, LP, PS, HP, HT	LP	The output type specific to each printer. LINEPRINTER (LP) files can be viewed by a text editor. POSTSCRIPT (PS) files require you to know PostScript to understand what will be shown on the printer. HPLASERJET (HP) files are binary files and cannot be edited or viewed. HTML (HT) files can be viewed by a browser.
INIT-STRING	NA	(none)	Sends control or other characters to the printer at the beginning of the report. This parameter is designed primarily for the LINEPRINTER and has limited use with other printer types. Specify nondisplay characters by placing their decimal values inside angle brackets. For example, <27> is the ESC or escape character.

Argument	*Choice or Measure*	*Default*	*Description*
RESET-STRING	NA	(none)	Sends control or other characters to the printer at the end of the report. This parameter is designed primarily for the LINEPRINTER and has limited use with other printer types. Specify nondisplay characters by placing their decimal values inside angle brackets. For example, <27> is the ESC or escape character.
COLOR	Yes, No	No	Specifies whether this printer can print in color.
POINT-SIZE	points	12	This argument does not apply to LINEPRINTER printers. It is the beginning size of the selected font.
FONT-TYPE	PROPORTIONAL, FIXED	Depends on the font	This argument applies only to HPLASERJET printers and needs to be specified only for font types not defined in the Fonts Available for HP LaserJet Printers in the SQR table.
SYMBOL-SET	HP defined sets	0U	This argument applies only to HPLASERJET printers. The default value of 0U is for the USASCII symbol set. For a complete list of the symbol sets, see the *HP LaserJet Technical Reference Manual.*

Argument	*Choice or Measure*	*Default*	*Description*
STARTUP-FILE	file name	POSTSCRI.STR	This argument applies only to POSTSCRIPT printers. This argument is used to specify an alternate startup file. Unless otherwise specified, the default startup file is located in the directory pointed to by the environment variable SQRDIR.
PITCH	characters/inch	10	This argument is for HPLASERJET printers only. If you specify a fixed pitch font, you should also indicate the pitch.

Argument	*Choice or Measure*	*Default*	*Description*
FONT	font_number	3	This is the font number of the typeface to use. For HPLASERJET printers, this is the typeface value as defined by Hewlett-Packard. For a complete list of the typeface numbers, see the *HP LaserJet Technical Reference Manual.* For POSTSCRIPT printers, SQR supplies a list of fonts and arbitrary font number assignments in the file POSTSCRI.STR. The font numbers are the same as those for HP LaserJet printers, wherever possible, so that you can use the same font number for reports to be printed on both types of printers. You can modify the font list in POSTSCRI.STR to add or delete fonts. Read the POSTSCRI.STR file for instructions. The Fonts Available for HP LaserJet Printers in SQR table lists the fonts available in SQR internally. The Fonts Available for PostScript Printers table lists the fonts available in the SQR POSTSCRI.STR file.

Argument	*Choice or Measure*	*Default*	*Description*
BEFORE-BOLD	any string	(none)	The BEFORE-BOLD and AFTER-BOLD arguments are for LINEPRINTER printers only. They specify the character string to enable or disable boldfacing. If the string contains blank characters, enclose it in single quote marks ('...'). To specify nonprinting characters, such as ESC, enclose the decimal value inside angle brackets as shown here: `BEFORE-BOLD=<27>[r ! Turn on bold AFTER-BOLD=<27>[u ! Turn it off` These arguments work with the BOLD argument of the PRINT command.
AFTER-BOLD	any string	(none)	See BEFORE-BOLD.

This table lists the fonts that are available in SQR for use with the FONT argument for HPLASERJET printer types.

Value	*Typeface*	*Style*
0	Line printer	Fixed
1	Pica	Fixed
2	Elite	Fixed
3	Courier	Fixed
4	Helvetica	Proportional
5	Times Roman	Proportional
6	Letter Gothic	Fixed

Value	*Typeface*	*Style*
8	Prestige	Fixed
11	Presentations	Fixed
17	Optima	Proportional
18	Garamondi	Proportional
19	Cooper Black	Proportional
20	Coronet Bold	Proportional
21	Broadway	Proportional
22	Bauer Bodini Black Condensed	Proportional
23	Century Schoolbook	Proportional
24	University Roman	Proportional

The font that you choose—in orientation, typeface, and point size—must be an internal font, available in a font cartridge, or downloaded to the printer.

For fonts not listed in the Fonts Available for HP LaserJet Printers in SQR table, you must indicate the font style using the FONT-TYPE argument to ensure that the correct typeface is selected by the printer.

This table lists the fonts that are available in SQR for use with the FONT argument for PostScript printer types:

Value	*Typeface*	*Boldface Type Available*
3	Courier	Y
4	Helvetica	Y
5	Times Roman	Y
6	Avant Garde Book	NA (Not Applicable)

Value	*Typeface*	*Boldface Type Available*
8	Palatino Roman	Y
11	Symbol	NA
12	Zapf Dingbats	NA
17	Zapf Chancery Medium Italic	NA
18	Bookman Light	NA
23	New Century Schoolbook Roman	Y
30	Courier Oblique	Y
31	Helvetica Oblique	Y
32	Times Italic	Y
33	Avant Garde Demi	NA
34	Avant Garde Book Oblique	NA
35	Avant Garde Demi Oblique	NA
36	Palatino Oblique	Y
37	New Century Schoolbook Italic	Y
38	Helvetica Narrow	Y
39	Helvetica Narrow Oblique	Y
40	Bookman Demi	NA
41	Bookman Light Italic	NA

Value	*Typeface*	*Boldface Type Available*
42	Bookman Demi Italic	NA

Other type faces can be added to the POSTSCRI.STR file.

Different fonts are available in SQR for Microsoft Windows when you are printing with Microsoft Windows printer drivers (using the -PRINTER:WP command-line flag). When you use the -PRINTER:WP flag, your report is sent directly to the default Microsoft Windows printer. To specify a nondefault Microsoft Windows printer, enter -PRINTER:WP:{printer name}. The {printer name} is the name assigned to your printer. For example, to send output to a Microsoft Windows printer named *NewPrinter*, you would use -PRINTER:WP:NewPrinter. If your printer name has spaces, enclose the entire argument in quotes.

Fonts are specified by number in the FONT qualifier of the ALTER-PRINTER command.

This table lists the fonts that are available when you are printing with Microsoft Windows printer drivers:

Value	*Windows Font/Name*	*Style*
3	Courier New	Fixed
300	Courier New	Bold
4	Arial	Proportional
400	Arial	Bold
5	Times New Roman	Proportional
500	Times New Roman	Bold
6	AvantGarde	Proportional
8	Palatino	Proportional
800	Palatino	Bold
11	Symbol	Proportional

Note. Fonts 6, 8, and 800 are not supplied with Microsoft Windows. You can get these fonts by purchasing the ADOBE Type Manager (ATM). The advantage of using ATM fonts is the compatibility for PostScript printer fonts. The Symbol font uses the SYMBOL_CHARSET instead of the usual ANSI_CHARSET character set. You can add more fonts by editing the appropriate Fonts section in the sqr.ini file.

See Chapter 7, "Using the PSSQR.INI File and the PSSQR Command Line," page 301.

Parameters

Parameter	*Description*
printer_name	A unique name to be used for referencing a printer definition and its attributes.

Note. The DECLARE-PRINTER Command Arguments table describes the other arguments of the DECLARE-PRINTER command. The table lists the options, default values, and description of each of the arguments.

Example

The following example illustrates the DECLARE-PRINTER command:

```
declare-printer HP-definition   ! Default HP definition

type=HP                         ! for all reports

font=4                          ! Helvetica

symbol-set=12U                  ! PC-850 Multilingual
end-declare
declare-printer PS-Sales        ! PS definition

for-reports=(sales)             ! for the Sales report

type=PS

font=5                          ! Times-Roman
end-declare
```

See Also

ALTER-PRINTER, DECLARE-REPORT

DECLARE-PROCEDURE

Syntax

```
DECLARE-PROCEDURE
[FOR-REPORTS=(report_name1[,report_namei]...)]
[BEFORE-REPORT=procedure_name[(arg1[,argi]...)]]
[AFTER-REPORT=procedure_name[(arg1[,argi]...)]]
[BEFORE-PAGE=procedure_name[(arg1[,argi]...)]]
[AFTER-PAGE=procedure_name[(arg1[,argi]...)]]
END-DECLARE
```


Description

Declares procedures that are triggered when a specified event occurs.

The DECLARE-PROCEDURE command can be used to define SQR procedures that are to be invoked before or after a report is printed or before the beginning or end of each page.

Issue the DECLARE-PROCEDURE in the SETUP section. For multiple reports, you can use the command as often as required to declare procedures required by all the reports. If you issue multiple DECLARE-PROCEDURE commands, the last one takes precedence. In this way, you can use one command to declare common procedures for ALL reports and others to declare unique procedures for individual reports. The referenced procedures can accept arguments.

If no FOR-REPORTS are specified, ALL is assumed. Initially, the default for each of the four procedure types is NONE. If a procedure is defined in one DECLARE-PROCEDURE for a report, that procedure is used unless NONE is specified.

Use the USE-PROCEDURE command to change the procedures to be used at runtime. To disable a procedure, specify NONE in the USE-PROCEDURE statement.

Parameters

Parameter	***Description***
FOR-REPORTS	Specifies one or more reports that use the given procedures. This argument is required only for a program with multiple reports. If you are writing a program that produces only a single report, you can ignore this argument.
BEFORE-REPORT	Specifies a procedure to be run when the first command that causes output to be generated (PRINT) is carried out. It can be used, for example, to create a report heading.
AFTER-REPORT	Specifies a procedure to be run just before the report file is closed at the end of the report. It can be used to print totals or other closing summary information. If no report was generated, the procedure does not run.
BEFORE-PAGE	Specifies a procedure to be run at the beginning of every page, just before the first output command for the page. It can be used, for example, to set up page totals.
AFTER-PAGE	Specifies a procedure to be run just before each page is written to the file. It can be used, for example, to display page totals. You can optionally specify arguments to be passed to any of the procedures. Arguments can be any variable, column, or literal.

Example

The following example illustrates the DECLARE-PROCEDURE command:

```
declare-procedure              ! These procedures will

before-report=report_heading   ! be used by all reports

after-report=report_footing
end-declare
declare-procedure             ! These procedures will

for-reports=(customer)        ! be used by the customer

before-page=page_setup        ! report

after-page=page_totals
end-declare
```

See Also

USE-PROCEDURE

DECLARE-REPORT

Syntax

```
DECLARE-REPORT report_name
[TOC=toc_name]
[LAYOUT=layout_name]
[PRINTER-TYPE=printer_type]
END-DECLARE
```

Description

Defines reports and their attributes.

Issue the DECLARE-REPORT in the SETUP section.

You can use the DECLARE-REPORT command to declare one or more reports to be produced in the application.

You must use this command when developing applications to produce more than one report.

Multiple reports can share the same layout and the same printer declarations or each report can use its own layout or printer definitions if the report has unique characteristics.

When you are printing multiple reports, unless you specify report names by using the -F command-line flag, the first report declared is generated with the name of program.lis, where *program* is the application name.

Additional reports are generated with names conforming to the rules dictated by the OUTPUT-FILE-MODE setting in the sqr.ini file.

When the -KEEP or -NOLIS flag is used, the first intermediate print file (.spf file) is generated with a name of program.spf and additional reports are generated with names conforming to the rules dictated by the OUTPUT-FILE-MODE setting in the sqr.ini file.

Parameters

Parameter	*Description*
report_name	Specifies the name of the report.
TOC	Specifies the name of the table of contents for this report.
LAYOUT	Specifies the name of the layout for this report. If no layout is specified, the default layout is used.
PRINTER-TYPE	Specifies the type of printer to be used for this report. If no printer type is specified, the default, LINEPRINTER, is used for this report. If no DECLARE-PRINTER is specified, DEFAULT-LP is used. Valid values for PRINTER-TYPE are HT, HP, PD, PS, LP, HTML, HPLASERJET, POSTSCRIPT, and LINEPRINTER.

Example

The following example illustrates the DECLARE-REPORT command:

```
declare-layout customer_layout
   left-margin
   right-margin
end-declare

declare-layout summary_layout
   orientation=landscape
end-declare

declare-report customer_detail

toc=detailed

layout=customer_layout

printer-type=postscript
end-declare
declare-report customer_summary

layout=summary_layout

printer-type=postscript
end-declare
.
.
.
use-report customer_detail

...print customer detail...
use-report customer_summary

...print customer summary...
```

See Also

USE-REPORT, DECLARE-LAYOUT, DECLARE-PRINTER, DECLARE-TOC

DECLARE-TOC

Syntax

```
DECLARE-TOC toc_name
[FOR-REPORTS=(report_name1[,report_namei]...)]
[DOT-LEADER=YES|NO]
[INDENTATION=position_count_num_lit]
[BEFORE-TOC=procedure_name[(arg1[,argi]...)]]
[AFTER-TOC=procedure_name[(arg1[,argi]...)]]
[BEFORE-PAGE=procedure_name[(arg1[,argi]...)]]
[AFTER-PAGE=procedure_name[(arg1[,argi]...)]]
[ENTRY=procedure-name [(argi [,argi] ...)]]
END-DECLARE
```

Description

Defines the table of contents and its attributes.

Use DECLARE-TOC in the SETUP section.

You can use the DECLARE-TOC command to declare one or more tables of contents for the application.

A table of contents can be shared between reports.

Parameters

Parameter	*Description*
toc_name	Specifies the name of the table of contents.
FOR-REPORTS	Specifies one or more reports that use this table of contents.
DOT-LEADER	Specifies whether a dot leader precedes the page number. The default setting is NO.
INDENTATION	Specifies the number of spaces by which each level is indented. The default setting is 4.
BEFORE-TOC	Specifies a procedure to be run before the application generates the table of contents. If no table of contents is generated, the procedure does not run.
AFTER-TOC	Specifies a procedure to be run after the application generates the table of contents. If no table of contents is generated, the procedure does not run.

Parameter	*Description*
BEFORE-PAGE	Specifies a procedure to be run at the start of every page.
AFTER-PAGE	Specifies a procedure to be run at the end of each page.
ENTRY	Specifies a procedure that is run to process each table of contents entry (instead of SQR doing it for you). When this procedure is invoked, the following SQR-reserved variables are populated with data about the TOC entry: #SQR-TOC-LEVEL contains the level. #SQR-TOC-TEXT contains the text. #SQR-TOC-PAGE contains the page number. These are global variables. If the procedure is local, you must precede it with an underscore (for example, #_sqr-toc-page). These three SQR-reserved variables are valid only within the scope of the ENTRY procedure. They can be referenced outside the scope, but their contents are undefined.

Example

The following example illustrates the DECLARE-TOC command:

```
begin-setup
  declare-toc common
    for-reports=(all)
    dot-leader=yes
    indentation=2
end-declare
end-setup
.
.
.
toc-entry level=1 text=$Chapter
toc-entry level=2 text=$Heading
.
.
```

See Also

BEGIN-FOOTING, BEGIN-HEADING, DECLARE-REPORT, TOC-ENTRY

DECLARE-VARIABLE

Syntax

```
DECLARE-VARIABLE
[DEFAULT-NUMERIC={DECIMAL[(prec_lit)]|FLOAT|INTEGER}]
[DECIMAL[(prec_lit)]num_var[(prec_lit)][num_var
[(prec_lit)]]...]
[FLOAT num_var[num_var]...]
[DATE date_var[date_var]...]
[INTEGER num_var[num_var]...]
[TEXT string_var[string_var]...]
END-DECLARE
```

Description

Enables you to explicitly declare a variable type.

You can set the default numeric type externally, using the -DNT command-line flag or the DEFAULT-NUMERIC setting in the Default-Settings section of the sqr.ini file. However, the setting in the DECLARE-VARIABLE command takes precedence over all other settings. If the command has not been used, then the -DNT command-line flag takes precedence over the setting in the sqr.ini file.

In addition to FLOAT, INTEGER, and DECIMAL, you can set DEFAULT-NUMERIC in the sqr.ini file and the -DNT command-line flag to V30. With V30, the program acts in the same manner as in versions prior to release 4.0; that is, all variables are FLOAT. V30 is not a valid setting for the DEFAULT-NUMERIC setting in the DECLARE-VARIABLE command.

The DECLARE-VARIABLE command enables you to determine the type of variables to use. This command can appear only in the SETUP section or as the first statement of a local procedure. The placement of the command affects its scope. When used in the SETUP section, it affects all variables in the entire program. Alternatively, when it is placed in a local procedure, its effect is limited to the scope of the procedure. If the command is in both places, the local declaration takes precedence over the SETUP declaration.

In addition to declaring variables, this command enables you to specify the default numeric type using the DEFAULT-NUMERIC setting as FLOAT, INTEGER, or DECIMAL. When dealing with money or when more precision is required, use the DECIMAL qualifier.

The DECLARE-VARIABLE command, the -DNT command-line flag, and the DEFAULT-NUMERIC setting in the sqr.ini file affect the way numeric literals are typed. If V30 is specified, then all numeric literals are FLOAT (just as in versions prior to release 4.0); otherwise, the use or lack of a decimal point determines the type of the literal as either FLOAT or INTEGER, respectively. Finally, not specifying the DECLARE-VARIABLE command, the -DNT command-line flag, and the DEFAULT-NUMERIC setting in the sqr.ini file is the same as specifying V30.

Parameters

Parameter	*Description*
DEFAULT-NUMERIC	Specifies the default type for numeric variables. Unless explicitly declared otherwise, a numeric variable assumes the variable type. This qualifier overrides any setting from the command-line flag -DNT or the DEFAULT- NUMERIC entry in the [Default-Settings] section of the sqr.ini file. If -DNT was not specified on the command line and the sqr.ini file entry has no DEFAULT-NUMERIC entry, then the default numeric type is FLOAT.
DECIMAL	Specifies that the numeric variables that follow are decimal variables with a precision specified with prec_lit. The precision can be assigned to the group of variables or to each individual variable. The precision is the total number of digits used to represent the number. This precision can range from 1 to 38. The default value is 16. The range of decimal numbers is from –9.99999999999999999999999999999999999E±4096 to +9.99999999999999999999999999999999999E±4096
FLOAT	Specifies that the numeric variables that follow are used as double-precision floating points. The range and precision of these numbers are machine-dependent.
DATE	Specifies that the date variables that follow can contain a date in the range of January 1, 4713 BC to December 31, 9999 AD.
INTEGER	Specifies that the numeric variables that follow are used as integers with a range of –2147483648 to +2147483647.
TEXT	Specifies that the string variables that follow are text variables.

Example

The following example illustrates the DECLARE-VARIABLE command:

```
begin-setup
  declare-variable
    default-numeric=float
    decimal  #decimal(10)
    integer  #counter
    date     $date
  end-declare
end-setup
.
.
let $date = strtodate('Jan 01 2004','Mon DD YYYY')
print $date (1,1)
position (+2,1)

let #counter = 0
while #counter < 10
 let #decimal = sqrt(#counter)
 add 1 to counter
 print #decimal  (+1,1) 9.999999999
end-while

do sub1($date, 'day', 10)

do sub2

.
.
begin-procedure sub1(:$dvar, $units, #uval)
declare-variable
 date $dvar
 integer #uval
end-declare
let $dvar = dateadd($dvar, $units, #uval)
print $dvar (+1,1)
position (+2,1)
end-procedure
.
.
begin-procedure sub2 LOCAL
declare-variable
 date $mydate
end-declare
let $mydate = dateadd($_date, 'year', 5)
print $mydate (+1,1)
position (+2,1)
end-procedure
.
.
```

#DEFINE

Syntax

#DEFINE *substitution_variablevalue*

Description

Declares a value for a substitution variable within the body of the report (rather than using the ASK command).

#DEFINE is useful for specifying constants such as column locations, printer fonts, or any number or string that is used in several locations in the program. When the value of the number or string must be changed, you need only change the #DEFINE command. All references to that variable change automatically, which makes modifying programs much simpler.

If the ASK command is used to obtain the value of a substitution variable that has already been defined, ASK uses the previous value and the user is not prompted. This enables you to predefine some variables and not others. When the report runs, ASK requests values for only those variables that have not had a value assigned.

You can use #DEFINE commands inside an include file. This is a method of gathering commonly used declarations into one place, and reusing them for more than one report.

The value in the #DEFINE command can have embedded spaces and does not need to be enclosed within quotes. The entire string is used as is.

The #DEFINE command cannot be broken across program lines.

Parameters

Parameter	***Description***
substitution_variable	The variable to be used as the substitution variable. The substitution variable is used to substitute any command, argument, or part of a SQL statement at compile time.
Value	The value to be substituted.

Example

This code example defines several constants:

```
#define  page_width  8.5
#define  page_depth  11
#define  light LS^10027
#define  bold  LS^03112
#define  col1  1
#define  col2  27
#define  col3  54
#define  order_by  state, county, city, co_name
```

This code example from a report uses the definitions from the preceding example:

```
   begin-setup

declare-printer contacts

   type=hp

   paper-size=({page_width}, {page_depth})

end-declare
end-setup
begin-heading 5
  print 'Company Contacts'     (1,1)  center
  print 'Sort:  {order_by}'    (2,1)  center
  print 'Company'              (4,{col1})
  print 'Contact'              (4,{col2})
  print 'Phone'                (4,{col3})
end-heading
begin-procedure main
begin-select
company      (1,{col1})
  print '{bold}'      (0,{col2})         ! Print contact in boldface.
contact         ()
  print '{light}'         ()            ! Back to lightface.
phone         (0,{col3})                ! Note: There must be enough
  next-listing                          ! space between col2
from customers                          ! and col3 for both
order by  {order_by}                    ! font changes and the
end-select                              ! contact field.
end-procedure
```

See Also

ASK

DISPLAY

Syntax

```
DISPLAY {any_lit|_var|_col}
[[:$]edit_mask|NUMBER|MONEY|DATE][NOLINE]
```

Description

Displays the specified column, variable, or literal.

The DISPLAY command can display data to a terminal. The data is displayed to the current location on the screen. If you want to display more than one field on the same line, use NOLINE on each display except the last.

Dates can be contained in a date variable or column, or a string literal, column, or variable. When a date variable or column is displayed without an edit mask, the date appears in the following manner:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the TIME Column Formats table.

When the program displays a date in a string literal, column, or variable using EDIT or DATE, the string uses the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats as listed in the Default Database Formats table, or the database-independent format SYYYYMMDD[HH24[MI[SS[NNNNNN]]]].

If you require more control over the display, use the SHOW command.

Parameters

Parameter	***Description***
any_lit|_var|_col	The text, number, or date to be displayed.
edit_mask	Causes the field to be edited before being displayed. For additional information regarding edit masks, see the PRINT command.
NUMBER	Indicates that any_lit|_var|_col is to be formatted using the NUMBER-EDIT-MASK of the current locale. This option is not valid with date variables.
MONEY	Indicates that any_lit|_var|_col is to be formatted using the MONEY-EDIT-MASK of the current locale. This option is not valid with date variables.
DATE	Indicates that any_lit|_var|_col is to be formatted using the DATE-EDIT-MASK of the current locale. This option is not valid with numeric variables. If DATE-EDIT-MASK has not been specified, the date is displayed by the default format for that database (see the Default Database Formats table).
NOLINE	Suppresses the carriage return after the field is displayed.

Example

The following segments illustrate the various features of the DISPLAY command:

```
!
! Display a string using an edit mask
!
display '123456789' xxx-xx-xxxx
```

Produces the following output:


```
123-45-6789


!
 ! Display a number using an edit mask
 !
 display 1234567.89 999,999,999.99
```

Produces the following output:

```
1,234,567.89

!
 ! Display a number using the default edit mask (specified in SQR.INI)
 !
 display 123.78
```

Produces the following output:

```
123.780000

!
 ! Display a number using the locale default numeric edit mask
 !
 alter-locale  number-edit-mask = '99,999,999.99'
 display 123456.78 number
```

Produces the following output:

```
123,456.78

!
 ! Display a number using the locale default money edit mask
 !
 alter-locale  money-edit-mask = '$$,$$$,$$9.99'
 display 123456.78 money
```

Produces the following output:

```
$123,456.78

!
 ! Display a date column using the locale default date edit mask
 !
 begin-select
 dcol
  from tables
 end-select
 alter-locale  date-edit-mask = 'DD-Mon-YYYY'
 display &dcol date
```

Produces the following output:

```
01-Jan-2004

!
 ! Display two values on the same line
 !
 display 'Hello' noline
 display ' World'
```

Produces the following output:

```
Hello World

!
 ! Display two values on the same line with editing of the values
 !
 alter-locale  money-edit-mask = '$$,$$$,$$9.99'
 let #taxes = 123456.78
 display 'You owe ' noline
 display #taxes money noline
 display ' in back taxes.'
```

Produces the following output:

```
You owe $123,456.78 in back taxes.
```

See Also

The SHOW command for information about screen control.

The LET command for information about copying, editing, or converting fields.

The EDIT parameter of the PRINT command for a description of the edit masks.

The ALTER-LOCALE command for a description of the arguments NUMBER-EDIT-MASK, MONEY-EDIT-MASK, and DATE-EDIT-MASK.

DIVIDE

Syntax

```
DIVIDE {src_num_lit|_var|_col} INTO dst_num_var
[ON-ERROR={HIGH|ZERO}][ROUND=nn]
```

Description

Divides one number into another.

The source field is divided into the destination field and the result is placed in the destination. The source is always first, the destination always second.

When dealing with money-related values (dollars and cents), use decimal variables rather than float variables. Float variables are stored as double-precision floating-point numbers, and small inaccuracies can appear when many numbers are divided in succession. These inaccuracies can appear due to the way different hardware and software implementations represent floating point numbers.

Parameters

Parameter	*Description*
src_num_lit|_var|_col	Divided into the contents of dst_num_var.

Parameter	*Description*
dst_num_var	Contains the result after execution.
ON-ERROR	Sets the result to the specified number when a division by zero is attempted. If ON-ERROR is omitted and a division by zero is attempted, SQR halts with an error message.
ROUND	Rounds the result to the specified number of digits to the right of the decimal point. For float variables, this value can be from 0 to 15. For decimal variables, this value can be from 0 to the precision of the variable. For integer variables, this argument is not appropriate.

Example

The following example illustrates the DIVIDE command:

```
divide  37.5  into  #price   ! #price / 37.5
divide  &rate  into  #tot  on-error=high
divide  #j  into  #subtot  on-error=zero
```

Note. In the preceding example, `High` is the maximum value and zero is the lowest value.

See Also

ADD

The LET command for a discussion of complex arithmetic expressions.

DO

Syntax

`DO` *`procedure_name`*`[(`*`arg1`*`[, `*`argi`*`]...)]`

Description

Invokes the specified procedure.

When the procedure ends, processing continues with the command following the DO command. You can use arguments to send values to or receive values from a procedure.

Arguments passed by a DO command to a procedure must match in number:

- Database text columns, string variables, and literals can be passed to procedure string or date arguments.
- Database numeric columns, numeric variables, and numeric literals can be passed to procedure numeric arguments.

- Numeric variables (DECIMAL, INTEGER, FLOAT) can be passed to procedure numeric arguments without regard to the argument type of the procedure.

  SQR automatically converts the numeric values upon entering and leaving the procedure as required.

- Date variables can be passed to procedure date or string arguments.

When a field in a DO command receives a value back from a procedure (a colon indicates that it is a back value, that is, a value that is being returned), it must be a string, numeric, or date variable, depending on the procedure argument; however, a date can be returned to a string variable and vice versa.

When a date is passed to a string, the date is converted to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Parameters

Parameter	***Description***
procedure_name	Specifies the name of the procedure to be run.
arg1 [, argi]	Specifies the arguments to be passed to the procedure. Arguments can be any type of variable or constant value.

Example

The following example illustrates the DO command:

```
do get_names
do add_to_list ($name)
do print_list ('A', #total, &co_name, $name)
```

See Also

The BEGIN-PROCEDURE command for information about passing arguments.

The PRINT command for information about date and time formats.

#ELSE

Syntax

```
#ELSE
```

Description

Compiles the code following the #ELSE command when a preceding #IF, #IFDEF, or #IFNDEF command is FALSE. (#ELSE is a compiler directive that works with the #IF, #IFDEF, and #IFNDEF compiler directives.)

See Also

The #IF, #IFDEF, and #IFNDEF commands for a description of each compiler directive.

ELSE

Syntax

```
ELSE
```

Description

ELSE is an optional command in an IF command.

See Also

The IF command for a description and example.

ENCODE

Syntax

ENCODE *src_code_string_lit* INTO *dst_txt_var*

Description

Assigns a nondisplay or display character to a string variable.

The ENCODE command can define nondisplay characters or escape sequences sent to an output device. These characters or sequences can perform complex output device manipulations. The ENCODE command also displays characters not on the keyboard. If your keyboard does not have the euro symbol, use the ENCODE feature to create a string variable for it.

The encode characters can be included in a report at the appropriate location using a PRINT or PRINT-DIRECT command.

Unicode (UCS-2) code that points from <1> to <65535> can be defined in the ENCODE command.

Parameters

Parameter	*Description*
src_code_string_lit	Specifies a string of characters to be encoded and placed in *dst_txt_var*.
dst_txt_var	Contains the result after execution.

Example

The following example illustrates the ENCODE command:

```
encode '<27>L11233' into $bold            ! Code sequence to turn bold on.
print $bold () code-printer=lp
```

See Also

The chr function described in the Miscellaneous Functions table under the LET command.

PRINT, PRINT-DIRECT

END-DECLARE, END-DOCUMENT, END-EVALUATE, END-FOOTING, END-HEADING

Syntax

```
END-DECLARE
END-DOCUMENT
END-EVALUATE
END-FOOTING
END-HEADING
```

Description

Completes a section or paragraph.

The END-DECLARE command completes a paragraph started with:


```
DECLARE-CHART
DECLARE-IMAGE
DECLARE-LAYOUT
DECLARE-PRINTER
DECLARE-PROCEDURE
DECLARE-REPORT
DECLARE-VARIABLE
```

Other END- commands complete the corresponding BEGIN- command:

```
BEGIN-DOCUMENT
EVALUATE
BEGIN-FOOTING
BEGIN-HEADING
```

Each command must begin on its own line.

Example

The following example illustrates the BEGIN-FOOTING and END-FOOTING commands:

```
begin-footing  2
   print  'Company Confidential'  (1)  center
end-footing
```

See Also

DECLARE-paragraph, BEGIN-section

#END-IF, #ENDIF

Syntax

```
#END-IF
```

Description

Ends an #IF, #IFDEF, or #IFNDEF command. (#END-IF is a compiler directive.)

#ENDIF (without the hyphen) is a synonym for #END-IF.

Example

The following example illustrates the #END-IF compiler directive:

```
#ifdef debuga
  show  'DebugA: #j = '  #j  edit  9999.99
  show  'Cust_num   = '  &cust_num
#end-if
```


See Also

The #IF, #IFDEF, and #IFNDEF commands for a description of each compiler directive.

END-IF

Syntax

```
END-IF
```

Ends an IF command.

See Also

The IF command for a description and example.

END-PROCEDURE, END-PROGRAM, END-SELECT, END-SETUP, END-SQL, END-WHILE

Syntax

```
END-PROCEDURE
END-PROGRAM
END-SELECT
END-SETUP
END-SQL
END-WHILE
```

Description

Completes the corresponding section or paragraph.

Each END- command completes the corresponding BEGIN- command:

```
BEGIN-PROCEDURE
BEGIN-PROGRAM
BEGIN-SELECT
BEGIN-SETUP
BEGIN-SQL
WHILE
```

Each command must begin on its own line.

Example

The following example illustrates the END-PROGRAM command:

```
begin-program
   do main
end-program
```

See Also

BEGIN-section, WHILE

EVALUATE

Syntax

```
EVALUATE {any_lit|_var|_col}
```

This command is equivalent to case/switch in C or Java. The general format of an EVALUATE command is:

```
EVALUATE {any_lit|_var|_col}
WHEN comparison_operator {any_lit|_var|_col}
SQR_Command...
[BREAK]
[WHEN comparison_operator {any_lit|_var|_col}
SQR_Command...
[BREAK]]
[WHEN-OTHER
SQR_Command...
[BREAK]]
END-EVALUATE
```

Description

Determines the value of a column, literal, or variable and takes action based on that value.

The EVALUATE command is useful for branching to different commands depending on the value of a specified variable or column.

EVALUATE commands can be nested.

Evaluating a date variable or column with a string results in a date comparison (chronological, not a byte-by-byte comparison as is done for strings). The string must be in the proper format as shown in the following list:

- For DATETIME columns and SQR DATE variables in the format specified by the SQR_DB_DATE_FORMAT setting, SQR uses one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.
- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting, or the format listed in the DATE Column Formats table.
- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting, or the format as listed in the TIME Column Formats table.

Parameters

Parameter	*Description*
any_lit|_var|_col	Specifies a text or numeric column; a text, numeric, or date variable; or a text or numeric literal to be used in the evaluation. In short, this is an evaluation argument.
comparison_operator	Any valid comparison operator. See the list of operators in the #IF command. See Chapter 2, "SQR Command Reference," #IF, page 135.
WHEN	Specifies the evaluation expression. The evaluation argument is compared with the argument, beginning from the first WHEN. If the expression is TRUE, SQR processes the commands after the WHEN. If the expression is FALSE, SQR processes the next WHEN expression. Each WHEN must be on its own line. If more than one WHEN expression appears directly before a set of commands, any one of them, if TRUE, causes the commands to be carried out.
BREAK	Causes an immediate exit of the EVALUATE command. Use BREAK at the end of a set of commands.
WHEN-OTHER	Signifies the start of default commands to be processed if all other WHEN expressions are FALSE. WHEN-OTHER must appear after all other WHEN expressions.

Example

The following example illustrates the EVALUATE command:

```
evaluate &code
  when = 'A'
     move 1 to #j
     break
  when = 'B'
  when = 'C'
     move 2 to #j     ! Will happen if &code is B or C.
     break
  when > 'D'
     move 3 to #j     ! Move 3 to #j and continue checking.
  when > 'H'
     add 1 to #j      ! Add 1 to #j and continue checking.
  when > 'W'
     add 2 to #j
     break
  when-other
     if isnull (&code)
         do null_code
      else
         move 0 to #j  ! Unknown code.
     end-if
     break
end-evaluate
```

See Also

The commands IF and LET for comparison operators.

EXECUTE (Sybase and Microsoft SQL Server)

Syntax

```
EXECUTE [-XC][ON-ERROR=procedure[(arg1[,argi]...)]]
[DO=procedure[(arg1[,argi]...)]]
[@#status_var=]stored_procedure_name
[[@param=]{any_col|_var|_lit}[,...]]
[INTO any_coldata_type[(length_int_lit)]
[,...]][WITH RECOMPILE]
```

The syntax of this command generally follows that of the Sybase Transact-SQL EXECUTE command, with the exception of optional arguments and the INTO argument.

Description

Runs a stored procedure in Sybase or Microsoft SQL Server database.

If the stored procedure specified in **stored_procedure_name** contains a SELECT query, the EXECUTE command must specify an INTO argument to process the values from the query. If no INTO argument is specified, then the values from the query are ignored.

EXECUTE retrieves just the first row when the following conditions are met:

- The DO procedure is not specified.
- The stored procedure, **stored_procedure_name,** selects one or more rows.
- An INTO argument is specified.

This is useful for queries returning a single row.

Parameters

Parameter	*Description*
-XC	(Sybase only) Specifies that the EXECUTE command share the same connection as the DO=procedure it can invoke. This argument is required to share Sybase temporary tables.

Parameter	*Description*
ON-ERROR	Declares an SQR procedure to run if an error occurs. If ON-ERROR is omitted and an error occurs, SQR halts with an error message. For severe errors (for example, passing too few arguments) SQR halts, even if an error procedure is specified. You can specify arguments to be passed to the ON-ERROR procedure. Arguments can be any variable, column, or literal.
DO	Specifies an SQR procedure to run for each row selected in the query. Processing continues until all rows have been retrieved. You can specify arguments to be passed to the procedure. Arguments can be any variable, column, or literal.
@#status_variable	Returns the procedure status in the specified numeric variable. The status is returned only after selected rows are retrieved.
stored_procedure_name	Names the stored procedure to run.
@param	Names the parameter to pass to the stored procedure. Parameters can be passed with or without names. If used without names, they must be listed in the same sequence as defined in the stored procedure.
any_lit|_var|_col	Specifies the value passed to the stored procedure. It can be a string, numeric, or date variable, a previously selected column, a numeric literal, or a string literal.
OUT[PUT]	Indicates that the parameter receives a value from the stored procedure. The parameter must be a string, numeric, or date SQR variable. Output parameters receive their values only after rows selected have been retrieved. If you specify multiple output parameters, they must be in the same sequence as defined in the stored procedure.
INTO	Indicates where to store rows that are retrieved from the stored procedure's SELECT statement. The INTO argument contains the names of the columns with data types and lengths (if needed). You must specify the columns in the same sequence and match the data type used in the stored procedure's SELECT statement. If the stored procedure contains more than one SELECT query, only the first query is described with the INTO argument. Rows from subsequent queries are ignored.
WITH RECOMPILE	Causes the query to recompile each time it is run rather than using the plan stored with the procedure. Normally, this is not required or recommended.

Example

The following code example invokes the stored procedure **get_total** with two parameters: a string literal and a string variable. The result from the stored procedure is stored in the variable *#total*.

```
execute get_total 'S. Q. Reporter' $State #Total Output
```

The following code example invokes the stored procedure **get_products** with two parameters. The stored procedure selects data into five column variables. The SQR procedure **print_products** is called for each row retrieved. The return status from the stored procedure is placed in the variable *#proc_return_status*.

```
execute do=print_products
   @#proc_return_status=
   get_products
   @prodcode=&code,            @max=#maximum
   INTO &prod_code         int,
      &description         char (45),
      &discount         float,
      &restock         char,
      &expire_date         datetime

begin-procedure print_products
print &prod_code            (+1,1)
print &description            (+5,45)
print &discount            (+5) edit 99.99
print &restock            (+5) match Y 0 5 Yes N 0 5 No
print &expire_date            (+5,) edit 'Month dd, yyyy'
end-procedure
```

EXIT-SELECT

Syntax

```
EXIT-SELECT
```

Description

Exits a SELECT paragraph immediately.

EXIT-SELECT causes SQR to jump to the command immediately following the END-SELECT command.

Use EXIT-SELECT when you need to end a query before all rows have been retrieved.

Example

The following example illustrates the EXIT-SELECT command:

```
begin-select
cust_num, co_name, contact, city, state, zip, employees
  add &employees to #tot_emps
  if #tot_emps >= 5000
    exit-select         ! Have reached required total emps.
  end-if
  do print_company
from customers order by employees desc
end-select
```

See Also

BEGIN-SELECT

EXTRACT

Syntax

```
EXTRACT {dst_txt_var|date_var} FROM
{{src_txt_lit|_var|_col}|{src_date_var|_col}}
{start_num_lit|_var}{length_num_lit|_var}
```

Description

Copies a portion of a string into a string variable.

You must specify the starting location of the string as an offset from the beginning of the string and its length. An offset of 0 (zero) begins at the leftmost character; an offset of 1 begins one character beyond that, and so on.

If the source is a date variable or column, it is converted to a string before the extraction according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR specifies the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

If the destination is a date variable, the string extracted from the source must be in one of the following formats:

- The format specified by the SQR_DB_DATE_FORMAT setting.
- One of the database-dependent formats (see the Default Database Formats table).
- The database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.

Parameters

Parameter	*Description*
dst_txt_var | date_var	Specifies a text or date variable into which the extracted string is placed.
{src_txt_lit|_var|_col}| {src_date_var|_col}	Specifies a text or date variable, column, or literal from which the string is to be extracted.

Parameter	*Description*
start_num_lit|_var	Specifies starting location of the string to be extracted.
length_num_lit|_var	Specifies length of the string to be extracted.

Example

The following example illustrates the EXTRACT command:

```
extract  $state  from  $record  45  2
extract  $foo from "SQR Rocks" 0 4 ! $foo='SQR'

code  from  &phone  0  3
extract  $zip_four  from  &zip  5  4
extract  $rec  from  $tape_block  #loc  #rec_len
```

Note. Oracle recommends that you not use the EXTRACT command when processing strings.

See Also

The substr function described in the Miscellaneous Functions table under the LET command.

FIND

The PRINT command for information about default date and time formats

FIND

Syntax

```
FIND {{obj_txt_lit|_var|_col}|{date_var|_col}} IN
{{src_txt_var|_col}|{date_var|_col}}
{start_int_lit|_var} dst_location_int_var
```

Description

Determines the location of a character sequence within a string.

FIND searches the specified string for a character sequence and, if the string is found, returns its location as an offset from the beginning of the specified string. If the sequence is not found, FIND returns –1 in *dst_location_int_var*.

You must specify an offset from which to begin the search and supply a numeric variable for the return of the location.

If the source or search object is a date variable or column, it is converted to a string before the search according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Parameters

Parameter	***Description***
{obj_txt_lit|_var|_col}|{date_var|_col}	Specifies a text variable, column, or literal that is to be sought in *src_txt_var|_col*.
{src_txt_var|_col}| {date_var|_col}	Specifies a text variable or column to be searched.
start_int_lit|_var	Specifies the starting location of the search.
dst_location_int_var	Specifies the returned starting location of the leftmost character of the matching text in {*src_txt_var*|*_col*| *date_var*|*_col* }.

Example

The following example illustrates the FIND command:

```
find  'aw.2'  in  &code5  0  #loc
find  ','  in  &name  0  #comma_loc
if #comma_loc = -1
   ...comma not found...
```

See Also

The instr function described in the Miscellaneous Functions table under the LET command.

EXTRACT

The PRINT command for information about default date and time formats.

GET

Syntax

```
GET dst_any_var...FROM src_array_name(element)
[field[(occurs)]]...
```

Description

Retrieves data from an array and places it into a date, string, or numeric variable.

Parameters

Parameter	*Description*
dst_any_var	Date, string, or numeric variables (not database columns) can be destination variables. Numeric variables (decimal, float, integer) are copied from number fields. String variables are copied from the char, text, or date field. Date variables are copied from the char, text, or date field. When a date field is copied to a string variable, SQR converts the date to a string in the format specified by the SQR_DB_DATE_FORMAT setting. If this has not been set, SQR uses the first database-dependent format listed in the Default Database Formats table. If the destination is a date variable, the string extracted from the source must be in the format specified by the SQR_DB_DATE_FORMAT setting, or one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.
src_array_name(element)	If the array's field names are listed, SQR takes the values from the fields and occurrences specified. If the array's field names are not listed, the values are taken from consecutively defined fields in the array.
field[(occurs)]	Array element and field occurrence numbers can be numeric literals (such as 123) or numeric variables (such as #j). If no field occurrence is stated, occurrence zero is used.

Example

The following code example copies *$name,$start_date,* and *#salary* from the first three fields in the *#j*th element of the *emps* array:

```
get $name $start_date #salary from emps(#j)
```

The following code example copies *#city_tot* and *#county_tot* from the fields *cities* and *counties* in the *#j*th element of the *states* array:

```
get #city_tot #county_tot from states(#j) cities counties
```

The following code example copies *$code* from the *#j*th occurrence of the *code* field in the *#n*th element of the *codes* array:

```
get $code from codes(#n) code(#j)
```

See Also

The PUT command for information about moving data into an array.

GET-COLOR

Syntax

```
GET-COLOR
[PRINT-TEXT-FOREGROUND=({color_name_var |{rgb})]
[PRINT-TEXT-BACKGROUND=({color_name_var |{rgb})]
```

Description

Retrieves the current colors.

The GET-COLOR command is allowed wherever the PRINT command is allowed. If the requested color settings do not map to a defined name, the name is returned as *RGBredgreenblue,* where each component is a three-digit number—for example, *RGB127133033.* You can use this format wherever you use a color name. The color name 'none' is returned if no color is associated with the specified area.

Parameters

Parameter	*Description*
PRINT-TEXT-FOREGROUND	Defines the color in which the text prints.
PRINT-TEXT-BACKGROUND	Defines the color to print as a background behind the text.
{*color_name_var*}	A color_name is composed of alphanumeric characters (A–Z, 0–9), the underscore (_) character, and the hyphen (-) character. The name must start with an alphabetical (A–Z) character and is not case-sensitive. The name 'none' is reserved and cannot be assigned a value. A name in the format (RGBredgreenblue) cannot be assigned a value. The name 'default' is reserved and can be assigned a value. 'Default' is used during execution when a referenced color is not defined in the runtime environment.

Parameter	*Description*
{*rgb*}	*red_lit* | *_var* | *_col, green_lit* | *_var* | *_col, blue_lit* | *_var* | *_col* where each component is a value in the range of 000 to 255. In the BEGIN-SETUP section, only literal values are allowed. The default colors implicitly installed with SQR include: black=(0,0,0) white=(255,255,255) gray=(128,128,128) silver=(192,192,192) red=(255,0,0) green=(0,255,0) blue=(0,0,255) yellow=(255,255,0) purple=(128,0,128) olive=(128,128,0) navy=(0,0,128) aqua=(0,255,255) lime=(0,128,0) maroon=(128,0,0) teal=(0,128,128) fuchsia=(255,0,255)

Example

The following example illustrates the GET-COLOR command:

```
begin-setup
   declare-color-map
   light_blue = (193, 222, 229)
   end-declare
end-setup

begin-program
   alter-color-map name = 'light_blue' value = (193, 233, 230)

   print 'Yellow Submarine' ()
      foreground = ('yellow')
      background = ('light_blue')

   get-color print-text-foreground = ($print-foreground)
   set-color print-text-foreground = ('purple')
   print 'Barney' (+1,1)
   set-color print-text-foreground = ($print-foreground)
end-program
```

See Also

DECLARE-COLOR-MAP, ALTER-COLOR-MAP, SET-COLOR

GOTO

Syntax

```
GOTO  label
```

Description

Skips to the specified label.

Labels must end with a colon (:) and can appear anywhere within the same section or paragraph as the GOTO command.

Parameters

Parameter	*Description*
label	Specifies a label within the same section or paragraph.

Example

The following example illustrates the GOTO command:

```
begin-select
price
 if &price < #old_price
    goto next
 end-if
 print &price (2,13,0) edit 999,999.99
   ...
next:
 add 1 to #count
from products
end-select
```

GRAPHIC BOX, GRAPHIC HORZ-LINE, GRAPHIC VERT-LINE

Syntax

The GRAPHIC commands have the following syntax:


```
GRAPHIC ({line_int_lit|_var},{column_int_lit|_var},
{width_int_lit|_var})

BOX {depth_int_lit|_var}
[rule_width_int_lit|_var[shading_int_lit|_var]]

GRAPHIC ({line_int_lit|_var},{column_int_lit|_var},
{length_int_lit|_var})

HORZ-LINE [rule_width_int_lit|_var]

GRAPHIC ({line_int_lit|_var},{column_int_lit|_var},
{length_int_lit|_var})

VERT-LINE [rule_width_int_lit|_var]
```

Description

Draws a box or line.

After GRAPHIC commands are carried out, SQR changes the current print location to the starting location of the graphic. This is different from the way the PRINT command works.

The GRAPHIC command has the following variations:

- BOX
- HORZ-LINE
- VERT-LINE

The following sections describe the individual GRAPHIC commands:

Parameters

Parameter	***Description***
BOX	BOX draws a box of any size at any location on the page. Boxes can be drawn with any size rule and can be shaded or left empty.
width and *depth*	The width is the horizontal size in character columns; depth is the vertical size in lines. The top left corner of the box is drawn at the line and column specified. The bottom right corner is calculated by the width and depth. You can specify relative placement with (+), (-), or numeric variables, as with regular print positions.
rule_width	The default rule width is 2 decipoints (an inch has 720 decipoints). The top horizontal line is drawn just below the base of the line above the starting point. The bottom horizontal line is drawn just below the base of the ending line. Therefore, a one-line deep box surrounds a single line.
shading	A number between 1 and 100, specifying the percentage of shading to apply. 1 is very light, and 100 is black. If no shading is specified, the box is blank. Specify a rule-width of zero if you do not want a border.

Parameter	*Description*
HORZ-LINE	HORZ-LINE draws a horizontal line from the location specified, for the length specified. Horizontal lines are drawn just below the base.
rule_width	The default rule width is 2 decipoints.
VERT-LINE	VERT-LINE draws a vertical line from the location specified for the length (in lines) specified. Vertical lines are drawn just below the base line of the line position specified to just below the base line of the line reached by the length specified. To draw a vertical line next to a word printed on line 27, position the vertical line to begin on line 26, for a length of 1 line.
rule_width	The default rule width is 2 decipoints.

Example

The following code example shows the GRAPHIC BOX command:

```
graphic (1,1,66) box 58 20     ! Draw box around page
graphic (30,25,10) box 10      ! Draw a 10-characters-wide-by-10- characters-long⇒
 box
graphic (1,1,66) box 5 0 8     ! Draw 5 line shaded box (without border)
graphic (50,8,30) box 1        ! Draw box around 1 line
```

The following code example shows the GRAPHIC HORZ-LINE command:

```
graphic (4,1,66) horz-line 10  ! Put line under page heading
graphic (+1,62,12) horz-line   ! Put line under final total
```

The following code example shows the GRAPHIC VERT-LINE command:

```
graphic (1,27,54) vert-line    ! Draw lines between columns
graphic (1,52,54) vert-line
graphic (3,+2,4) vert-line 6   ! Red line the paragraph
```

See Also

The ALTER-PRINTER and DECLARE-PRINTER commands for information about setting and changing the FONT, FONT-TYPE, POINT-SIZE, and PITCH.

#IF

Syntax

```
#IF {txt_lit|num_lit}comparison_operator
{txt_lit|num_lit}
```

Description

Indicates that the commands following #IF are to be compiled when the expression is TRUE. (#IF is a compiler directive.)

SQR has five compiler directives that enable different pieces of SQR code to be compiled, depending on the existence or value of substitution variables (not program variables, such as string, numeric, or date).

Substitution variables defined automatically for each -DEBUGxxx letter can also be used with the #IF, #IFDEF, and #IFNDEF directives. They can enable or disable entire sections of an SQR program from the command line, depending on the -DEBUGxxx flag.

You can nest #IF, #IFDEF, and #IFNDEF directives to a maximum of 10 levels.

The #IF, #IFDEF, and #IFNDEF directives cannot be broken across program lines.

The following table lists the compiler directives.

Directive	***Example***	***Description***
#IF	#IF {option}='A'	Compiles the commands following the #IF directive if the substitution variable option is equal to 'A'. The test is not case-sensitive. Only one simple expression is allowed per #IF command.
#ELSE	#ELSE	Compiles the commands following the #ELSE directive when the #IF expression is FALSE.
#ENDIF	#ENDIF	Ends the #IF directive. #ENDIF can also be typed #END-IF (with a hyphen).
#IFDEF	#IFDEF option	Compiles the commands following the #IFDEF directive if the substitution variable option is defined.
#IFNDEF	#IFNDEF option	Compiles the command following the #IFNDEF directive if the substitution variable option is not defined.

Parameters

Parameter	*Description*
txt_lit | num_lit	Any text or numeric literal.
comparison_operator	Any of the comparison operators as shown here: =Equal !=Not Equal <>Not Equal <Less than >Greater than <=Less than or equal >=Greater than or equal

Example

The following example illustrates the #IF compiler directive:

```
begin-setup
  ask type 'Use Male, Female or Both (M,F,B)'
end-setup
begin-procedure Main
#if {type} = 'M'
  ...code for M here
#else
#if {type} = 'F'
  ...code for F here
#else
#if {type} = 'B'
  ...code for B here
#else
  show 'M, F or B not selected.   Report not created.'
  stop
#endif   ! for B
#endif   ! for F
#endif   ! for M

#ifdef  debug
  show 'DEBUG:  Cust_num = '  &cust_num  edit  099999
#endif

#ifndef debugB            ! DebugB turned on with -DEBUGB on
  do test_procedure       ! SQR command line.
#endif
```

See Also

The #DEBUG command for information about the -DEBUG command-line flag.

IF

Syntax

```
IF logical_expression
```

IF commands have the following structure:

```
IF logical_expression
SQR Command...
[ELSE
SQR Command...]
END-IF
```

Description

Carries out commands depending on the value of a condition.

The expression is evaluated as a logical TRUE or FALSE. A value or expression that evaluates to nonzero is TRUE.

Each IF command must have a matching END-IF command.

IF commands can be nested.

Comparing a date variable or column with a string results in a date comparison (chronological, not a byte-by-byte comparison as is done for strings). The string must be in the proper format as described in the following list:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.
- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting, or the format listed in the Date Column Formats table.
- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting, or the format as listed in the Time Column Formats table.

Parameters

Parameter	*Description*
logical_expression	Any valid logical expression. See the LET command for a description of logical expressions.

Example

The following example illustrates the IF command:

```
if &price > &old_price and instr(&code, 'M', 1) > 0
   add 1 to #price_count
   if #price_count > 50
      show 'More than 50 prices found.' noline
      input $x 'Continue? (Y/N)'
      if upper($x) = 'N'
         stop
      end-if
   end-if
else
   add 1 to #old_price_count
end-if
if #rows   ! Will be TRUE if #rows is non-zero.
   do print-it
end-if

if $date1 > 'Apr 21 2004 23:59'
   do past_due
end-if
```

See Also

The LET command for a description of logical expressions.

EVALUATE

#IFDEF

Syntax

`#IFDEF` *`substitution_variable`*

Description

Indicates that the following commands are to be compiled when the substitution variable has been declared by an ASK or #DEFINE command, or by the -DEBUG flag on the SQR command line. (#IFDEF is a compiler directive.)

Parameters

Parameter	*Description*
substitution_variable	The variable to be used as the substitution variable.

See Also

The #IF command for a description of each compiler directive.

#IFNDEF

Syntax

```
#IFNDEF substitution_variable
```

Description

Indicates that the following commands are to be compiled when the substitution variable has not been declared by an ASK or #DEFINE command, or by the -DEBUG flag on the SQR command line. (#IFNDEF is a compiler directive.)

Parameters

Parameter	*Description*
substitution_variable	The variable to be used as the substitution variable.

See Also

The #IF command for a description of each compiler directive.

#INCLUDE

Syntax

```
#INCLUDE filename_lit
```

Description

Includes an external source file in the SQR report specification.

You may want to keep commonly used routines in a single file and reference or *include* that file in programs that use the routine. For example, you might have a set of #DEFINE commands for different printers to control initialization, font changes, and page size declarations. You can reference the appropriate include file depending on which printer you want to use.

Include files can be nested up to four levels. Variable substitution scanning takes place before the #INCLUDE command is processed. This enables you to substitute all or part of the include file name at runtime, adding flexibility for controlling which file is included for the run.

Parameters

Parameter	*Description*
filename_lit	A file name that is valid for the platform on which the application is to be compiled.

Example

The following example illustrates the #INCLUDE command:

```
#include 'gethours.dat'         ! Common procedure.
#include 'XYZheader.dat'        ! Common report heading for XYZ Company.
#include 'printer{num}.dat'     ! Include printer definitions for
                                ! printer {num}, which is passed
                                ! on the command line:
                                ! SQR REP1A SAM/JOE 18
                                ! where 18 is the arbitrary number
                                ! assigned your printer
                                ! definition file, 'printer18.dat'.
                                ! The report would contain the
                                ! command:  ASK num
                                ! in the SETUP section, preceding
                                ! this #include statement.
```

INPUT

Syntax

```
INPUT input_var[MAXLEN=nn][prompt]
[TYPE={CHAR|TEXT|NUMBER|INTEGER|DATE}]
[STATUS=num_var][NOPROMPT][BATCH-MODE]
[FORMAT={txt_lit|_var|_col}]
```

Description

Accepts data entered by the user at a terminal.

Use MAXLEN to prevent the user from entering data that is too long. If an INSERT or UPDATE command references a variable for which the length is greater than the length defined in the database, the SQL is rejected and SQR halts. If the maximum length is exceeded, the terminal beeps (on some systems, this may cause the screen to flash instead).

If *prompt* is omitted, SQR uses the default prompt. Enter [*$|#*]*var*: . In either case, a colon (:) and two spaces are added to the prompt.

Specifying TYPE causes data type checking to occur. If the string entered is not the type specified, the terminal beeps and an error message is displayed. The INPUT command is then rerun. If TYPE=DATE is specified, then *input_var* can be a date or text variable; however, TYPE=DATE is optional if *input_var* is a date variable. If a numeric variable is used, it is validated as a numeric variable. The types CHAR, TEXT, and DATE are invalid types. The data types supported are described in the following table:

Data Type	*Description*
CHAR, TEXT	Any character. This is the default datatype.
NUMBER	A floating point number in the format [+|-]9999.999[E[+|-]99]
INTEGER	An integer in the format [+|-]99999
DATE	A date in one of the following formats: • MM/DD/YYYY [BC|AD] [HH:MI[:SS[.NNNNNN]] [AM|PM]] • MM-DD-YYYY [BC|AD] [HH:MI[:SS[.NNNNNN]] [AM|PM]] • MM.DD.YYYY [BC|AD] [HH:MI[:SS[.NNNNNN]] [AM|PM]] • SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]

Specifying STATUS causes the INPUT command to finish regardless of what the user enters. No error message is displayed. A nonzero error code is stored in the indicated numeric variable if the length or datatype entered is incorrect.

The following table lists the values of the STATUS argument of the INPUT command:

Status Value	*Indicates*
0	Successful.
1	Bad type (did not match the datatype of TYPE).
2	Too long (longer than MAXLEN, or the input for an INTEGER variable is < –2147483648 or > +2147483647).
3	No arguments remain on the command line. The command was ignored.

By using NOPROMPT and STATUS with the SHOW command, you can write a sophisticated data entry routine.

FORMAT can be used only with dates. It can be a date edit mask or the keyword DATE. Use the keyword DATE if the date must be in the format as specified with the INPUT-DATE-EDIT-MASK setting for the current locale. If FORMAT has not been set, use a database-independent format for the data as listed in the datatypes table.

Parameters

Parameter	*Description*
input_var	Specifies a text, numeric, or date variable for the input data.
MAXLEN	Specifies the maximum length for the data.
prompt	Specifies the prompt displayed to the user.
TYPE	Specifies the datatype required for the input.
STATUS	Specifies a numeric variable for a return status code.
NOPROMPT	Prevents the prompt from being displayed before the INPUT command is processed.
BATCH-MODE	If BATCH-MODE is specified and no more arguments are in the command line, a value of 3 is returned in the STATUS variable and the user is not prompted for input.
FORMAT	Specifies the format for entering a date. The Date Edit Format Codes table lists date edit format codes.

Example

The following example shows several INPUT commands:

```
input  $state  maxlen=2  'Please enter state abbreviation'
input  #age  'Enter lower age boundary'  type=integer
input  $start_date  'Enter starting date for report'  type=date
input  $date_in  format='Mon dd yyyy'
input  $date  format=date
```

The following example shows another INPUT command:

```
show clear-screen (5,32) reverse 'CUSTOMER SUMMARY' normal
Try_again:
show  (12,20)  'Enter Start Date: '  clear-line
input  $start-date  noprompt  status=#istat  type=date
if  #istat  !=  0
     show (24,1) 'Please enter date in format DD-MON-YY' beep
     goto  try_again
end-if
show  (24,1)  clear-line            ! Clear error message line.
```

The following example illustrates the use of the BATCH-MODE option:

```
begin-program
 while (1)
  input $A status=#stat batch-mode
  if #stat = 3
   break
  else
   do procedure ($a)
  end-if
 end-while
end-program
```

See Also

ALTER-LOCALE

The INPUT-DATE-EDIT-MASK setting in the chapter "Using the PSSQR.INI File and the PSSQR Command Line."

LAST-PAGE

Syntax

LAST-PAGE *position* [*pre_txt_lit*[*post_txt_lit*]]

Description

Places the last page number on each page, as in *page n of m.*

The text strings specified in *pre_txt_lit* and *post_txt_lit* are printed immediately before and after the number.

Using LAST-PAGE causes SQR and SQRT to delay printing until the last page has been processed so that the number of the last page is known.

Parameters

Parameter	*Description*
position	Specifies the position for printing the last page number. See the POSITION command for a description of the *position* parameter.
pre_txt_lit	Specifies a text string to be printed before the last page number.
post_txt_lit	Specifies a text string to be printed after the last page number.

Example

The following example illustrates the LAST-PAGE command:


```
begin-footing 1
   page-number         (1,37)  'Page '    ! Will appear as
   last-page            ()   ' of ' '.'    ! "Page 12 of 25."
end-footing
```

See Also

PAGE-NUMBER, BEGIN-HEADING, BEGIN-FOOTING

LET

Syntax

LET *dst_var=expression*

Description

Assigns the value of an expression to a string, numeric, or date variable.

Valid expressions are formed as a combination of operands, operators, and functions. String, numeric, date, and array field operands can be used in an expression and embedded functions. SQR supports a standardized set of mathematical operators and logical comparison operators working within a carefully defined set of precedence rules. SQR also provides a rich set of mathematical, string, date, and file manipulation functions along with a number of special purpose utility functions. All combined, the SQR expression provides a powerful tool that can be tailored to suit any information processing need. The following detail outlines the specific behavior of each expression component: (1) the operand, (2) the operator, and (3) the function.

Parameters

Parameter	*Description*
dst_var	A string, numeric, or date variable or array field to which the result of the expression is assigned.
expression	The expression to evaluate.

Operands

Operands form the backbone of an SQR expression. Operands do not have to be the same type. You can combine string, numeric, and array field operands to form a valid expression. SQR performs a sequence of automatic operand conversions when it evaluates expressions that contain dissimilar operand types. As the expression is evaluated, operands of lower precision are converted to match the operand of higher precision. Consider the following code example:

```
let  #answer = #float * #decimal  / #integer
```


Because the *multiply* and *divide* operators are equal in precedence, the expression is evaluated as *(#float * #decimal)* / *#integer*. Working from the inside out, the *#float* variable is converted to a decimal type in which a multiply is performed yielding the simplified expression *(#decimal)*/*#integer*. SQR now converts the *#integer* operand to a decimal type before performing the final divide. When finished with the expression evaluation, SQR converts the result to match the type of the *#answer* variable.

Converting operands of lower precision to operands of higher precision preserves the number of significant digits. The number of significant digits is not lost when an integer is converted to float or decimal. In a similar manner, the number of significant digits is preserved when floating point operands are converted to the decimal type. The number of significant digits is sacrificed only when the final result is converted to match the type of the *#answer* variable and this variable is less precise than the highest of the operands being evaluated. In the example, precision is not lost if the *#answer* is declared as a decimal type. SQR considers integer variables as the lowest in the precision hierarchy, followed by float and then decimal.

Here are a few simple expression examples:

```
let #discount  =  round (&price * #rate / 100, 2)
let $name  =  $first_name || ' ' || $last_name
let customer.total (#customer_id) =

   customer.total (#customer_id) + #invoice_total
if not range(upper($code), 'A', 'G')
   ...processing when out of range...
let store.total (#store_id, #qtr) =

   store.total (#store_id, #qtr) + #invoice_total
let $date1 = strtodate ('Apr 10 2004', 'MON DD YYYY')
```

The following sections list operators and functions supported in expressions.

Operators

Operators of the same precedence are processed in the sequence in which they appear in the expression, from left to right. Use parentheses to override the normal precedence rules. All numeric types (decimal, float, integer) are supported for all operators.

This table lists operators in descending order of precedence (operators listed in the same row within the table have the same precedence):

Operator	***Explanation***
||	Concatenate two strings or dates
+, -	Sign prefix (positive or negative)
^	Exponent
*, /, %	Multiply, divide, remainder: a % b = mod(a,b) for integers

Operator	*Explanation*
+, -	Plus, minus **Note.** SQR distinguishes between a sign prefix and arithmetic operation by the context of the expression.
>, <, >=, <=, <>, !=, =	Comparison operators: greater than, less than, greater or equal to, less than or equal to, not equal to (!= or <>), equal to.
not	Logical NOT
and	Logical AND
or, xor	Logical OR, XOR (exclusive OR)

Functions

This section lists numeric, file-related, and miscellaneous functions. The functions are listed in alphabetical order.

Function arguments are enclosed in parentheses and can be nested. Arguments referenced as x, y, or z indicate the first, second, or third argument of a function. Otherwise, functions take a single argument or no arguments. All arguments are evaluated before a function is evaluated.

Not all functions support all numeric types (decimal, float, integer). Certain functions do not support the decimal type directly, but convert input decimal operands to the float type before the function is evaluated. The following table annotates the functions that directly support the decimal type and the ones that do not.

Use parentheses to override the normal precedence rules.

This table describes numeric functions:

Function	*Description*
`abs`	Returns the absolute value of *num_value.* This function returns a value of the same type as *num_value.* Syntax: *`dst_var`* `= abs`*`(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. *dst_var* = decimal, float, or integer variable. Example: `let #dabsvar = abs(#dvar)`

Function	*Description*
`acos`	Returns the arccosine of *num_value* in the range of 0 to p radians. The value of *num_value* must be between –1 and 1. This function returns a float value. Syntax: *`dst_var = acos(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #facosvar = acos(#fvar)`
`asin`	Returns the arcsine of *num_value* in the range of –p/2 to p/2 radians. The value of *num_value* must be between –1 and 1. This function returns a float value. Syntax: *`dst_var = asin(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fasinvar = asin(#fvar)`
`atan`	Returns the arctangent of *num_value* in the range of –p/2 to p/2 radians. The value of *num_value* must be between –1 and 1. This function returns a float value. Syntax: *`dst_var = atan(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fatanvar = atan(#fvar)`

Function	*Description*
`ceil`	Returns a value representing the smallest integer that is greater than or equal to *num_value*. This function returns a value of the same type as *num_value.* Syntax: *dst_var* = `ceil`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. *dst_var* = decimal, float, or integer variable. Example: `let #fceilvar = ceil(#fvar)`
`cos`	Returns the cosine of *num_value*. This function returns a float value. Syntax: *dst_var* = `cos`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fcosvar = cos(#fvar)`
`cosh`	Returns the hyperbolic cosine of *num_value*. This function returns a float value. Syntax: *dst_var* = `cosh`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fcoshvar = cosh(#fvar)`

Function	*Description*
`deg`	Returns a value expressed in degrees of *num_value*, which is expressed in radians. This function returns a float value. Syntax: *`dst_var = deg(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fdegvar = deg(#fvar)`
`e10`	Returns the value of 10 raised to *num_value*. This function returns a float value. Syntax: *`dst_var = e10(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fe10var = e10(#fvar)`
`exp`	Returns the value of e raised to *num_value*. This function returns a float value. Syntax: *`dst_var = exp(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fexpvar = exp(#fvar)`

Function	*Description*
`floor`	Returns a value representing the largest integer that is less than or equal to *num_value*. This function returns a value of the same type as *num_value*. Syntax: *dst_var* = `floor`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. *dst_var* = decimal, float, or integer variable. Example: `let #ffloorvar = floor(#fvar)`
`log`	Returns the natural logarithm of *num_value*. This function returns a float value. Syntax: *dst_var* = `log`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #flogvar = log(#fvar)`
`log10`	Returns the base-10 logarithm of *num_value*. This function returns a float value. Syntax: *dst_var* = `log10`*(num_value)* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #flog10var = log10(#fvar)`

Function	*Description*
`mod`	Returns the fractional remainder, f, of *x_value*/ *y_value* such that *x_value* = i * *y_value* + f, where i is an integer, f has the same sign as *x_value*, and the absolute value of f is less than the absolute value of *y_value*. The arguments are promoted to the type of the greatest precision and the function returns a value of that type. Syntax: *dst_var* = mod(*x_value, y_value*) *x_value* = decimal, float, or integer literal, column, variable, or expression. *y_value* = decimal, float, or integer literal, column, variable, or expression. *dst_var* = decimal, float, or integer variable. Example: `let #fmodvar = mod(#fxvar, #fyvar)`
`power`	Returns the value of *x_value* raised to the power of *y_value*. This function returns a float value. Syntax: *dst_var* = power(*x_value, y_value*) *x_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *y_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fpowervar = power(#fxvar, #fyvar)`

Function	*Description*

`rad`	Returns a value expressed in radians of *num_value*, which is expressed in degrees. This function returns a float value.
	Syntax:
	`dst_var = rad(num_value)`
	num_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	place_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #fradvar = rad(#fvar)`

`round`	Returns a value that is *num_value* rounded to place_value digits after the decimal separator. This function returns a value of the same type as *num_value.*
	Syntax:
	`dst_var = round(num_value, place_value⇒`
	`⇒`
	`)`
	num_value = decimal, float, or integer literal, column, variable, or expression.
	place_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #frndvar = round(#fvar, #fplace) ⇒`
	`(#x, #y)`

`sign`	Returns a –1, 0, or +1 depending on the sign of *num_value.* This function returns a float value.
	Syntax:
	`dst_var = sign(num_value)`
	num_value = decimal, float, or integer literal, column, variable, or expression.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #fsignvar = sign(#fvar)`

Function	*Description*
`sin`	Returns the sine of *num_value.* This function returns a float value. Syntax: *`dst_var`* `= sin`*`(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fsinvar = sin(#fvar)`
`sinh`	Returns the hyperbolic sine of *num_value.* This function returns a float value. Syntax: *`dst_var`* `= sinh`*`(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fsinhvar = sinh(#fvar)`
`sqrt`	Returns the square root of *num_value.* This function returns a float value. Syntax: *`dst_var`* `= sqrt`*`(num_value)`* *num_value* = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float. *dst_var* = decimal, float, or integer variable. Example: `let #fsqrtvar = sqrt(#fvar)`

Function	*Description*

`tan`	Returns the tangent of *num_value*. This function returns a float value.
	Syntax:
	`dst_var = tan(num_value)`
	num_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #ftanvar = tan(#fvar)`

`tanh`	Returns the hyperbolic tangent of *num_value*. This function returns a float value.
	Syntax:
	`dst_var = tanh(num_value)`
	num_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #ftanhvar = tanh(#fvar)`

`trunc`	Returns a value that is *num_value* truncated to place_value digits after the decimal separator. This function returns a value of the same type as *num_value*.
	Syntax:
	`dst_var = trunc(num_value, place_value⇒`
	`⇒`
	`)`
	num_value = decimal, float, or integer literal, column, variable, or expression.
	place_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #ftruncvar = trunc(#fvar,⇒`
	`#fplace)`

The transcendental functions sin, cos, tan, sinh, cosh, and tanh take their arguments in radians. The functions asin, acos, and atan return radian values. To convert from radians to degrees or degrees to radians, use the rad or deg functions as shown here:

```
let #x = sin(rad(45))            ! Sine of 45 degrees.
let #y = deg(asin(#x))           ! Convert back to degrees.
```

If arguments or intermediate results passed to a numeric function are invalid for that function, SQR halts with an error message.

For example, passing a negative number to the sqrt function causes an error. Use the cond function described in the Miscellaneous Functions table to prevent division by zero or other invalid function or operator argument values.

The following table lists file-related functions. These functions return 0 (zero) when successful; otherwise, they return the system error code.

Function	***Description***

delete	Deletes the file *filename.* The function returns either a 0 (zero) to indicate success or the value returned from the operating system to indicate an error.
	Syntax:
	`stat_var = delete(filename)`
	filename = text literal, column, variable, or expression.
	stat_var = decimal, float, or integer variable.
	Example:
	`let #fstatus = delete($filename)`

exists	Determines whether the file *filename* exists. The function returns either a 0 (zero) to indicate success or the value returned from the operating system to indicate an error.
	Syntax:
	`stat_var = exists(filename)`
	filename = text literal, column, variable, or expression.
	stat_var = decimal, float, or integer variable.
	Example:
	`let #fstatus = exists($filename)`

rename	Renames *old_filename* to *new_filename.* The function returns either a 0 (zero) to indicate success or the value returned from the operating system to indicate an error.
	Syntax:
	`stat_var = rename(old_filename, new_filename)`
	old_filename = text literal, column, variable, or expression.
	new_filename = text literal, column, variable, or expression.
	stat_var = decimal, float, or integer variable.
	Example:
	`let #fstatus = rename($old_filename, $new_⇒ filename)`

The following table lists miscellaneous functions. These functions return a string value unless otherwise indicated.

In these functions where a string argument is expected and a date variable, column, or expression is entered, SQR converts the date to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Except where noted in an individual function, if a string variable, column, or expression is entered where a date argument is expected, the string must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats listed in the Default Database Formats table, or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.

Function	***Explanation***

array	Returns a pointer to the starting address of the specified array field. The value returned from this function can be used only by a user-defined function. See the routine printarray in the file UFUNC.C for complete instructions on how to use this function.
	Syntax:
	`array_var = array(array_name, field_name)`
	array_name = text literal, column, variable, or expression
	field_name = text literal, column, variable, or expression
	array_var = text variable
	Example:
	`let #fstatus = printarray(array('products',⇒ 'name'), 10, 2, 'c')`

ascii	Returns the numeric value for the first character in *str_value*. This function returns a float value.
	Syntax:
	`ascii_var  = ascii(str_value)`
	str_value = date or text literal, column, variable, or expression
	ascii_var = decimal, float, or integer variable
	Example:
	`let #fascii = ascii($filename)`

Function	*Explanation*

asciic	Returns the numeric value for the first character (rather than byte) of the specified string.
	Syntax:
	`ascii_var = asciic(str_value)`
	str_value = date or text literal, column, variable, or expression
	ascii_var = decimal, float, or integer variable
	Example:
	`let #fascii = asciic($filename)`

chr	Returns a string that is composed of a character with the numeric value of *num_value.*
	Syntax:
	`dst_var = chr(num_value)`
	num_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = text variable.
	Example:
	`let $svar = chr(#num)`

cond	Returns *y_value* if the *x_value* is nonzero; otherwise, returns *z_value.* If *y_value* is numeric, the *z_value* must also be numeric; otherwise, date and textual arguments are compatible. If either the *y_value* or *z_value* is a date variable, column, or expression, a date is returned. The return value of the function depends on which value is returned.
	Syntax:
	`dst_var = cond(x_value, y_value, z_value)`
	x_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	y_value = Any literal, column, variable, or expression
	z_value = Any literal, column, variable, or expression
	dst_var = Any variable
	Example:
	`let #avg = #total / cond(&rate != 0, &rate, 1)`

Function	*Explanation*

dateadd	Returns a date after adding (or subtracting) the specified units to the *date_value*.
	Syntax:
	`dst_var = dateadd(date_value, units_value, ⇒ quantity_value)`
	date_value = date variable or expression.
	units_value = text literal, column, variable, or expression. Valid units are 'year', 'quarter', 'week', 'month', 'day', 'hour', 'minute', and 'second'.
	quantity_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to float.
	dst_var = date variable
	Example:
	`let $date = dateadd($startdate, 'day', 7.5)`

datediff	Returns the difference between the specified dates expressed in *units_value*. The function returns a float value. The result can be negative if the first date is earlier than the second date.
	Syntax:
	`dst_var = datediff(date1_value, date2_value, ⇒ units_value)`
	date1_value = date variable or expression.
	date2_value = date variable or expression.
	units_value = text literal, column, variable, or expression. Valid units are 'year', 'quarter', 'week', 'month', 'day', 'hour', 'minute', and 'second'
	dst_var = decimal, float, or integer variable.
	Example:
	`let #diff = datediff($date1, $date2, 'hour')`

datenow	Returns the current local date and time from the client machine.
	Syntax:
	`dst_var = datenow()`
	dst_var = date variable
	Example:
	`let $date = datenow()`

Function	*Explanation*
datetostr	Converts the date *date_value* to a string in the format format_mask. Syntax: *dst_var* = datetostr(*date_value* [, *format_mask*]) *date_value* = date variable or expression. *format_mask* = text literal, column, variable, or expression. The keyword DATE can be used to specify the DATE-EDIT-MASK setting from the current locale. If this argument is not specified, the format specified by the SQR_DB_DATE_FORMAT setting is used. If this has not been set, the first database-dependent format listed in the Default Database Formats table is used. *dst_var* = text variable Example: let $formdate = datetostr($date, 'Day Mon DD, YYYY') let $localedate = datetostr($date, DATE)

edit	Formats *source_value* according to edit_mask and returns a string containing the result.
	Syntax:
	dst_var = edit(*source_value, edit_mask)*
	source_value = Any literal, column, variable, or expression
	edit_mask = text literal, column, variable, or expression
	dst_var = text variable
	Example:)
	`let $phone  =  edit(&phone, '(xxx) xxx-xxxxx')⇒`
	`let $price = edit(#price, '999.99')⇒`
	`let $today = edit($date, 'DD/MM/YYYY'`

getenv	Returns the value of the specified environment variable. If the environment variable does not exist, an empty string is returned. Syntax: *dst_var* = getenv(*env_value)* *env_value* = text literal, column, variable, or expression *dst_var* = text variable Example: `let $myuser = getenv('USER')`

Function	*Explanation*

instr	Returns the numeric position of *sub_value* in *source_value* or 0 (zero) if not found. The search begins at offset *offset_value*. This function returns a float value.
	Syntax:
	`dst_var = instr(source_value, sub_value, offset_⇒ value)`
	source_value = date or text literal, column, variable, or expression.
	sub_value = text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #offset = instr(&description, 'auto', 10)`

instrb	Performs the same functionality as the instr function except that the starting point and returned value are expressed in bytes rather than in characters.
	Syntax:
	`dst_var = instrb(source_value, sub_value, offset_⇒ value)`
	source_value = date or text literal, column, variable, or expression.
	sub_value = text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	dst_var = decimal, float, or integer variable.
	Example:
	`let #offset = instrb(&description, 'auto', 10)`

isblank	Returns a value of 1 (one) if source_val is an empty string, null string, or composed entirely of white-space characters; otherwise, returns a value of 0 (zero).
	Syntax:
	`dst_var = isblank(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #blank = isblank(&description)`

Function	*Explanation*

isnull	Returns a value of 1 (one) if source_val is null; otherwise, returns a value of 0 (zero).
	Syntax:
	`dst_var = isnull(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #null = isnull($date)`

length	Returns the number of characters in *source_value.*
	Syntax:
	`dst_var = length(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #length = length(&description)`
	Note. Oracle recommends that you use either the lengthp or lengtht function instead of the length function.

lengthb	(Multibyte versions of SQR only) Has the same functionality as the length function except that the return value is expressed in bytes rather than in characters.
	Syntax:
	`dst_var = lengthb(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #length = lengthb(&description)`
	Note. Oracle recommends that you use either the lengthp or lengtht function instead of the lengthb function.

lengthp	Returns the length of a given string in print positions.
	Syntax:
	`dst_var = lengthp(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #printlen = lengthp(&string)`

Function	*Explanation*

lengtht	Returns the length of a given string in bytes when converted (transformed) to a specified encoding.
	Syntax:
	`dst_var = lengtht(source_value, encoding_value)`
	source_value = date or text literal, column, variable, or expression
	encoding_value = text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #sjislen = lengtht($string, 'Shift-JIS')`

lower	Converts the contents of *source_value* to lowercase and returns the result.
	Syntax:
	`dst_var = lower(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $lower = lower(&description)`

lpad	Pads the *source_value* on the left to a length of *length_value* using *pad_value* and returns the result.
	Syntax:
	`dst_var = lpad(source_value, length_value, pad_⇒ value)`
	source_value = date or text literal, column, variable, or expression.
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	pad_value = text literal, column, variable, or expression.
	dst_var = text variable.
	Example:
	`let $lpad = lpad($notice, 25, '.')`

Function	*Explanation*

ltrim	Trims characters in *source_value* from the left until a character is not in *set_value* and returns the result.
	Syntax:
	`dst_var = ltrim(source_value, set_value)`
	source_value = date or text literal, column, variable, or expression
	set_value = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $ltrim = ltrim(&description, '.')`

nvl	Returns *y_value* if the *x_value* is null; otherwise, returns *x_value*. If *x_value* is numeric, *y_value* must also be numeric; otherwise, date and textual arguments are compatible. In any case, the *x_value* determines the type of expression returned. The return value of the function depends on which value is returned.
	Syntax:
	`dst_var = nvl(x_value, y_value)`
	x_value = Any literal, column, variable, or expression
	y_value = Any literal, column, variable, or expression
	dst_var = Any variable
	Example:
	`let $city = nvl(&city, '-- not city --')`
	If *x_value* is a date and *y_value* is textual, *y_value* is validated according to the following rules:
	For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYMDD[HH24[MI[SS[NNNNNN]]]]'.
	For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting, or the format listed in the DATE Column Formats table.
	For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting, or the format as listed in the TIME Column Formats table.

Function	*Explanation*

range	Returns a value of 1 (one) if *x_value* is between *y_value* and *z_value*; otherwise, returns a value of 0 (zer0). If the first argument is text or numeric, the other arguments must be of the same type. If the first argument is a date, the remaining arguments can be dates, text, or both. You can also perform a date comparison on a mix of date and text arguments, for example, where *x_value* is a date and *y_value* and *z_value* are text arguments. In a comparison of this sort, *y_value* must represent a date that is earlier than that of *z_value.*
	
	Syntax:
	
	dst_var = range*(x_value, y_value, z_value)*
	x_value = Any literal, column, variable, or expression
	
	y_value = Any literal, column, variable, or expression
	
	z_value = Any literal, column, variable, or expression
	
	dst_var = decimal, float, or integer variable
	
	Example:
	
	`let #inrange = range(&grade, 'A', 'D')⇒`
	`let #inrange = range($date, $startdate,⇒`
	`$enddate)⇒`
	`let #inrange = range($date, $startdate, '15-Apr-⇒`
	`04')⇒`
	`let #inrange = range(#price, #low, #high)`
	If *x_value* is a date and *y_value* and/or *z_value* is textual, then *y_value* and/or *z_value* is validated according to the following rules:
	
	For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYM-MDD[HH24[MI[SS[NNNNNN]]]]'.
	
	For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting, or the format listed in the table DATE Column Formats.
	
	For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting, or the format as listed in table TIME Column Formats.

Function	*Explanation*

replace	Inspects the contents of *source_value* and replaces all occurrences of from_string with to_string and returns the modified string.
	Syntax:
	`dst_var = replace(source_value, from_string, to_string)`
	source_value = date or text literal, column, variable, or expression
	from_string = text literal, column, variable, or expression
	to_string = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $replaced = replace($paragraph, 'good', 'excellent')`

roman	Returns a string that is the character representation of *source_value* expressed in lowercase roman numerals.
	Syntax:
	`dst_var = roman(source_value)`
	source_value = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $roman = roman(#page-count)`

rpad	Pads the *source_value* on the right to a length of *length_value* using *pad_value* and returns the result.
	Syntax:
	`dst_var = rpad(source_value, length_value, pad_value)`
	source_value = date or text literal, column, variable, or expression
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	pad_value = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $rpad = rpad($notice, 25, '.')`

Function	*Explanation*

rtrim	Trims characters in *source_value* from the right until a character is not in *set_value* and returns the result.
	Syntax:
	`dst_var = rtrim(source_value, set_value)`
	source_value = date, or text literal, column, variable, or expression
	set_value = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $rtrim = rtrim(&description, '.')`

strtodate	Converts the string *source_value* in the format format_mask to a date type. Syntax: `dst_var = strtodate(source_value [, format_mask])` *source_value* = text literal, column, variable, or expression. *format_mask* = text literal, column, variable, or expression that describes the exact format of the *source_value*. The keyword DATE can be used to specify the DATE-EDIT-MASK setting from the current locale. If this argument is not specified, then *source_value* must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats (see the Default Database Formats table), or the database-independent format 'SYYYYMDD[HH24[MI[SS[NNNNNN]]]]'. Valid format codes are specified in the Date Edit Format Codes table. *dst_var* = date variable Example: let $date = strtodate($str_date, 'Mon DD, YYYY') let $date = strtodate($str_date, DATE)

Function	*Explanation*

substr	Extracts the specified portion *source_value.* The extraction begins at *offset_value* (origin is 1) for a length of *length_value* characters.
	Syntax:
	`dst_var = substr(source_value, offset_value, ⇒ length_value)`
	source_value = date or text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	dst_var = text variable.
	Example:
	`let $piece = substr(&record, 10, #len)`
	Note. Oracle recommends that you use either the substrp or substrt function instead of the substr function.

substrb	Has the same functionality as the substr function except that the starting point and length are expressed in bytes rather than in characters.
	Syntax:
	`dst_var = substrb(source_value, offset_value, ⇒ length_value)`
	source_value = date or text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	dst_var = text variable.
	Example:
	`let $piece = substrb(&record, 10, #len)`
	Note. Oracle recommends that you use either the substrp or substrt function instead of the substrb function.

Function	*Explanation*

substrp	Returns a substring of a given string starting at a specified print position into the string and of a specified print length.
	Syntax:
	`dst_var = substrp(source_value, offset_value, ⇒ length_value)`
	source_value = date or text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	dst_var = decimal, float, or integer variable.
	Example:
	`let $sub = substrp(&string, #printpos, #printlen)`

substrt	Returns a substring of a given string starting at a specified byte and byte length in a given encoding.
	Syntax:
	`dst_var = substrt(source_value, offset_value, ⇒ length_value, encoding_value)`
	source_value = date or text literal, column, variable, or expression.
	offset_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	length_value = decimal, float, or integer literal, column, variable, or expression. The value is always converted to integer.
	encoding_value = text literal, column, variable, or expression.
	dst_var = text variable.
	Example:
	`let $sjisPrep = substrt(&string, 1, 10, 'Shift-⇒ JIS')`

to_char	Converts *source_value* to a string, using maximum precision.
	Syntax:
	`dst_var = to_char(source_value)`
	source_value = decimal, float, or integer literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $string = to_char(#number)`

Function	*Explanation*

to_multi_byte	Converts the specified string in the following way: Any occurrence of a double-byte character that also has a single-byte representation (numerals, punctuation, roman characters, and katakana) is converted.
	Syntax:
	`dst_var = to_multi_byte (source_value)`
	source_value = date or text literal, column, variable, or expression
	Example:
	`let $multi = to_multi_byte (&text)`

to_number	Converts *source_value* to a number. This function returns a float value.
	Syntax:
	`dst_var = to_number(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #value = to_number($number)`

to_single_byte	Converts the specified string in the following way: For SJIS, EBCDIK290, and EBCDIK1027, any occurrence of a single-byte character that also has a multibyte representation (numerals, punctuation, roman characters, and katakana) is converted. This function also converts a sequence of kana characters followed by certain grammatical marks into a single-byte character that combines the two elements. For all other encodings, the string is not modified.
	Syntax:
	`dst_var = to_single_byte(source_value)`
	source_value = date or text literal, column, variable, or expression
	Example:
	`let $single = to_single_byte (&text)`

Function	*Explanation*

translate	Inspects the contents of *source_value* and converts characters that match those in *from_set* to the corresponding character in *to_set* and returns the translated string.
	Syntax:
	`dst_var = translate(source_value, from_set, to_set)`
	source_value = date or text literal, column, variable, or expression
	from_set = text literal, column, variable, or expression
	to_set = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $translated = translate(edit(&price, '999,999.99'),',',',',')`

transform	Returns a Unicode string that is a specified transform of a given string.
	Syntax:
	`dst_var = transform(source_value, transform_value)`
	source_value = date or text literal, column, variable or expression
	transform_value = text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $hiragana = transform($string, 'ToHiragana')`

unicode	Returns a Unicode string from the string of hexadecimal values provided.
	Syntax:
	`dst_var = unicode(source_value)`
	source_value = text literal, column, variable or expression
	dst_var = text variable
	Example:
	`let $uniStr = unicode('u+5e73 u+2294')`

upper	Converts the contents of *source_value* to uppercase and returns the result.
	Syntax:
	`dst_var = upper(source_value)`
	source_value = date or text literal, column, variable, or expression
	dst_var = text variable
	Example:
	`let $upper = upper(&description)`

Function	*Explanation*

wrapdepth	Returns the number of print lines required by *source_value*. See the PRINTWRAP command for detailed descriptions of the parameters to this function. This function returns a float value.
	Syntax:
	`dst_var = wrapdepth(source_value, wrap_width, ⇒ line_height, on, strip)`
	source_value = text literal, column, variable, or expression
	wrap_width = decimal, float, or integer literal, column, variable, or expression
	line_height = decimal, float, or integer literal, column, variable, or expression
	on = text literal, column, variable, or expression
	strip = text literal, column, variable, or expression
	dst_var = decimal, float, or integer variable
	Example:
	`let #depth = wrapdepth(&description,40,1,'<13>',''⇒ )`

Writing Custom Functions

In addition to using the preceding built-in functions, you can write your own functions in C, using the supplied source file UFUNC.C.

You can pass any number of arguments to your function and values can be returned by the function or passed back in variables.

After editing and recompiling UFUNC.C, you must relink SQR.

The following step-by-step example shows how to add a user-defined function to SQR so that it can be invoked using the LET, IF, or WHILE command.

The example adds the C function random, which returns a random number. The function accepts a parameter that is used as the seed to start a new sequence of numbers. If the seed is zero, then the same sequence is used.

When adding functions to UFUNC, remember the following considerations:

- String functions require the following arguments:
  - (int) Number of arguments.
  - (char *) or (double *) Array of argument pointers, to either char[ ] or double.
  - (char *) Address for result string. If unchanged, function returns a NULL string.
  - (int) Maximum length of result string, in bytes.

- Numeric functions require the following arguments:
  - (int) Number of arguments.
  - (char *) or (double *) Array of argument pointers, to either char[ ] or double.
  - (double *) Address for result numeric value. If unchanged, function returns zero.

To add the random function to SQR, add the following modifications to the UFUNC.C file that was provided with SQR:

- Add the prototype for the random function: static void random CC_ARGS((char *, char *));
- Add the function name to the declaration list. The name of the function called from SQR is random. The return type is *n* for numeric. The number of arguments passed is 1, and the argument type is *n* for numeric. The function name in UFUNC.C is random. The characters PVR must be entered before the function name.

Name	***Return_type***	***Number of Arguments***	***Arg_Types***	***Function***
"max",	'n',	0,	"n",	PVR max,
"max",	'n',	0,	"n",	PVR max,
"split",	'n',	0,	"C",	PVR split,
"printarray",	'n',	4,	"cnnc",	PVR printarray,
"random",	'n',	1,	"n",	PVR random,

```
/* Last entry must be NULL -- do not change */
"", '\0', 0, "", 0
};
```

At the end the of UFUNC.C file, add the following random routine. The routine name must be lowercase; however, in your SQR program it can be referenced in either uppercase or lowercase.

```
static void random CC_ARGL((argc, argv, result))
CC_ARG(int, argc)               /* The number arguments passed   */
CC_ARG(double *, argv[])        /* The argument list             */
CC_LARG(double *, result)       /* Where to store result         */
{
if (*argv[0] != 0)
       srand(*argv[0]);
*result = rand();
return;
}
```

After these modifications, recompile UFUNC.C and relink SQR. See the programmer's reference manual for details about your particular machine.

This is a simple SQR program that uses the random function:


```
begin-program
  do get-random-number
  do process-calculations
end-program

begin-procedure
   let #seed = 44
   let #ran = random(#seed)
end-procedure

begin-procedure process-calculations
 .
 .
 .
```

Example

These examples show some complex expressions:

```
let #j = ((#a + #b) * #c) ^ 2
if #j > 2 and sqrt(#j) < 20 or #i + 2 > 17.4
while upper(substr(&descrip,1,#j+2)) != 'XXXX'

and not isnull(&price)
let #len = length(&fname || &initial || &lname) + 2
let $s = edit(&price * &rate, '99999.99')
let summary.total(#j) = summary.total(#j) + (&price * &rate)
if summary.total(#j) > 1000000
let store.total (#store_id, #dept)

   = store.total (#store_id, #dept) + #total
let #diff = datediff(datenow(), strtodate('1995','YYYY'),'day')
let $newdate = dateadd(datenow(),'month',50)
let $date1 = datetostr(strtodate(&sale_date), 'Day Month DD, YYYY')
```

SQR analyzes LET, IF, and WHILE expressions when it compiles your code and saves the result in an internal format so that repetitive execution is at maximum speed.

LOAD-LOOKUP

Syntax

In the SETUP section:


```
LOAD-LOOKUP
NAME=lookup_table_name
TABLE=database_table_name
KEY=key_column_name
RETURN_VALUE=return_column_name
[ROWS=initial_row_estimate_int_lit]
[EXTENT=size_to_grow_by_int_lit]
[WHERE=where_clause_txt_lit]
[SORT=sort_mode]
[QUIET]
[SCHEMA=schema_name]
[PROCEDURE=procedure_name]
[COMMAND=command_string]
[GETDATA=getdata_string]
[PARAMETERS=(...)]
[FROM-ROWSET=(m,n,-n,m-,all)]
[FROM-PARAMETER=rowset_name]
```

In the body of the report:

```
LOAD-LOOKUP
NAME=lookup_table_name
TABLE=database_table_name
KEY=key_column_name
RETURN_VALUE=return_column_name
[ROWS=initial_row_estimate_lit|_var|_col]
[EXTENT=size_to_grow_by_lit|_var|_col]
[WHERE=where_clause_txt_lit|_var|_col]
[SORT=sort_mode]
[QUIET]
[SCHEMA=schema_name]
[PROCEDURE=procedure_name]
[COMMAND=command_string]
[GETDATA=getdata_string]
[PARAMETERS=(...)]
[FROM-ROWSET=(m,n,-n,m-,all)]
[FROM-PARAMETER=rowset_name]
```

Description

Loads an internal table with columns from the database. Enables a quick search using the LOOKUP command.

Use the LOAD-LOOKUP command in conjunction with one or more LOOKUP commands.

LOAD-LOOKUP retrieves two columns from the database, the KEY field and the RETURN_VALUE field. Rows are ordered by KEY and stored in an array.

LOAD-LOOKUP commands specified in the SETUP section are always loaded and cannot reference variables for the ROWS, EXTENT, and WHERE arguments.

When you use the LOOKUP command, SQR searches the array (with a binary search) to find the RETURN_VALUE corresponding to the KEY referenced in the lookup.

Usually this type of lookup can be done with a database join, but joins take substantially longer. However, if your report is small and the number of rows to be joined is small, a lookup table can be slower because the entire table has to be loaded and sorted for each report run.

By default, SQR lets the database sort the data. This works well if the database and SQR both use the same character set and collating sequence. The SORT argument enables you to specify the sorting method if this is not true. Additionally, if the machine that SQR is running on is faster than the machine that the database is running on, letting SQR perform the sort could decrease the execution time of the report.

The only limit to the size of a lookup table is the amount of memory that your computer has available. You could conceivably load an array with many thousands of rows. The binary search is performed quickly regardless of how many rows are loaded.

Except for the amount of available memory, the number of lookup tables that can be defined is unlimited.

Parameters

Parameter	*Description*
NAME	The name of the lookup table. The array name is referenced in the LOOKUP command.
TABLE	The name of the table in the database, where the KEY and RETURN_VALUE columns or expressions are stored.
KEY	The name of the column that is used as the key in the array that is used for looking up the information. Keys can be character, date, or numeric data types. If the key is numeric, SQR permits only integers with 12 digits or fewer for the KEY column. Keys can be any database-supported expression. See the RETURN_VALUE argument.
RETURN_VALUE	The name of the column (expression) that is returned for each corresponding key. The following example is for ORACLE. See your database manual for the correct syntax. RETURN_VALUE='name||"-"||country||"-"||population'
ROWS	The initial size of the lookup table. This argument is optional, and if it is not specified, a value of 100 is used.
EXTENT	The amount to increase the array when it becomes full. This argument is optional, and if it is not specified, a value of 25% of the ROWS value is used.
WHERE	A WHERE clause used to select a subset of all the rows in the table. If it is specified, the selection begins after the word WHERE. The WHERE clause is limited to 255 characters.
SORT	The sorting method to be used. The following values are permitted: DC: Database sorts data, case-sensitive sort DI: Database sorts data, case-insensitive sort SC: SQR sorts data, case-sensitive sort SI: SQR sorts data, case-insensitive sort The default is SC or the method specified by the -LL command-line flag. The DI method is applicable only to databases that provide this feature and have been installed in that manner.

Parameter	*Description*
QUIET	Suppresses the message *Loading lookup array*... when the command executes. The warning message stating the number of duplicate keys found is also suppressed.

Example

The following command loads the array *states* with the columns *abbr* and *name* from the database table *stateabbrs*, where country is *USA*.

```
load-lookup

name=states

rows=50

table=stateabbrs

key=abbr

return_value=name

where=country='USA'
```

The preceding array is used in the example for the LOOKUP command to retrieve the full text of a state name from the abbreviation.

The following example uses the LOOKUP command to validate data entered by a user using an INPUT command:

```
get_state:
input $state 'Enter state abbreviation'
uppercase $state
lookup states $state $name
if $name = ''         ! Lookup didn't find a match
  show 'No such state.'
  goto get_state
end-if
```

Enclose any command argument with embedded spaces by single quotes, as shown in the following example:

```
where='country=''USA'' and region = ''NE'''
```

The entire WHERE clause is surrounded by quotes. The two single quotes around USA and NE are translated to one single quote in the SQL statement.

The following example uses joins in a LOAD-LOOKUP command by including two tables in TABLE and the join in WHERE:


```
load-lookup
name=states
rows=50
sort=sc
table='stateabbrs s, regions r'
key=abbr
return_value=name
where='s.abbr = r.abbr and r.location = ''ne'''
```

LOOKUP

Syntax

```
LOOKUP lookup_table_name{key_any_lit|_var|_col}
{ret_txt_var|_date_var}
```

Description

Searches a lookup table (an array) for a key value and returns the corresponding text string.

Speeds up processing for long reports. For example, if you want to print the entire state name rather than the abbreviation, you could use LOAD-LOOKUP and then LOOKUP to do this.

Parameters

Parameter	*Description*
lookup_table_name	Specifies the lookup table. This table must have been loaded previously with a LOAD- LOOKUP command.
key_any_lit | _var | _col	The key used for the lookup.
ret_txt_var | _date_var	A string variable into which the corresponding value is returned.

Example

The following example works in conjunction with the example for the LOAD-LOOKUP command:

```
lookup  states  &state_abbr  $state_name
```

This example searches the states lookup table for a matching *&state_abbr* value; if the value is found, the example returns the corresponding state name in *$state_name.* If it is not found, a null is placed in *$state_name.*

See Also

The LOAD-LOOKUP command.

LOWERCASE

Syntax

LOWERCASE *txt_var*

Description

Converts the contents of a text variable to lowercase.

Parameters

Parameter	*Description*
txt_var	Specifies a text variable to be converted to lowercase.

Example

The following example illustrates the LOWERCASE command:

```
input  $answer  'Type EXIT to stop'
lowercase  $answer     ! Allows user to enter
                       ! upper or lowercase.
if  $answer = 'exit'
   ...etc...
```

See Also

The lower function listed in the Miscellaneous Functions table.

MBTOSBS

Syntax

MBTOSBS {*txt_var*}

Description

Converts a double-byte string to its single-byte equivalent.

This command converts the specified string in the following way: any occurrence of a double-byte character that also has a single-byte representation (numerals, punctuation, roman characters, and katakana) is converted.

Parameters

Parameter	*Description*
txt_var	Specifies the string to be converted.

See Also

The TO_SINGLE_BYTE function of the LET command.

MOVE

Syntax

```
MOVE {src_any_lit|_var|_col} TO dst_any_var
[[:$]format_mask|NUMBER|MONEY|DATE]
```

Description

Moves the source field to the destination field. Optionally, you can reformat the field using the format_mask argument. Source and destination fields can be different types: numeric, text, or date. MOVE is also useful for converting from one type to another; however, date and numeric variables are incompatible.

When a date variable or column is moved to a string variable, the date is converted according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Finally, as the example shows, the edit mask can be contained in a string variable.

Parameters

Parameter	*Description*
src_any_lit | _var | _col	Specifies any source column, variable, or literal. Note that a date can be stored in a date variable or column, or a string literal, column, or variable. When you are using a date format_mask or the keyword DATE with the MOVE command, the source, if a string literal, column, or variable, must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats as listed in the Default Database Formats table, or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.
dst_any_var	Specifies a destination variable.
format_mask	Specifies an optional format mask. For additional information regarding edit masks, see the PRINT command.
NUMBER	Indicates that src_any_lit|_var|_col is to be formatted using the NUMBER-EDIT-MASK from the current locale. This option is not allowed with date variables. (See the ALTER_LOCALE command.)
MONEY	Indicates that src_any_lit|_var|_col is to be formatted using the MONEY-EDIT-MASK from the current locale. This option is not allowed with date variables. (See the ALTER_LOCALE command.)
DATE	Indicates that src_any_lit|_var|_col is to be formatted using the DATE-EDIT-MASK from the current locale. This option is not allowed with numeric variables. (See the ALTER_LOCALE command.)

Example

The following example illustrates the various features of the MOVE command:

```
 !
 ! Convert a string in place
 !
 move '123456789' to $ssn
 move $ssn to $ssn xxx-xx-xxxx
 show '$SSN = ' $ssn
```

Produces the following output:

```
$SSN = 123-45-6789
```

```
 !
 ! Convert a number to a string using an edit mask
 !
 move 1234567.89 to #value
 move #value to $value 999,999,999.99
 show '$Value = ' $value
```

Produces the following output:

```
$Value =   1,234,567.89

 !
 ! Convert a number to a string using a variable edit mask
 !
 move 123 to #counter
 move '099999' to $mask
 move #counter to $counter :$mask
 show '$Counter = ' $counter
```

Produces the following output:

```
$Counter = 000123

 !
 ! Convert a number to a string using the default edit mask
 !
 ! SQR, by default, outputs six digits of precision.
 ! If you require more or less precision, specify an edit mask.
 !
 move 123.78 to #defvar
 move #defvar to $defvar
 show '$DefVar = ' $defvar
```

Produces the following output:

```
$DefVar = 123.780000

 !
 ! Convert the number to a string using the locale default
 ! numeric edit mask
 !
 alter-locale number-edit-mask = '99,999,999.99'
 move 123456.78 to #nvar
 move #nvar to $nvar number
 show '$NVar = ' $nvar
```

Produces the following output:

```
$NVar =    123,456.78

 !
 ! Convert the money value to a string using the locale default
 ! money edit mask
 !
 alter-locale money-edit-mask = '$9,999,999.99'
 move 123456.78 to #mvar
 move #mvar to $mvar money
 show '$MVar = ' $mvar
```

Produces the following output:

```
$MVar = $  123,456.78
```

```
  !
 ! Convert the date column to a string using the locale default
 ! date edit mask
 !
 begin-select
 dcol
  from tables
 end-select
 alter-locale date-edit-mask = 'Mon-DD-YYYY'
 move &dcol to $dvar date
 show '$DVar = ' $dvar
```

Produces the following output:

```
$DVar = Jan-01-2004
```

```
  !
 ! Reset date to first day of the month
 ! ($date1 and $date2 have been defined as date variables)
 !
 let $date1 = datenow()
 move $date1 to $date2 'MMYYYY'
 show '$Date2 = ' $date2 edit 'MM/DD/YY HH:MI'
```

Produces the following output if the report was run in October of 2004:

```
$Date2 = 10/01/04 00:00
```

```
  !
 ! Convert date to a string
 ! ($date1 has been defined as a date variable)
 !
 move $date1 to $str_date 'DD-MON-YYYY'
 show '$Str_Date = ' $str_date
```

Produces the following output:

```
$Str_Date = 01-DEC-2004
```

```
  !
 ! Convert string (in partial format of SYYYYMMDDHHMISSNNN) to a
 ! date
 !
 move '20041129' to $date1
 show '$Date1 = ' $date1 edit 'Mon DD YYYY HH:MI'
```

Produces the following output:

```
$Date1 = Nov 29 2004 00:00
```

See Also

The LET command for information about copying, editing, or converting fields.

The EDIT parameter of the PRINT command for a description of the edit masks.

The ALTER-LOCALE command for a description of the arguments NUMBER-EDIT-MASK, MONEY-EDIT-MASK, and DATE-EDIT-MASK.

The PRINT command regarding the default date-time components as a result of moving an incomplete date to a date variable.

MULTIPLY

Syntax

```
MULTIPLY {src_num_lit|_var|_col} TIMES dst_num_var
[ROUND=nn]
```

Description

Multiplies the first field by the second and places the result into the second field.

When dealing with money-related values (dollars and cents), use decimal variables rather than float variables. Float variables are stored as double-precision floating-point numbers, and small inaccuracies can appear when the program multiplies many numbers in succession. These inaccuracies can appear due to the way different hardware and software implementations represent floating point numbers.

Parameters

Parameter	*Description*
src_num_lit | _var | _col	Specifies a numeric source column, variable, or literal.
dst_num_var	Specifies a destination numeric variable.
ROUND	Rounds the result to the specified number of digits to the right of the decimal point. For float variables, this value can be from 0 to 15. For decimal variables, this value can be from 0 to the precision of the variable. For integer variables, this argument is not appropriate.

Example

The following example illustrates the MULTIPLY command:

```
multiply  &quantity  times  #cost
multiply  1.5  times  #result
```

See Also

The ADD command for more information.

The LET command for a discussion of complex arithmetic expressions.

NEW-PAGE

Syntax

```
NEW-PAGE [erase_from_line_num_lit|_var|_col]
```

Description

Writes the current page and begins a new one.

For line printers, this command can optionally erase the old page starting at a specified line. After this action is performed, the location on the page is unchanged—that is, the value of #current-line is the same. The default action is to erase the entire page and reset #current-line to its initial value for the page.

In reports in which an overflow page is needed, sometimes retaining information from the first page on succeeding pages is useful.

Each NEW-PAGE occurrence adds a form feed character to the output file unless you specify FORMFEED=NO in the DECLARE-LAYOUT for this program in the SETUP section.

Note. A NEW-PAGE automatically occurs if page overflow is detected. Tabular reports do not require explicit NEW-PAGE commands; use NEXT-LISTING instead.

Parameters

Parameter	*Description*
erase_from_line_num_lit | _var | _col	Specifies a numeric column, variable, or literal for line printers.

Example

The following example illustrates the NEW-PAGE command:

```
! Write current page, then erase it
! beginning at line 5.
new-page  5
```

NEW-REPORT

Syntax

```
NEW-REPORT {report_filename_txt_lit|_var|_col}
```

Description

Closes the current report output file and opens a new one with the specified file name.

This command is normally used with single reports only. When used with multiple report declarations, this command affects the *current* report only.

The internal page counter is reset to 1 when NEW-REPORT is executed.

Note. SQR does not create a report output file until the first page is completed. NEW-REPORT might not create a new file, for example, if no data is selected and nothing is printed on the page.

Parameters

Parameter	*Description*
{report_filename_txt_lit | _var | _col}	Specifies a new file name.

Example

The following example shows the NEW-REPORT command:

```
new-report  'rep2a.lis'
new-report  $next-file
```

Assign the report file name within an SQR report by issuing the NEW-REPORT command before printing. You might even prompt for the file name to use, as shown in the following example:

```
begin-report
  input  $file  'Enter report file name'
  new-report  $file
  ...
```

After execution of this command, the reserved variable *$sqr-report* is updated to reflect the new report name.

See Also

DECLARE-REPORT, USE-REPORT

The -F command-line flag.

NEXT-COLUMN

Syntax

```
NEXT-COLUMN [AT-END={NEWLINE|NEWPAGE}]
[GOTO-TOP={num_lit|_var|_col}]
[ERASE-PAGE={num_lit|_var|_col}]
```

Description

Sets the current position on the page to the next column defined with the COLUMNS command.

Parameters

Parameter	*Description*
AT-END	Takes effect if the current column is the last one defined when NEXT-COLUMN is invoked.
GOTO-TOP	Causes the current line in the next column to be num_lit|_var|_col. This argument is useful when you are printing columns down the page.
ERASE-PAGE	Specifies where to begin erasing the page when an AT-END=NEWPAGE occurs.

Example

The following example prints columns across the page:

```
columns 10 50                     ! Define two columns
begin-select
name   (0,1,20)
phone  (0,+3,0) edit (xxx)bxxx-xxxx
  next-column  at-end=newline     ! Print names
                                  ! across the page
from phonelist                    ! within two columns.
order by name
end-select
```

The following example prints columns down the page:

```
columns 10 50
move 55 to #bottom_line
begin-select
name   (0,1,20)
phone  (0,+3,0) edit (xxx)bxxx-xxxx
  if #current-line >= #bottom_line
     next-column  goto-top=1  at-end=newpage
  else
     position (+1,1)
  end-if
from phonelist
order by name
end-select
```

See Also

COLUMNS, USE-COLUMN

NEXT-LISTING

Syntax

```
NEXT-LISTING[NO-ADVANCE]
[SKIPLINES={num_lit|_var|_col}]
[NEED={num_lit|_var|_col}]
```

Description

Ends the current set of detail lines and begins another.

NEXT-LISTING is used in tabular reports. This command causes a new vertical offset in the page to begin.

After NEXT-LISTING is executed, line 1 is reset one line below the deepest line previously printed in the page body. That is, if you then write PRINT (1, 5), the string is printed on the next available line starting in column 5. Note that the SQR-reserved variable #current-line still reflects the actual line number within the page body.

The value of SKIPLINES must be a nonnegative integer. If it is less than 0 (zero), then 0 is assumed.

The value of NEED must be an integer greater than 0. If it is less than or equal to 0, then 1 is assumed.

Parameters

Parameter	*Description*
NO-ADVANCE	Suppresses any line movement when no printing has occurred since the previous NEXT-LISTING or NEW-PAGE. The default increments the line position even when nothing was printed.

Parameter	Description
SKIPLINES	Causes the specified number of lines to be skipped before setting up the new offset.
NEED	Specifies the minimum number of lines needed to begin a new listing or set of detail lines. If this number of lines does not exist, a new page is started. You can use NEED to prevent a group of detail lines from being broken across two pages.

Example

The following example shows the NEXT-LISTING command:

```
begin-select
cust_num  (1,1)      edit 099999          ! Each detail group prints
city   (,+3)                              ! starting on line 1 since
name   (2,10,30)                          ! NEXT-LISTING keeps
address   (,+2)                           ! moving line 1 down the
  next-listing  skiplines=1  need=2       ! page. NEED=2 keeps 2
from customers order by cust_num          ! line detail groups from
end-select                                ! breaking across pages.
```

Note. The NEXT-LISTING command automatically issues a Use-Column 1 command if columns are active.

OPEN

Syntax

```
OPEN {filename_lit|_var|_col} AS
{filenum_num_lit|_var|_col}
{FOR-READING|FOR-WRITING|FOR-APPEND}
{RECORD=length_num_lit[:FIXED|:FIXED_NOLF|:VARY]}
[STATUS=num_var]
```

Description

Opens an operating system file for reading or writing. After a file is opened, it remains open until explicitly closed by the CLOSE command. A maximum of 256 files can be opened at one time.

Parameters

Parameter	Description
{filename_lit | _var | _col}	Specifies the file name. The file name can be literal, variable, or column. This makes prompting for a file name at runtime easy.

Parameter	*Description*
{filenum_num_lit | _var | _col}	Specifies a number that identifies the file in the application. All file commands use the file number to reference the file. File numbers can be numeric variables and literals. The number can be any positive integer less than 64,000.
FOR-READING	When a file is opened for reading, SQR procures all data sequentially. SQR does not allow for random access of information.
FOR-WRITING	When a file is opened for writing, a new file is created. If a file of the same name already exists, it can be overwritten (this depends on the operating system).
FOR-APPEND	When a file is opened in append mode, the current file contents are preserved. All data written is placed at the end of the file. SQR creates the file if one does not already exist. For existing files, make sure that the attributes used are the same as those used when the file was created. Failure to do this causes the unpredictable results.
RECORD	For the VARY file type, this is the maximum size for a record. For the FIXED file type, this is the size of each record without the line terminator. For the FIXED_NOLF file type, this is the size of each record.
FIXED	This file type assumes that all records contained within the file are the same length. Terminate each record by a line terminator (system-dependent). You can use this file type when writing or reading binary data.
FIXED_NOLF	This file type specifies that all records contained within the file are the same length with no line terminators. When writing records, SQR pads short records with blank characters to ensure that each record is the same length. This file type can be used when SQR is writing or reading binary data.
VARY	This file type specifies that the records can be of varying length. Each record is terminated by a line terminator (system-dependent). Only records containing display characters (no binary data) can be used safely. When SQR is reading records, any data beyond the maximum length specified is ignored. This is the default file type.
STATUS	Sets the numeric variable to zero if the OPEN command succeeds and to –1 if it fails. Without the STATUS argument, a failure on OPEN causes SQR to halt. By using a STATUS variable, you can control what processing occurs when a file cannot be opened.

Example

This section contains two examples: a regular open command and an expanded command:

```
open 'stocks.dat' as 1  for-reading record=100
open 'log.dat'    as 5  for-writing record=70
open $filename    as #j for-append  record=80:fixed
open $filename    as 2  for-reading record=80:fixed_nolf

open $filename    as 6  for-reading record=132:vary status=#filestat
if #filestat != 0
  ... error processing ...
end-if
```

An encoding directive added to the OPEN command allows differently encoded files to be managed in a single run of SQR. When no ENCODING is specified on the OPEN command, SQR uses the file input or output encoding specified in the INI file unless the file is UCS-2 encoded and auto-detection of UCS-2 files is enabled. The complete syntax of the OPEN command is:

```
OPEN {filename_lit | _var | _col} AS {filenum_num_lit |
_var | _col}
{ FOR-READING | FOR-WRITING | FOR-APPEND }
{ RECORD = length_num_lit[:FIXED | :FIXED_NOLF |
:VARY]}
[ STATUS = num_var ]
[ ENCODING = { _var | _col | ASCII | ANSI | SJIS | JEUC
| EBCDIC |
EBCDIK290 | EBCDIK1027 | UCS-2 | UTF-8 | others... }
```

The ENCODING directive is allowed only when you are converting to Unicode internally.

See Also

The READ, WRITE, and CLOSE commands for information about using files.

PAGE-NUMBER

Syntax

PAGE-NUMBER *position*[*pre_txt_lit*[*post_txt_lit*]]

Description

Places the current page number on the page.

The text specified in *pre_txt_lit* and *post_txt_lit* is printed immediately before and after the number.

Parameters

Parameter	*Description*
position	Specifies the position of the page number. See the POSITION command for a description and examples of the *position* parameter.
pre_txt_lit	Specifies a text string to be printed before the page number.
post_txt_lit	Specifies a text string to be printed after the page number.

Example

The following example shows the PAGE-NUMBER command:


```
begin-footing 1
   page-number         (1,37)  'Page '      ! Will appear as
   last-page           ()  ' of '  '.'       ! "Page 12 of 25."
end-footing
```

See Also

LAST-PAGE

POSITION

Syntax

```
POSITION position
[@document_marker[COLUMNS{num_lit|_var|_col}
[num_lit|_var|_col]...]]
```

Description

Sets the current position on a page.

Parameters

Parameter	*Description*
position	A position qualifier consisting of *(line, column, width),* where *column* and *width* are numbers that denote characters and spaces. *Line, column, and width* are all optional. If line or column is omitted, the print position is set by default to the current position, which is the position following the last printed item. Width is set by default to the width of the text that is being printed. A plus sign or minus sign indicates relative positioning in SQR. A plus sign moves the print position forward from the current position, and a minus sign moves it back.
@document_markerg	References a location defined in a document paragraph. In this case, the position used is the location of that marker in the text of the document.
COLUMNS	Defines columns beginning at the location of the document marker. The columns defined are relative to the position of the document marker. When COLUMNS is used, the entire command cannot be broken across more than one program line.

Example

The following example shows the POSITION command:

```
position  (12,5)          ! Set current position to line 12, column 5.
position  (+2,25)         ! Set position 2 lines down, at 25th column.
position  ()  @total_location              ! Set position to document
print #total  ()  edit  999,999,999        ! marker @total_location.
position  ()  @name_loc  columns 1 30
print name  ()            !  Columns are defined at @name_loc and
next-column               !  29 characters to the right of @name_loc
print title ()
```

See Also

The COLUMNS command for more information.

PeopleTools 8.52: SQR for PeopleSoft Developers, "Creating Form Letters"

PRINT

Syntax

PRINT {*any_lit*|*_var*|*_col*} *position*[*format_command*[*format_cmd_params*]...]...

Description

Puts data on the page at a specified position.

See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Changing Fonts."

Parameters

Parameter	***Description***
{any_lit | _var | _col}	Specifies the data to be printed. Dates can be contained in a date column or variable, or in a string literal, column, or variable. When you are using EDIT or DATE with the PRINT command, a date in a string literal, column, or variable must be in an acceptable format. See the description of *EDIT* for further details.
position	A position qualifier consisting of *(line, column, width)*, where *column* and *width* are numbers that denote characters and spaces. *Line, column, and width* are all optional. If line or column is omitted, the print position is set by default to the current position, which is the position following the last printed item. Width is set by default to the width of the text that is being printed. Position can be relative. See the POSITION command for a full description and examples of relative positioning.
format_command [format_cmd_params]	Specifies optional formatting commands and parameters.

Format Commands

The PRINT command has the following format commands::

- BACKGROUND
- BOLD
- BOX
- CENTER
- CODE-PRINTER
- DATE
- EDIT
- FILL
- FOREGROUND
- MONEY
- NOP
- NUMBER
- ON-BREAK
- SHADE
- UNDERLINE
- WRAP

Note. The SHADE and BOX format options are not supported for output to HTML format.

Some of these format commands can be used in combination with others and some are mutually exclusive. The following tables show which can be used together. An X indicates that they can be used together.

	BACKGROUND/ FOREGROUND	***BOLD***	***BOX***	***CENTER***	***CODE-PRINTER***	***DATE/ EDIT/ MONEY/ NUMBER***
BACKGROUND/ FOREGROUND		X		X		X
BOLD	X		X	X		X
BOX	X	X		X		X

	BACKGROUND/ FOREGROUND	***BOLD***	***BOX***	***CENTER***	***CODE-PRINTER***	***DATE/ EDIT/ MONEY/ NUMBER***
CENTER	X	X	X			X
CODE-PRINTER						
DATE/ EDIT/ MONEY/ NUMBER	X	X	X	X		
FILL	X	X	X	X		
MATCH	X	X	X	X		
NOP	X	X	X	X	X	X
ON-BREAK	X	X	X	X		X
SHADE	X	X	X	X		X
UNDERLINE	X	X		X		X
WRAP	X	X	X			

	FILL	***MATCH***	***NOP***	***ON-BREAK***	***SHADE***	***UNDERLINE***	***WRAP***
BACKGROUND/ FOREGROUND	X	X	X	X	X		X
BOLD	X	X	X	X	X	X	X
BOX	X	X	X	X	X		X
CENTER	X	X	X	X	X	X	

	FILL	*MATCH*	*NOP*	*ON-BREAK*	*SHADE*	*UNDERLINE*	*WRAP*
CODE-PRINTER			X				
DATE/ EDIT/ MONEY/ NUMBER			X	X	X	X	
FILL			X	X	X	X	X
MATCH			X	X	X	X	
NOP	X	X		X	X	X	X
ON- BREAK	X	X	X		X	X	X
SHADE	X	X	X	X		X	X
UNDERLINE	X	X	X	X	X		X
WRAP	X		X	X	X	X	

The following sections describe the PRINT format commands.

BOLD

BOLD causes the string or number to print in **bold** type.

For HP LaserJet printers, the appropriate boldface font must be loaded in the printer.

For PostScript printers, the appropriate boldface must be defined in the PostScript startup file, postscri.str.

See the DECLARE-PRINTER Command Arguments table for information about which fonts can be in boldface font.

For line printers, when the BEFORE-BOLD and AFTER-BOLD arguments on the DECLARE-PRINTER command are used, the specified strings are added before and after the data that is to be in boldface. If BEFORE-BOLD and AFTER-BOLD are not specified, BOLD has no effect.

For example:

```
print &name (+1, 20) bold
print 'Your account is in arrears' (1,1)  bold
```

BOX

BOX draws a one-line-deep graphical box around the printed data. This option has no effect for line printers.

For example:

```
print &grand_total (+5, 20) box
print 'Happy Birthday !!' (1,1)  box
```

Note. For HP LaserJet printers using proportional fonts, BOX and SHADE cannot determine the correct length of the box because it varies with the width of the characters printed. BOX and SHADE work well with fixed-pitch fonts and with all PostScript fonts.

Note. The BOX format option is not supported for output to HTML format.

CENTER

CENTER centers the field on a line. The position qualifier for column is ignored. For example:

```
print  'Quarterly Sales'  (1)  center
```

CODE-PRINTER

CODE-PRINTER has the following syntax:

```
CODE-PRINTER = printer_type
```

Valid values for *printer_type* are HT, HP, PS, LP, HTML, HPLASERJET, POSTSCRIPT, and LINEPRINTER.

CODE-PRINTER

Adds nondisplay characters to the program for the purpose of sending a sequence to the printer. CODE-PRINTER causes the string to be placed *behind* the page buffer, rather than within it, so alignment of printed data is not affected by the white space consumed by the nondisplay characters. Only strings can be printed by means of CODE-PRINTER.

Because the report might be printed on different types of printers, you should specify for which type this data is to be used. The report is ignored if it is printed to a different type. If necessary, you can send a different sequence to another type with a second PRINT statement.

For example:

```
encode '<27>[5U' into $big_font
encode '<27>[6U' into $normal_font
...
print $big_font (0, +2) code-printer=lp
print &phone () edit '(xxx) xxx-xxxx'
print $normal_font () code-printer=lp
```

In the previous example, the two CODE-PRINTER arguments put the *$big_font* and *$normal_font* sequences into the output, without overwriting any data in the page buffer. Sequences printed with the CODE-PRINTER argument are positioned by the regular line and column positioning. However, unlike the PRINT command, the current print location after execution is the beginning location where the CODE-PRINTER string was placed. Multiple coded strings printed using the same line and column location appear in the output in the same sequence in which they were printed.

DATE

You cannot use DATE with numeric columns or variables. DATE indicates that the field is to be formatted with the DATE-EDIT-MASK from the current locale. (See the ALTER_LOCALE command.) If this entry is not set, then the date is printed according to the following rules shown in the Date table.

Column Type	*Default Mask*	*If not set*
DATETIME	SQR_DB_DATE_FORMAT	See the Default Database Formats table for the format that is used.
DATE	SQR_DB_DATE_ONLY_FORMAT	See the DATE Column Formats table for the format that is used.
TIME	SQR_DB_TIME_ONLY_FORMAT	See the TIME Column Formats table for the format that is used.

EDIT

EDIT has the following syntax:

```
EDIT  edit_format
```

EDIT causes each field to be edited before it is printed. The three types of edits are::

- Text edit
- Numeric edit
- Date edit

The following table lists the text edit format characters:

Character	*Description*
X	Use character in field.
B	Insert blank.
~ (tilde)	Skip character in field.
R[n]	Reverse sequence of string for languages such as Hebrew. The optional number indicates right justification within length indicated.

Any other character, such as punctuation, in a text edit mask is treated as a constant and is included in the edited field.

The characters 8, 9, 0, V, and $ are illegal in a text edit mask because they are used to indicate that the mask is for a numeric edit.

The following table lists the numeric edit format characters:

Character	***Description***
8	Digit, zero fill to the right of the decimal point, trim leading blanks (left-justify the number).
9	Digit, zero fill to the right of the decimal point, space fill to the left.
0	Digit, zero fill to the left.
$	Dollar sign, optionally floats to the right.
B	Treated as a 9, but if a value is zero, the field is converted to blanks.
C	Entered at the end of the mask, causes the comma and period characters to be transposed when the edit occurs. This is to support monetary values for which periods delimit thousands and commas delimit decimals. (Example: 1.234,56).
E	Scientific format. The number of 9s after the decimal point determines the number of significant digits displayed. The E can be uppercase or lowercase; the display follows the case of the mask.
V	Implied decimal point.
MI	Entered at the end of the mask, causes a minus to appear at the right of the number.
PR	Entered at the end of the mask, causes angle brackets (< >) to be displayed around the number if the number is negative.
PS	Entered at the end of the mask, causes parentheses to be displayed around the number if the number is negative.
PF	Entered at the end of the mask, causes floating parentheses to be displayed around the number if the number is negative.

Character	*Description*
NA	Entered at the end of the mask, causes NA to be displayed if the numeric column variable is null. The case of NA follows that of the mask.
NU	Entered at the end of the mask, causes blanks to be displayed if the numeric column variable is null.
.	Decimal point.
,	Comma.

Characters other than those listed in the Numeric Edit Format Characters table are not allowed for numeric edit masks and cause errors during processing.

The following table shows sample edit masks and resulting fields:

Mask	*Value*	*Display*
999.99	34.568	34.57
9,999,999V9999	123,456.7890	123,4567890
8,888,888.888	123,456.789	123,456.789
9,999	1234	1,234
9,999	123	123
09999	1234	01234
9999	–123	–123
9999	–1234	****
9999	12345	****
9999mi	–123	123-

Mask	*Value*	*Display*
9999pr	–123	< 123>
999999ps	–123	(123)
999999pf	–123	(123)
9999na	(null)	NA
9999nu	(null)	(blank)
$$9,999.99c	1234.56	$1.234,56
$$9,999.99	1234.56	$1,234.56
$$9,999.99	12.34	$ 12.34
$$$,$$9.99	12.34	$12.34
9.999e	123456	1.235e+05
B9,999	0	(blank)
B9,999	12345	12,345
(xxx)bxxx-xxxx	2169910551	(216) 991-0551
xxx-xx-xxxx	123456789	123-45-6789
~~xx~xx	ABCDEFGHIJ	CDFG
r10	ABCDEFG	GFEDCBA

The following example shows some uses of edit masks:

```
print #total (7,55,0)  edit $999,999.99              ! $ 12,345.67
print #total (7,55,0)  edit $$$9,999.99              ! $12,345.67
print #total (7,55,0)  edit 999,999.99pr             ! < 12,345.67>(if neg)
print #comm  (7,55,0)  edit b99,999.99               ! Blank if zero
print &cnum  (16,1,0)  edit 099999                   ! 001234
print #cat   (5,10,0)  edit 9.999E                   ! 1.235E+04
print #phone (16,60,0) edit (xxx)bxxx-xxxx           ! (216) 397-0551
print #total (7,55,0)  edit £££9,999.99              ! Dollar-Symbol £
```

The following table lists the date edit format codes:

Character	***Description***
YYY YY Y	Last 3, 2, or 1 digit of the year. On input, for calculating the 4-digit year, the current century, decade, or both are used. For example, a 9 using the Y mask would result in the year 1999 if the current year is in the 1990s.
YYYY SYYYY	4-digit year, S prefixes BC dates with "-".
RR	Last 2 digits of year; for years in other centuries. See the Date Edit Format Code-RR table.
CC or SCC	Century; S prefixes BC dates with "–".
BC AD	BC/AD indicator.
Q	Quarter of year (1,2,3,4; JAN–MAR=1).
RM	Roman numeral month (I–XII; JAN=I).
WW	Week of year (1–53) when week 1 starts on the first day of the year and continues to the seventh day of the year.
W	Week of the month (1–5) when week 1 starts on the first day of the month and ends on the seventh.
DDD	Day of year (1–366).
DD	Day of month (1–31).
D	Day of week (1–7). Sunday is the first day of the week.
DAY	Name of day.

Character	*Description*
DY	Abbreviated name of day.
ER	Japanese Imperial Era. Returns the name of the Japanese Imperial Era in the appropriate kanji (Heisei is the current era).
EY	Year of the Japanese Imperial Era. Returns the current year within the Japanese Imperial Era. **Note.** The common Japanese date format is 'YYYY<nen>MM<gatsu>DD<nichi>' where <nen>, <gatsu>, and <nichi> are the kanji strings for year, month, and day respectively.
J	Julian day; the number of days since Jan 1, 4713 BC. Numbers specified with *J* must be integers.
AM PM	Meridian indicator.
HH	Assumes 24-hour clock unless meridian indicator is specified.
HH12	Hour of day (1–12).
HH24	Hour of day (0–23).
SSSSS	Seconds past midnight (0–86399).
N NN NNN NNNN NNNNN NNNNNN	Fractions of a second. Precise to microseconds; however, for most hardware and databases, this much accuracy is not attainable.
MONTH	Name of month.
MON	Abbreviated name of month.
MM	Month (01–12; JAN=01).
MI	Minute (0–59).

Character	*Description*
SS	Second (0–59).
|	Used to concatenate different masks.

Note. If the last two digits of the year are between 00 and 49, the return date is in the current century. If the last two digits of the year are between 50 and 99, the return date is in the century after the current one.

Last 2 Digits of Current Year	*2-Digit Year is 00–49*	*2-Digit Year is 50–99*
00–49	The return date is in the current century.	The return date is in the century before the current one.
50–99	The return date is in the century after the current one.	The return date is in the current century.

All masks can be used by the strtodate function except for CC, SCC, Q, W, and WW.

A backslash forces the next character into the output from the mask. For example, a mask of *The cu\rre\nt \mo\nth is Month* results in the output string of *The current month is January*. Without the backslashes, the output string would be *The cu04e7t january is January*.

A vertical bar can be used as a delimiter between format codes; however, in most cases the bar is not necessary. For example, the mask 'YYYY|MM|DD' is the same as 'YYYYMMDD'.

Any other character, such as punctuation, in a date edit mask is treated as a constant and is included in the edited field. If the edit mask contains spaces, it must be enclosed in single quotes (').

The masks MON, MONTH, DAY, DY, AM, PM, BC, AD, and RM are case-sensitive and follow the case of the mask entered. For example, if the month is January, the mask Mon yields *Jan* and MON yields *JAN*.

All other masks are case-insensitive and can be entered in either uppercase or lowercase. In addition, National Language Support is provided for the following masks: MON, MONTH, DAY, DY, AM, PM, BC, and AD.

See the ALTER-LOCALE command in the SQR Samples section for additional information.

If the value of the date field being edited is *Mar 14 2004 9:35*, the edit masks produce the results in the following table.

Edit Mask	*Result*
dd/mm/yy	14/03/04
DD-MON-YYYY	14-MAR-2004
'Month dd, YYYY'	March 14, 2004

Edit Mask	*Result*
MONTH-YYYY	MARCH-2004
HH:MI	09:35
'HH:MI PM'	09:35 AM
YYYYMMDD	20040314
MM.DD.YYYY	03.14.2004
Mon	Mar
DD|D|DDD	143073

In addition to being used with the EDIT argument, edit masks can be used with the MOVE, CONCAT, DISPLAY, and SHOW commands, and with the edit function of the LET command. You edit the field using the supplied mask before storing or displaying it.

When a date with missing date components, time components, or both is displayed or printed, the defaults are as shown in this list:

- The default year is the current year.
- The default month is the current month.
- The default day is one.
- The default time is zero (00:00:00.000000).

For example, assuming today is September 7, 2004, the following assignment would produce an equivalent date-time of September 1, 2004 13:21:00.000000:

```
let $date1 = strtodate('13:21','HH:MI')
```

Edit masks can be changed dynamically by storing them in a string variable and referencing the variable name preceded by a colon (:). For example:

```
move  '$999,999.99'  to  $mask
print  #total  (5,10)  edit  :$mask
show  #total  edit  :$mask
```

When a date that is stored in a string literal, column, or variable is printed with an edit mask, it must be in one of the following formats:

- The format specified by the environment variable SQR_DB_DATE_FORMAT, or the corresponding setting in the sqr.ini file.
- One of the database-dependent formats, as listed in the Default Database Formats table.

- The database-independent format, 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]'.

When a date column or variable is printed without an edit mask, the date is printed in the format specified by the environment variable SQR_DB_DATE_FORMAT or the corresponding setting in the sqr.ini file. If this has not been set, then the date is printed in the primary database format (the first entry) listed in the Default Database Formats table.

This applies to DISPLAY, MOVE, and SHOW commands as well as PRINT.

The following table lists default date formats for each database.

Database	***Default Database Formats***
Oracle	DD-MON-YY
Informix	YYYY-MM-DD HH:MI:SS.NNN MM/DD/YYYY MM-DD-YYYY MM.DD.YYYY
ODBC	MM-DD-YYYY
DB2	YYYY-MM-DD-HH:MI:SS.NNNNNN YYYY-MM-DD
Sybase	MON DD YYYY HH:MIPM MON DD YYYY [HH:MI[:SS[:NNN]][PM] MON DD YYYY [HH:MI[:SS[.NNN]][PM] YYYYMMDD [HH:MI[:SS[:NNN]]PM] YYYYMMDD [HH:MI[:SS[.NNN]]PM]

Database	***DATE Column Formats***
DB2	YYYY-MM-DD
Informix	MM/DD/YYYY
ODBC	DD-MON-YYYY

Database	***TIME Column Formats***
DB2	HH24.MI.SS
ODBC	HH24:MI:SS

FILL

FILL fills the page with the specified character or string as indicated by the print position and length.

The following example prints a line of stars and then a line of dashes followed by stars:

```
print  '*'  (1,1,79)  fill        ! Fill line with *'s
print  '-*'  (+1,20,40)  fill     ! Fill with '-*' characters.
```

FOREGROUND/BACKGROUND

When you specify a color on the PRINT command, its scope is that of the PRINT command. If you do not define the specified color name, then the setting for *default* is used. Use the color name *none* to deactivate color for the specified area.

```
PRINT {any_lit|_var|_col}
[FOREGROUND =({color_name_lit|_var|_col}|{rgb})]
[BACKGROUND =({color_name_lit|_var|_col}|{rgb})]
```

MATCH

MATCH has the following syntax:

```
MATCH  match_text  { line_num_lit|_var|_col }
{ column_num_lit|_var|_col }  print_text ...
```

MATCH compares a field to a list of key values and if a match is found, prints the corresponding string at the specified line and column. If the match_text contains white space, it must be enclosed in single quotes ('). Any number of match texts can be tested, but each must have its own line, column, and print_text. If a match is not found, the unmatched field is printed at the position specified in the parentheses. Line and column positions for each matched string are treated as fixed or relative positions depending on the type of positioning used in the position qualifier for the PRINT command. For example:

```
 print  &type_buyer  (20,12)  match
 A 20 12 Casual
 B 20 22 Impulsive
 C 21 12 Informed
 D 21 22 Choosey
```

To use relative line and fixed-column positioning, for example, you could use the following code:

```
print  $state  (0,25)  match
 OH 0 25 Ohio
 MI 0 37 Michigan
 NY 0 25 'New York'
```

The column positions are treated as fixed locations due to the fixed 25 positions declared in parentheses.

MONEY

MONEY indicates that the column or variable is to be formatted using the MONEY-EDIT-MASK from the current locale (see the ALTER_LOCALE command). This can be used only with a numeric column or variable.

NOP

NOP suppresses the print command, causing *no operation* to be executed. This argument is useful for temporarily preventing a field from printing. For example:

```
print  &ssn  (1,1)  nop
 Hide the social security number.
```

NUMBER

NUMBER indicates that the column or variable is to be formatted with the NUMBER-EDIT-MASK from the current locale (see the ALTER-LOCALE command). This argument can be used only with a numeric column or variable.

ON-BREAK

ON-BREAK has the following syntax:

```
 ON-BREAK [PRINT={ALWAYS|CHANGE|CHANGE/TOP-PAGE|NEVER}]
[SKIPLINES={num_lit|_var|_col}]
[PROCEDURE=procedure_name[(arg1[ ,argi]...)]]
[AFTER=procedure_name[(arg1[ ,argi]...)]]
[BEFORE=procedure_name[(arg1[ ,argi]...)]]
[SAVE=txt_var]
[LEVEL=nn]
[SET=nn]
```

ON-BREAK causes the specified action in a tabular report when the value of a field changes (a break occurs). The default action prints the field only when its value changes (PRINT=CHANGE). ON-BREAK has the following qualifiers:

- PRINT specifies when the break field is printed.
  - ALWAYS duplicates the break field for each detail group.
  - CHANGE prints the value only when it changes.

    This is the default.
  - CHANGE/TOP-PAGE prints the value both when it changes and at the top of each new page.
  - NEVER suppresses printing.
- SKIPLINES specifies how many lines to skip when the value changes.
- PROCEDURE specifies the procedure to be invoked when the value changes.

  This qualifier cannot be used with either the AFTER or BEFORE qualifier.
- AFTER and BEFORE procedures specify procedures to invoke either after or before the value changes.

  If no rows are fetched, neither procedure is run. AFTER and BEFORE can be used only within a SELECT paragraph.

- The sequence of events is shown here:
  - SAVE indicates a string variable in which the previous value of a break field is stored.
  - LEVEL specifies the level of the break for reports containing multiple breaks.

    For example, a report sorted by state, county, and city might have three break levels: state is level 1 (the most major), and city is level 3 (the most minor). When a break occurs, other breaks with equal or higher level numbers are cleared. The level number also affects the sequence in which AFTER and BEFORE procedures are processed.
  - SET assigns a number to the set of leveled breaks in reports with more than one set of independent breaks.

Following is the sequence of events for a query containing ON-BREAK fields:

1. Any BEFORE procedures are processed in ascending LEVEL sequence before the first row of the query is retrieved.
2. When a break occurs in the query, the following events occur:
   - AFTER procedures are processed in descending sequence from the highest level to the level of the current break field.
   - SAVE variables are set with the new value.
   - BEFORE procedures are processed in ascending sequence from the current level to the highest level break.
   - Any breaks with the same or higher level numbers are cleared so that they do not break on the next value.
   - If a PROCEDURE has been declared, the procedure is invoked.
   - If SKIPLINES was specified, the current line position is advanced.
   - The value is printed (unless PRINT=NEVER was specified).

3. After the query finishes (at END-SELECT) any AFTER procedures are processed in descending level sequence, for example:

```
begin-select
state  (+1,1,2)  on-break level=1  after=state-tot  skiplines=2
county (,+2,14)  on-break level=2  after=cnty-tot  skiplines=1
city   (,+2,14)  on-break level=3  after=city-tot
...
end-select
```

   The breaks are processed in the following way:

   - When city breaks, the city-tot procedure is executed.
   - When county breaks, first the city-tot procedure is executed, then the cnty-tot procedure is executed.
   - When state breaks, the city-tot, cnty-tot, and state-tot procedures are processed in that sequence.

     If any BEFORE breaks were indicated, they would also be processed automatically, after all of the AFTER breaks and in sequence from lower to higher level numbers, for example:

```
begin-select
state  (+1,1,2) on-break level=1 before=bef-state after=state-tot
county (,+2,14) on-break level=2 before=bef-cnty after=cnty-tot
city   (,+2,14) on-break level=3 before=bef-city after=city-tot
...
end-select
```

Now when state breaks, this sequence of procedures is executed:

- City-tot
- Cnty-tot
- State-tot
- Bef-state
- Bef-cnty
- Bef-city

When program flow enters the query at BEGIN-SELECT, the three BEFORE procedures are executed in sequence:

- Bef-state
- Bef-cnty
- Bef-city

After the last row is retrieved, at END-SELECT, the three AFTER procedures are executed in sequence:

- City-tot
- Cnty-tot
- State-tot

The SAVE qualifier saves the previous break value in the specified string variable for use in an AFTER procedure. You may want to print the previous break field with a summary line:

```
print  &state  (+1,1)  on-break  after=state-tot  save=$old-state
```

The SET qualifier enables you to have subreports with leveled breaks. Because the ON-BREAKs are separated into sets, the associated leveled breaks in each set do not interfere with each other.

```
begin-select
state  (+1,1,2)  on-break  set=1  after=state-tot  level=1
```

SET=1 associates this leveled break with other breaks having the same set number.

SHADE

Draws a one-line deep, shaded graphical box around the printed data. For line printers, this argument has no effect.

```
print 'Company Confidential' (1,1)  shade
print &state (+2, 40) shade
```

Note. For HP LaserJet printers using proportional fonts, BOX and SHADE cannot determine the correct length of the box because it varies with the width of the characters printed. BOX and SHADE work well with fixed-pitch fonts and with all PostScript fonts.

Note. The SHADE format option is not supported for output to HTML format.

UNDERLINE

UNDERLINE prints the specified data with underlined characters. For line printers, UNDERLINE generates backspace and underscore characters, which emulates underlining. For example:

```
print &name (+1, 45) underline
print 'Your account is in arrears' (1,1)  underline
```

WRAP

WRAP wraps text at word spaces. Additional text is moved to a new line. WRAP has the following syntax:

```
WRAP {line_length_lit|_var|_col}
{max_lines_lit|_var|_col}[KEEP-TOP]
[STRIP=strip_chars][ON=break_chars][R]
[LINE-HEIGHT={line_height_lit|_var|_col}]

line_length_lit|_var|_col
```

WRAP specifies the maximum paragraph width in characters.

Note. After a string wraps, the current position is one character to the right of the last character in the column. When a string ends on the last position of a line, an implicit line feed causes the new current position to be the first character of the following line. In the SETUP section, use the DECLARE-LAYOUT command to make the page width one character wider than the right edge of the wrapped text to avoid generating an implicit line feed.

In this example, the line position is 1 for each of the three wrapped fields: note1, note2, and note3.


```
print &comment (48,20,0) wrap 50 3 ! Paragraph is 50
                                   ! characters wide,
                                   ! with a maximum
                                   ! depth of 3 lines.

print  &note1    (1,20,30)  wrap  30  4
print  &note2    (1,+2,30)  wrap  30  4
print  &note3    (1,+2,30)  wrap  30  4
```

The current print position after a wrap occurs at the bottom right edge of the wrapped paragraph. To continue printing on the same line, you must use a fixed line number for the next field.

```
max_lines_lit|_var|_col
```

Specifies the maximum paragraph depth in lines. Usually, the line length and maximum lines are indicated with numeric literals. However, WRAP can also reference numeric variables or columns. This is useful when you want to change the width or depth of a wrapped paragraph during report processing. The numeric variable can optionally be preceded by a colon (:), for example:

```
print  $comments  (1,30) wrap  #wrap_width  6
print  $message   (5,45) wrap  #msg_wid  #msg_lines
```

KEEP-TOP retains the current line position except if a page break occurs, in which case, line 1 is used as the current line position. The default action is to set the next print position at the bottom of the wrapped data.

In the following example, the column &resolution prints on the same line as the first line of the column &instructions:

```
print &phone (+1,10) edit '(xxx) xxx-xxxx'
print &instructions (+1,10,30) wrap 6 30 keep-top
print &resolution (0,+3,25)
```

The STRIP and ON arguments affect which characters are to be converted before wrapping, and which characters force a wrap to occur.

- Characters in the STRIP string argument are converted to spaces before the wrap occurs.
- Characters in the ON string argument cause a wrap at each ON character found. The ON character is not printed.

Both arguments accept regular characters and nondisplay characters for which decimal values are surrounded by angled brackets, <nn>, where *nn* is a decimal number between 1 and 255, representing the character in the current character set of the operating system. For example, to print a long data type that contains embedded carriage returns, the setup would be:

```
print  &long_field  (5,20)  wrap 42  30  on=<13>
```

The paragraph wraps at each carriage return, rather than at the usual word boundaries. If the ON character is not found within the width specified for the paragraph, the wrap occurs at a word space. The following example converts the STRIP characters to spaces before wrapping on either a line feed <10> or a space (the default):

```
print &description (20,10) wrap 50 22  strip=/\^@<13>  on=<10>
```

WRAP can also be used to print reversed characters, for support of languages such as Hebrew. An R after the length and max_lines arguments causes the field to be reversed before the wrap takes place. In addition, the entire paragraph is right-justified within the length indicated.


```
! Reverse wrap, in 30 character field.
print &comment (2,35) wrap 30 5 r
print $notes (1,50) wrap 50 7 r
```

LINE-HEIGHT specifies the number of lines to skip between each line of the wrapped data. By default, a value of 1 (single space) is assumed. The following example prints the comment column with one blank line between each printed line for a maximum of four printed lines:

```
print &comment (1,1) wrap 40 4 line-height = 2        ! Double space text
```

See Also

The LET command for information about copying, editing, or converting fields.

The ALTER-LOCALE command for a description of the arguments NUMBER-EDIT-MASK, MONEY-EDIT-MASK, and DATE-EDIT-MASK.

DISPLAY, SHOW

PRINT-BAR-CODE

Syntax

```
PRINT-BAR-CODE position
{TYPE={bar_code_type_num_lit|_var|_col}}
{HEIGHT={bar_code_height_num_lit|_var|_col}}
{TEXT={bar_code_txt_lit|_var|_col}}
[CAPTION={bar_code_caption_txt_lit|_var|_col}]
[CHECKSUM={checksum_lit}]
```

Description

Prints industry standard bar codes. SQR supports the bar code types listed in the following table.

Type	*Description*	*Text Length*	*Text Type*	*CHECKSUM RECOGNIZED*
1	UPC-A	11, 13, or 16	9	
2	UPC-E	11, 13, or 16	9	
3	EAN/JAN-13	12, 14, or 17	9	
4	EAN/JAN-8	7, 9, or 12	9	
5	3 of 9 (Code 39)	1 to 30	9, X, p	y

Type	*Description*	*Text Length*	*Text Type*	*CHECKSUM RECOGNIZED*
6	Extended 3 of 9	1 to 30	9, X, x, p, c	y
7	Interleaved 2 of 5	2 to 30	9	y
8	Code 128	1 to 30	9, X, x, p, c	
9	Codabar	1 to 30	9	y
10	Zip+4 Postnet	5, 9, or 11	9	
11	MSI Plessey	1 to 30	9	y
12	Code 93	1 to 30	9, X, p	y
13	Extended 93	1 to 30	9, X, x, p	y
14	UCC-128	19	9	
15	HIBC	1 to 30	9	y

Parameters

Parameter	*Description*
position	Specifies the position of the upper left corner. Position parameters can be relative. See the POSITION command for examples of relative positioning. Document markers are not allowed. After the statement executes, the current position is returned to this location; however, the next listing line is the next line below the bottom of the bar code. (This is different from the way the PRINT command works.)
TYPE	Specifies the type of bar code to be printed. Types are shown in the Bar Code Types table.
HEIGHT	Specifies the height of the bar code in inches. The height must be between 0.1 and 2 inches. The code prints to the nearest one-tenth of an inch. For Zip+4 Postnet, the height of the bar code is fixed. The height should be between 0.2 and 2.0 for Zip+4 Postnet. If it is less than 0.2, the bar code extends above the position specified.

Parameter	*Description*
TEXT	Specifies the text to be encoded and printed. The number and type of text characters permitted or required depends on the bar code type. See the Bar Code Types table for specifications.
CAPTION	Specifies optional text to be printed under the bar code in the current font. SQR attempts to center the caption under the bar code; however, for proportional fonts this may vary slightly. CAPTION is not valid for Zip+4 Postnet. If specified, it is ignored.
CHECKSUM	Specifies an optional check sum to be computed and printed within the bar code. Valid values are YES and NO, where NO is the default. **Note.** Some barcode types ignore the CHECKSUM qualifier. See the Bar Code Types table for those barcode types for which CHECKSUM is relevant.

Example

This example shows how to use the PRINT-BAR-CODE command to create a UPC-A bar code:

```
begin-program
   print 'Sample UPC-A Barcode' (1,1)
   print-bar-code (3,1)
      type=1                ! UPC-A
      height=0.3
      text='01234567890'
      caption='0 12345 67890'
end-program
```

This example shows how to use the PRINT-BAR-CODE command to create a ZIP+4 Postnet code:

```
begin-program
   print 'Sample Zip+4 Postnet'       (1,1)
   print 'John Q. Public'             (3,1)
   print '1234 Main Street'           (4,1)
   print 'AnyTown, USA 12345-6789'    (5,1)
   print-bar-code                     (7,1)
      type=10
      height=0.2
      text='12345678934'
end-program
```

Note. SQR does not check bar code syntax. See your bar code documentation for the proper formatting of certain bar codes.

PRINT-CHART

Syntax

```
PRINT-CHART[chart_name]position
DATA-ARRAY-ROW-COUNT={x_num_lit|_var|_col}
DATA-ARRAY-COLUMN-COUNT={x_num_lit|_var|_col}
DATA-ARRAY=array_name
[DATA-LABELS=data_labels_lit | _var | _col]
[COLOR-PALETTE=color_palette_lit | _var | _col]]
[ITEM-COLOR=(Chart_item_keyword, txt_lit |var | (r,g,b))]
[DATA-ARRAY-COLUMN-LABELS={NONE|array_name|
{({txt_lit|_var}[,{txt_lit|_var}]...)}}]
[CHART-SIZE=(chart_width_num_lit|_var,chart_depth_num_lit|_var)]
[TITLE={title_txt_lit|_var|_col}]
[SUB-TITLE={subtitle_txt_lit|_var|_col}]
[FILL={fill_lit|txt_var|_col}]
[3D-EFFECTS={3d_effects_lit|txt_var|_col}]
[BORDER={border_lit|txt_var|_col}]
[POINT-MARKERS={point_markers_lit|txt_var|_col}]
[TYPE={chart_type_lit|txt_var|_col}]
[LEGEND={legend_lit|txt_var|_col}]
[LEGEND-TITLE={legend_title_txt_lit|_var|_col}]
[LEGEND-PLACEMENT={legend_placement_lit|txt_var|_col}]
[LEGEND- PRESENTATION={legend_presentation_lit|txt_var|_col}]
[PIE-SEGMENT-QUANTITY-DISPLAY=    {pie_segment_quantity_display_lit|txt_var|_col
}]
[PIE-SEGMENT-PERCENT- DISPLAY={pie_segment_percent_display_lit
|txt_var|_col}]
[PIE-SEGMENT-EXPLODE={pie_segment_explode_lit
|txt_var|_col}]
[X-AXIS-LABEL={x_axis_label_txt_lit|_var|_col}]
[X-AXIS-MIN-VALUE={x_axis_min_value_lit|_num_lit|_var
|_col}]
[X-AXIS-MAX-VALUE={x_axis_max_value_lit|_num_lit|_var
 |_col}]
[X-AXIS-SCALE={x_axis_scale_lit|txt_var|_col}]
[X-AXIS-MAJOR-TICK-MARKS={x_axis_major_tick_marks_lit
|txt_var|_col}]
[X-AXIS-MINOR-TICK-MARKS={x_axis_minor_tick_marks_lit
|txt_var|_col}]
[X-AXIS-MAJOR-INCREMENT={x_axis_major_increment_lit
|_num_lit|_var|_col}]
[X-AXIS-MINOR-INCREMENT={x_axis_minor_increment_lit
|_num_lit|_var|_col}]
X-AXIS-TICK-MARK-PLACEMENT=
{x_axis_tick_mark_placement_lit|txt_var|_col}]
[X-AXIS-GRID={x_axis_grid_lit|txt_var|_col}]
[Y-AXIS-LABEL={y_axis_label_lit|txt_var|_col}]
[Y-AXIS-MIN-VALUE={y_axis_min_value_lit|_num_lit
|_var|_col}]
[Y-AXIS-MAX-VALUE={y_axis_max_value_lit|_num_lit
|_var|_col}]
[Y-AXIS-SCALE={y_axis_scale_lit|txt_var|_col}]
[Y-AXIS-MAJOR-TICK-MARKS={y_axis_major_tick_marks_lit
|txt_var|_col}]
[Y-AXIS-MINOR-TICK-MARKS={y_axis_minor_tick_marks_lit
|txt_var|_col}]
[Y-AXIS-MAJOR-INCREMENT={y_axis_major_increment_lit
|_num_lit|_var|_col}]
```

```
[Y-AXIS-MINOR-INCREMENT={y_axis_minor_increment_lit
|_num_lit|_var|_col}]
[Y-AXIS-TICK-MARK-PLACEMENT=
   {y_axis_tick_mark_placement_lit|txt_var|_col}]
[Y-AXIS-GRID={y_axis_grid_lit|txt_var|_col}]
```

Note. If you do not define CHART-SIZE with this command, you must define it with DECLARE-CHART.

Description

Prints a chart. Only PostScript printers or HP printers that support HPGL (generally, this is HPLaserJet 3 and higher) render chart output.

The PRINT-CHART command directs SQR to generate a chart according to the named chart, if any, and the overridden attributes, if any.

Note. PRINT-CHART can be used without referencing a named chart if all required attributes for the DECLARE-CHART are supplied in addition to all its required parameters. The PRINT-CHART command directs SQR to display the chart on the current page using the attribute values at the moment the command is executed. Manipulation of chart attribute values has no effect on the appearance of the chart after the PRINT-CHART command has been executed. For example, if you execute a PRINT-CHART with TITLE=$ttl and $ttl='Encouraging Results', and then change the value of $ttl to 'Discouraging Results' immediately afterward, then the chart is printed with first value, 'Encouraging Results'. PRINT-CHART expects the DATA-ARRAY to be organized in a particular way. See the Chart Array Field Types (fewer than four fields) table for details. PRINT-CHART fills the area defined by CHART-SIZE as much as possible while maintaining an aesthetically pleasing ratio of height to width. In cases in which the display area is not well suited to the chart display, the chart is centered within the specified region, and the dimensions are scaled to accommodate the region. Do not be alarmed if the chart does not fit exactly inside the box that you have specified. It means that SQR has accommodated the shape of the region to provide the best looking chart possible. Chart commands used to send output to a line printer are ignored. Only PostScript printers or HP printers that support Hewlett Packard's HPGL (generally, this is HP LaserJet model 3 and higher) render chart output. If you attempt to print a chart to a LaserJet printer that does not support HPGL, the HPGL command output might become part of your output, leaving one or more lines of meaningless data across your report.

All the attributes defined for DECLARE-CHART are valid for the PRINT-CHART command. PRINT-CHART has five additional parameters. The position of the chart is described with the first parameter. The data that supports the chart is defined in the additional attributes: DATA-ARRAY, DATA-ARRAY-ROW-COUNT, DATA-ARRAY-COLUMN-COUNT, and DATA-ARRAY-COLUMN-LABELS.

As mentioned, each chart type meets a specific organizational requirement. The Chart Array Field Types (fewer than four fields) table describes these requirements.

Note. If the first field in the array designated by DATA-ARRAY is of type CHAR, then the value on the x-axis is the contents of that column. If the first field is not of type CHAR, then the value of the x-axis is the row number of the array designated by DATA-ARRAY, beginning with 1. Pie charts show the character value in the legend area. Histograms show the character value on the y-axis. XY-Scatter charts do not use the character value and none is needed in the array.

Note. If a PIE chart contains many small slices, the user must set the PIE-SEGMENT-QUANTITY-DISPLAY argument, the PIE-SEGMENT-PERCENT-DISPLAY argument, or both to NO to prevent the values from one slice overwriting the values of another slice.

Chart Type	*Field 0*	*Field 1*	*Field 2*	*Field 3*
PIE	Type=char Pie segment labels, the names associated with each segment	Type=num The value associated with each pie segment	(Optional) Type=char Pie segment explode flag setting, '*Y*' or '*N*'	
LINE BAR STACKED-BAR 100%-BAR OVERLAPPED-BAR HISTOGRAM AREA STACKED-AREA 100%-AREA	Type=char X-Axis values	Type=num Series 1 Y-Axis values	(Optional) Type=num Series 2 Y-Axis values	(Optional) Type=num Series 3… Y-Axis values
XY-SCATTER-PLOT	Type=num Series 1 X-Axis values	Type=num Series 1 Y-Axis values	(Optional) Type=num Series 2 X-Axis values	(Optional) Type=num Series 2 ... Y-Axis values
FLOATING-BAR	Type=char X-Axis values	Type=num Series 1 Y-Axis offset	Type=num Series 1 Y-Axis duration	(Optional) Type=Num Series 2 ... Y-Axis offset

Chart Type	*Field 0*	*Field 1*	*Field 2*	*Field 3*	*Field 4*
HIGH-LOW-CLOSE	Type=char X-Axis values	Type=num High value	Type=num Low value	Type=num Closing value	(Optional) Type=num Opening value

Parameters

Parameter	*Description*
chart_name	Specifies the name of the chart from the DECLARE-CHART command. This name is not necessary if you specify the CHART-SIZE and all other pertinent attributes in the PRINT-CHART command.
Position	(row, column) Specifies the position of the upper left corner. Position parameters can be relative. See the POSITION command for examples of relative positioning. Document markers are not allowed. After execution, the current position is returned to this location; however, the next listing line is the next line below the bottom of the chart area. (This is different from the way the PRINT command works.)

Parameter	*Description*
DATA-ARRAY	Specifies the name of the array containing the data to be plotted. This must be the name of an array defined with CREATE-ARRAY.
DATA-ARRAY- ROW-COUNT	Specifies the number of rows or sets of data to be used from the DATA-ARRAY. If the DATA-ARRAY has a greater number of rows, only DATA-ARRAY-ROW-COUNT is included in the chart.
DATA-ARRAY- COLUMN-COUNT	Specifies the number of columns to be used from the DATA-ARRAY. If the DATA-ARRAY has a greater number of columns, only DATA-ARRAY-COLUMN-COUNT is included in the chart.
DATA-ARRAY- COLUMN-LABELS	Specifies labels for each Y-Axis value of the data set (fields) in DATA-ARRAY. These labels are displayed in the legend box. Column labels are ignored for pie charts. See the Chart Array Field Types (fewer than four fields) table for applicable fields for each type of chart. For definitions of the other arguments, see the DECLARE-CHART Command Arguments table.

Example

In this example, a pie chart is printed without explicit reference to a chart declared with DECLARE-CHART. All necessary arguments must be supplied in PRINT-CHART.

```
.
.
create-array
      name=unit_sales
      size=12
      field=product:char
      field=units:number
      field=explode:char
.
.
print-chart (15, 20)
   title                   = 'Green City Store Sales'
   sub-title               = '(Second Quarter)'
   chart-size              = (50, 28)
   type                    = pie
   data-array              = unit_sales
   data-array-column-count  = 3
   data-array-row-count     = 7
   3d-effects              = yes
   fill                    = color
```

See Also

DECLARE-CHART

PRINT-DIRECT

Syntax

```
PRINT-DIRECT
[NOLF]
[PRINTER={LINEPRINTER|POSTSCRIPT|HPLASERJET|HTML|LP|PS|HP|HT}]
{txt_lit|_var|_col}...
```

Description

Writes directly to the print output file without using the SQR page buffer.

PRINT-DIRECT can be used for special applications that cannot be accomplished directly with PRINT commands, such as initializing a page with graphics or other special sequences. Because this text is often printer-dependent and because the report can be printed on different types of printers that require different control characters, you can use the PRINTER qualifier to specify the printer type. If no PRINTER qualifier is specified, the command applies to all printer types.

When using PRINT-DIRECT in conjunction with PRINT commands, be aware that the SQR page buffer is copied to the output file only when each page is full or when a NEW-PAGE command is issued. One approach is to use PRINT-DIRECT commands inside a BEFORE-PAGE or AFTER-PAGE procedure (declared with the DECLARE-PROCEDURE command) so that they are coordinated with the information coming out of the page buffer.

Parameters

Parameter	***Description***
NOLF	Specifies that no carriage return and line feed is to be printed. By default, printed text is followed by a carriage return and line feed character.
PRINTER	Specifies the type of printer to which this text applies.
{*txt_lit* | *_var* | *_col*}	The text to be printed.

Example

The following example shows the PRINT-DIRECT command:

```
print-direct  printer=ps  '%%Page: '  $page-number
print-direct  nolf  printer=lp  reset
```

PRINT-IMAGE

Syntax

```
PRINT-IMAGE[image_name]position
[TYPE={image_type_lit|_var|_col}]
[IMAGE-SIZE=(width_num_lit|_var|_col,height_num_lit
|_var|_col)]
[SOURCE={file_name_txt_lit|_var|_col}]
```

Note. If TYPE, IMAGE-SIZE, and SOURCE are not defined in PRINT-IMAGE, they must be defined in DECLARE-IMAGE.

Description

Prints an image.

The PRINT-IMAGE command can be placed in any section of a report with the exception of the SETUP section. The image file pointed to can be any file of the proper format.

PRINT-IMAGE can be used without referencing a named image if all required attributes for the DECLARE-IMAGE are supplied in addition to all its required parameters.

Parameters

Parameter	*Description*
image_name	Specifies the name of an image specified by a DECLARE-IMAGE.
position	(row, column) Specifies the position of the upper left corner. Position parameters can be relative. See the POSITION command for examples of relative positioning. Document markers are not allowed. After execution, the current position is returned to this location. (This is different from the way the PRINT command works.)
TYPE	Specifies the image type. Types can be EPS-FILE, HPGL-FILE, GIF-FILE, JPEG-FILE, or BMP-FILE (for Microsoft Windows).
IMAGE-SIZE	Specifies the width and height of the image.
SOURCE	Specifies the name of a file containing the image.

Example

For PostScript:

```
print-image office-signature (50, 20)
print-image (50, 20)
   type            = eps-file
   source           = 'sherman.eps'
   image-size = (10, 3)
```

For Microsoft Windows:

```
print-image company-logo (+21, 25)
   type=bmp-file
   source='m:\logos\gustavs.bmp'
   image-size=(75,50)
```

See Also

DECLARE-IMAGE

PUT

Syntax

```
PUT {src_any_lit|_var|_col}...
INTO dst_array_name(element)[field[(occurs)]]...
```

Description

Moves data into an array.

Columns retrieved from the database and SQR variables or literals can be moved into an array. The array must have been created previously by the CREATE-ARRAY command.

Considerations for Using PUT

When a date variable or column is moved into a text or char array field, the date is converted to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

When a string variable, column, or literal is moved to a date array field, the string must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats as listed in the DATE Column Formats table is used, or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]' is used.

```
dst_array_name(element)
```

If array fields are listed, data is placed into each field in the sequence in which it is listed, in the occurrence specified for that field.

If array fields are not listed, data is placed into consecutive fields in the order in which they were defined in the CREATE-ARRAY command; data is copied into occurrence zero of each field of the element specified in the array.

```
field [ ( occurs ) ]
```

Array element and field occurrence numbers can be numeric literals (123) or numeric variables *(#j)*.

If no occurrence is specified, occurrence zero is used.

Parameters

Parameter	***Description***
src_any_var	The source variable or literal to be moved into the array. Numeric variables, literals, and database columns can be put into number (decimal, float, integer) fields. String variables, literals, and database columns can be put into char, text, or date fields. Date variables can be put into date, char, or text fields.

Example

In the following example, the four variables *&name*,*#count*,*$date1*, and *$code* are placed in the first four fields defined in the names array. The data is put into the *#j*th element of the array.

```
put  &name  #count $date1  $code  into  names(#j)
```

The following command places *#j2*,*#j3*, and *#j4* into the zero through 2nd occurrences of the *tot* field in the *#j* th element of the totals array.

```
put  #j2  #j3  #j4  into  totals(#j)  tot(0)  tot(1)  tot(2)
```

The following command copies *#count* into the *#j2*th occurrence of the count field in the *#j*th element of the states array.

```
put  #count  into  states(#j)  count(#j2)
```

READ

Syntax

```
READ {filenum_lit|_var|_col} INTO {any_var:length_int_lit}...[STATUS=
status_num_var]
```

Description

Reads the next record of a file into the specified variables.

Text and binary data are parsed according to the following criteria:

- Text data is any string of characters. The length of the variable name indicates how many characters to place in the variable.

  After text is transferred, trailing blanks in the variable are omitted.

- If the field was written as a date variable, then it can be read into a date variable or text variable.

  When reading a date into a date variable, it must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats as listed in the DATE Column Formats table, or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]'.

- Binary numbers can be 1, 2, or 4 bytes in length.

  They must be read into numeric variables. Note that the bytes making up the binary number must be in the standard sequence expected by your operating system.

- When the program is reading binary data, the file must be opened with the FIXED or FIXED-NOLF qualifier.

- Only the integer portion of the number is represented with binary numbers.

  To maintain the decimal portion of the number, convert the number to a string variable.

- If you use binary numbers, the file is not portable across platforms.

  Different hardware represents binary numbers differently.

The total length indicated for the variables must be less than or equal to the length of the record being read.

If no more records exist to read, the *#end-file* reserved variable is set to 1; otherwise, it is set to 0 (zero). Your program should check this variable after each READ command.

If STATUS is specified, SQR returns 0 if the read is successful; otherwise, it returns the value of *errno*, which is system-dependent.

Parameters

Parameter	***Description***
filenum_lit|_var | _col	Specifies the number assigned in the OPEN command to the file to be read.
any_var :length_int_lit	Specifies one or more variables into which data from the record that is read are to be put. *length_int_lit* specifies the length of each field of data.
STATUS	Specifies an optional variable into which a read status is returned.

Example

The following example shows several READ commands:

```
read  1  into  $name:30  $addr:30  $city:20  $state:2  $zip:5
read  3  into  $type:2  #amount:2  #rate:1  $code:5  $date:11
read  #j into  #sequence:2  $name:20  $title:15
```

The following example shows a READ command that reads two dates. One is loaded into a date variable; the other is loaded into a string variable, which is then converted to a date using the strtodate function.

```
.
.
.
declare-variable
 date $date1 $date2
 text $text
end-declare
.
.
.
read 4 into $date1:18 $text1:18
let $date2 = strtodate($text1,'SYYYYMMDDHHMISSNNN')
        or
let $date2 = strtodate($text1)
```

The following example shows a READ command with an INSERT loop:

```
begin-sql
 begin transaction
end-sql

while  1    ! Infinite loop, exited by BREAK, below.
 read  10  into  $company:40  $parent:30  $location:50
 if #end-file
  break   ! End of file reached.
 end-if
 begin-sql
  insert into comps (name, parent, location)
     values ($company, $parent, $location)
 end-sql
 add 1 to #inserts
 if #inserts >= 100
   begin-sql
    end transaction;
    begin transaction
   end-sql
 move 0 to #inserts
 end-if
end-while

begin-sql
 end transaction
end-sql
```

See Also

The OPEN, CLOSE, and WRITE commands for information about files.

ROLLBACK

Syntax

```
ROLLBACK
```

Description

An automatic rollback is performed whenever SQR ends due to program errors. ROLLBACK is useful for testing or for certain error conditions.

ROLLBACK is an SQR command and should not be used inside an SQL paragraph.

Note. The ROLLBACK command can be used with Oracle, DB2, Informix, and ODBC (Microsoft SQL Server is accessible only with SQR Server for ODBC). For Sybase and Microsoft SQL Server, use BEGIN TRANSACTION and ROLLBACK TRANSACTION within SQL paragraphs as in the following example. See the COMMIT command for an example of ROLLBACK.

Example

The following example shows the ROLLBACK command:

```
if #error-status = 1
   rollback
   stop
end-if
```

See Also

The COMMIT command.

SBTOMBS

Syntax

```
SBTOMBS { txt_var }
```

Description

Converts a single-byte character into a multibyte equivalent.

This command converts the specified string in the following way: Any occurrence of a single-byte character that also has a multibyte representation (numerals, punctuation, roman characters, and katakana) is converted. This command also converts a sequence of a kana character followed by certain grammatical marks into a single multibyte character, which combines the two elements.

Parameters

Parameter	*Description*
txt_var	Specifies the string to be converted.

See Also

The TO_MULTI_BYTE function of the LET command.

SECURITY

Syntax

```
SECURITY
[SET=(sid [,sid]...)]
[APPEND=(sid [,sid]...)]
[REMOVE=(sid [,sid]...)]
[MODE=mode]
```

Description

Enables you to mark sections of a report for security purposes.

The SECURITY command can be repeated as many times as desired for the current report. After the SECURITY command is carried out, all subsequent commands for the current report are constrained by the designated *sids* until the report ends or another SECURITY command executes.

You can use the SECURITY command wherever you use the PRINT command.

Parameters

Parameter	*Description*
SET	Sets the list of security IDs for subsequent commands. The previous list of security IDs is replaced by the specified security IDs. This argument is optional and can be used only once.
sid	Can be any string literal, column, or variable. The value is case-sensitive.
APPEND	Appends the specified security IDS to the current list. This argument is optional and can be used multiple times.
REMOVE	Removes the specified security IDS from the current list. This argument is optional and can be used multiple times.

Parameter	*Description*
MODE	Used to enable (reactivate) or disable (suspend) the security feature for the current report. This argument is optional and can be used only once.
mode	Can be any string literal, column, or variable. The value is not case-sensitive and can be either ON or OFF.

Example

The following example shows the SECURITY command:

```
Begin-Report
   Security Set=('Directors', 'Vice-Presidents')
      .
      .         ! Only Directors and VPS can see this
      .
   Security Remove=('Directors')
      .
      .         ! Only VPS can see this
      .
   Security Mode='Off'
      .
      .         ! Anybody can see this
      .
   Security Mode='On' Append=('Managers')
      .
      .         ! Only VPs and Managers can see this
      .
   Security Append=('Engineers')
      .
      .         ! Only VPs, Managers, and Engineers can see this
      .
End-report
```

SET-COLOR

Syntax

```
SET-COLOR
   [PRINT-TEXT-FOREGROUND=({color_name_lit|_var|_col|{rgb})]
   [PRINT-TEXT-BACKGROUND=({color_name_lit|_var|_col|{rgb})]
```

Description

Defines default colors.

The SET-COLOR command is allowed wherever the PRINT command is allowed. If the specified color name is not defined, SQR uses the settings for the color name 'default.' Use the color name 'none' to color for the specified area.

Parameters

Parameter	*Description*
PRINT-TEXT- FOREGROUND	Defines the color in which the text is printed.
PRINT-TEXT- BACKGROUND	Defines the color to print as a background for the text.
{*color_name_lit* | *_var* | *_col*}	A *color_name* is composed of alphanumeric characters (A–Z, 0–9), the underscore (_) character, and the hyphen (-) character. It must start with an alphabetical (A–Z) character and is case-insensitive. The name 'none' is reserved and cannot be assigned a value. A name in the format (RGBredgreenblue) cannot be assigned a value. The name 'default' is reserved and can be assigned a value. 'Default' is used during execution when a referenced color is not defined in the runtime environment.
{*rgb*}	*red_lit* | *_var* | *_col, green_lit* | *_var* | *_col, blue_lit* | *_var* | *_col* where each component is a value in the range of 000 to 255. In the BEGIN-SETUP section, only literal values are allowed. The default colors implicitly installed with SQR include: black=(0,0,0) white=(255,255,255) gray=(128,128,128) silver=(192,192,192) red=(255,0,0) green=(0,255,0) blue=(0,0,255) yellow=(255,255,0) purple=(128,0,128) olive=(128,128,0) navy=(0,0,128) aqua=(0,255,255) lime=(0,128,0) maroon=(128,0,0) teal=(0,128,128) fuchsia=(255,0,255)

Example

The following example shows the SET-COLOR command:

```
begin-setup
   declare-color-map
   light_blue = (193, 222, 229)
   end-declare
end-setup

begin-program
   alter-color-map name = 'light_blue' value = (193, 233, 230)

   print 'Yellow Submarine' ()
      foreground = ('yellow')
      background = ('light_blue')

   get-color print-text-foreground = ($print-foreground)
   set-color print-text-foreground = ('purple')
   print 'Barney' (+1,1)
   set-color print-text-foreground = ($print-foreground)
end-program
```

See Also

DECLARE-COLOR-MAP, ALTER-COLOR-MAP, GET-COLOR

SET-GENERATIONS

Syntax

SET-GENERATIONS=(*dimension,hierarchy,dimension,hierarchy,dimension,hierarchy,...,...*)

Description

Specifies dimension hierarchy for the previously declared dimension.

Returns the set of members in the dimension 'product' that are at the 5th generation in the dimension's hierarchy. (Returns all 'Brand Name' members (Generation Level 5) under the product hierarchy of 'all products.drink.alcoholic beverages.beer and wine.' This would increase the result set to a list of beers and wines.) Returns the set of members in the dimension 'time' that are at the 1st generation deep into the dimension. (Returns all 'Year' members (Generation Level 1) under the time hierarchy of '2004.Q1.2'. This reduces result set to '2004'.)

Example

The following example shows the SET-GENERATIONS command:

```
set-generations=('product',5,'time',1 )
```

SET-LEVELS

Syntax

```
Set-levels=(dimension, level, dimension, level,...,...)
```

Description

Extends the dimension hierarchy for the previously declared dimension.

Set-levels used with only the previous 'set-members' returns all members under the product hierarchy and the next two generations (Product SubCategory and Brand Name) for the product hierarchy of all products.drink.alcoholic beverages.beer and wine'. Set-levels used with the previous 'set-members' and 'set-generations' returns all members for generation levels 5 through 7 under the product hierarchy of all products.drink.alcoholic beverages.beer and wine.'

Example

The following example shows the SET-LEVELS command:

```
set-levels=('product',2 )
```

SET-MEMBERS

Syntax

```
set-members=(dimension, hierarchy, dimension, hierarchy,..., ...)
```

Description

Returns the set of members in a dimension, level, or hierarchy for which name is specified by a string.

Example

Returns the set of members in the dimension 'product' at the specific hierarchy of 'all products', at a specific level of 'drink', at a specific level of 'alcoholic beverages', at a specific level of 'beer and wine'. Returns the set of members in the dimension 'time' at the specific hierarchy of '2004', at the specific level of 'Q1', at the specific level of '2'.

```
set-members=('product','all products.drink.alcoholic beverages.beer⇒
 and wine','time','2004.Q1.2')
```

SHOW

Syntax

```
SHOW[cursor_position]
[CLEAR-SCREEN|CS|CLEAR-LINE|CL][any_lit|_var|_col]
[EDITedit_mask|NUMBER|MONEY|DATE][BOLD][BLINK]
[UNDERLINE][REVERSE][NORMAL][BEEP][NOLINE]...
```

Description

Displays one or more variables or literals on the screen. In addition, cursor control is supported for ANSI terminals.

Any number of variables and screen positions can be used in a single command. Each one is processed in sequence.

Screen locations can be indicated by either fixed or relative positions in the format (A,B), where A is the line and B is the column on the screen. A, B, or both can also be numeric variables. Relative positions depend on where the previous SHOW command ended. If the line was advanced, the screen cursor is usually immediately to the right of the previously displayed value and one line down.

Fixed or relative cursor positioning can be used only within the boundaries of the terminal screen. Scrolling off the screen using relative positioning, for example (+1,1), is not supported. Instead, use a SHOW command without any cursor position when you want to scroll. Also, you cannot mix SHOW and DISPLAY commands while referencing relative cursor positions.

The SHOW command does not advance to the next line if a cursor location (...), CLEAR-SCREEN, CLEAR-LINE, or BEEP is used. (A SHOW command without any of these arguments automatically advances the line.) To add a line advance, add (+1,1) to the end of the line or use an extra empty SHOW command.

Only ANSI terminals are supported for cursor control, screen blanking, line blanking, and display characteristics.

Dates can be contained in a date variable or column, or a string literal, column, or variable. When the program displays a date variable or column without an edit mask, the date is displayed according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the Default Database Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

When displaying a date in a string literal, column, or variable using EDIT or DATE, the string must be in the format specified by the SQR_DB_DATE_FORMAT setting, one of the database-dependent formats as listed in the Default Database Formats table, or the database-independent format 'SYYYYMMDD[HH24[MI[SS[NNNNNN]]]].

Parameters

Parameter	*Description*
cursor_position	Specifies the position on the screen to begin the display.
{CLEAR-SCREEN | CS}	Clears the screen and sets the cursor position to (1,1).
{CLEAR-LINE | CL}	Clears a line from the current cursor position to the end of the line.
{any_lit | _var | _col}	Specifies the information to be displayed.
EDIT	Shows variables under an edit mask. If the mask contains spaces, enclose it in single quotes. For additional information regarding edit masks, see the PRINT command.
NUMBER	Indicates that any_lit|_var|_col is to be formatted with the NUMBER-EDIT-MASK from the current locale. (See the ALTER-LOCALE command.) This option is not valid for date variables.
MONEY	Indicates that any_lit|_var|_col is to be formatted with the MONEY-EDIT-MASK from the current locale. (See the ALTER-LOCALE command.) This option is not valid for date variables.
DATE	Indicates that any_lit|_var|_col is to be formatted with the DATE-EDIT-MASK from the current locale. (See the ALTER-LOCALE command.) This option is not valid for numeric variables. If DATE-EDIT-MASK has not been specified, the date is displayed with the default format for that database (see the Default Database Formats table).
BOLD, BLINK, UNDERLINE, and REVERSE	Changes the display of characters on terminals that support those characteristics. Some terminals support two or more characteristics at the same time for the same text. To disable all special display characteristics, use NORMAL.
NORMAL	Disables all special display characteristics set with BOLD, BLINK, UNDERLINE, and REVERSE.
BEEP	Causes the terminal to beep.
NOLINE	Inhibits a line advance.

Example

The following program segments illustrate the various features of the SHOW command:

```
 !
 ! Show a string using an edit mask
 !
 let $ssn = '123456789'
 show $ssn edit xxx-xx-xxxx
```

Produces the following output:

```
123-45-6789
```

```
 !
 ! Show a number using an edit mask
 !
 show 1234567.89 edit 999,999,999.99
```

Produces the following output:

```
1,234,567.89
```

```
 !
 ! Show a number using the default edit mask
 !
 show 123.78
```

Produces the following output:

```
123.780000
```

```
 !
 ! Show a number using the locale default numeric edit mask
 !
 alter-locale number-edit-mask = '99,999,999.99'
 show 123456.78 number
```

Produces the following output:

```
   123,456.78
```

```
 !
 ! Show a number using the locale default money edit mask
 !
 alter-locale money-edit-mask = '$$,$$$,$$8.99'
 show 123456.78 money
```

Produces the following output:

```
$123,456.78
```

```
 !
 ! Show a date column using the locale default date edit mask
 !
 begin-select
 dcol
  from tables
 end-select
 alter-locale date-edit-mask = 'DD-Mon-YYYY'
 show &dcol date
```

Produces the following output:

```
01-Jan-2004

 !
 ! Show two values on the same line
 !
 show 'Hello' ' World'
```

Produces the following output:

```
Hello World

 !
 ! Show two values on the same line with editing of the values
 !
 let #taxes = 123456.78
 show 'You owe ' #taxes money ' in back taxes.'
```

Produces the following output:

```
You owe $123,456.78 in back taxes.
```

The following program illustrates the usage of additional options of the SHOW command. Only terminals that support the ANSI escape characters can use the cursor control, screen blanking, line blanking, and display attributes.

```
begin-program
 !
 ! Produces a menu for the user to select from
 !
 show clear-screen
      (3,30)  bold 'Accounting Reports for XYZ Company' normal
      (+2,10)  '1. Monthly Details of Accounts'
      (+1,10)  '2. Monthly Summary'
      (+1,10)  '3. Quarterly Details of Accounts'
      (+1,10)  '4. Quarterly Summary'

 !
 ! Show a line of text and numerics combined
 !
 show (+2,1)
      'The price is ' #price edit 999.99

      '    Total = ' #total edit 99999.99

 !
  ! Put an error message on a particular line
 !
 show (24,1) clear-line 'Error in SQL.  Please try again.' beep
end-program
```

See Also

The LET command for information about copying, editing, or converting fields.

The EDIT parameter of the PRINT command for a description of the edit masks.

The ALTER-LOCALE command for a description of the arguments NUMBER-EDIT-MASK, MONEY-EDIT-MASK, and DATE-EDIT-MASK.

DISPLAY

STOP

Syntax

```
STOP [QUIET]
```

Description

The STOP command halts SQR and executes a ROLLBACK command (not in Sybase, Microsoft SQL Server , or Informix). All report page buffers are flushed if they contain data; however, no headers or footers are printed and the AFTER-PAGE and AFTER-REPORT procedures are not executed.

STOP is useful in testing.

Parameters

Parameter	*Description*
QUIET	Causes the report to finish with the "SQR: End Of Run" message, instead of ending with an error message.

Example

The following example shows the STOP command:

```
if #error-status = 1
   rollback
   stop
else
   commit
   stop quiet
end-if
```

STRING

Syntax

```
STRING {src_any_lit|_var|_col}...BY {delim_txt_lit|_var|_col}
INTO dst_txt_var
```

Description

Concatenates a list of variables, columns, or literals into a single text variable. Each member of the list is separated by the specified delimiter string.

The destination string must not be included in the list of source strings.

Parameters

Parameter	*Description*
{src_any_lit|_var|_col}	Specifies one or more fields to be concatenated, separated by the *delim_txt_lit|_var|_col* character or characters, and placed into the *dst_txt_var* variable. If the source is a date variable or column, it is converted to a string according to the following rules: For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting. If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table. For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting. If this has not been set, SQR uses the format listed in the Default Database Formats table. For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting. If this has not been set, SQR uses the format as listed in the TIME Column Formats table.
{delim_txt_lit|_var|_col}	Specifies one or more characters to be used as separator characters between the source fields.
dst_txt_var	Specifies the destination field for the concatenated result.

Example

The following example shows the STRING command:

```
string  &name  &city  &state  &zip  by  ' - '  into  $show-info
                  ! Result:  Sam Mann - New York - NY - 11287
string  &cust_num  &entry-date  &total  by  ','  into  $cust-data
                  ! Result:  100014,12-MAR-04,127
                  ! Use null delimiter.
string  &code1  &code2  &code3  by  ''  into  $codes123
                  ! Result:  AGL
```

See Also

The UNSTRING command for additional information.

The "||" concatenation operator in the Operators table under the LET command.

SUBTRACT

Syntax

```
SUBTRACT {src_num_lit|_var|_col} FROM dst_num_var[ROUND=nn]
```

Description

Subtracts the first value from the second and moves the result into the second field.

When dealing with money-related values (dollars and cents), use decimal variables rather than float variables. Float variables are stored as double-precision floating-point numbers, and small inaccuracies can appear when you are subtracting many numbers in succession. These inaccuracies can appear due to the way floating point numbers are represented by different hardware and software implementations.

Parameters

Parameter	*Description*
{src_num_lit | _var | _col}	Is subtracted from the contents of *dst_num_var*.
dst_num_var	Contains the result after execution.
ROUND	Rounds the result to the specified number of digits to the right of the decimal point. For float variables this value can be from 0 (zero) to 15. For decimal variables, this value can be from 0 to the precision of the variable. For integer variables, this argument is not appropriate.

Example

The following example shows the SUBTRACT command:

```
subtract  1  from  #total      ! #total - 1
subtract &discount from #price ! #price - &discount
```

See Also

The ADD command for more information.

The LET command for information about complex arithmetic expressions.

TOC-ENTRY

Syntax

```
TOC-ENTRY
TEXT={src_txt_lit|_var|_col}
[LEVEL={level_num_lit|_var|_col}]
```

Description

Enter the text in the table of contents at the desired level.

Parameters

Parameter	*Description*
TEXT	Specifies the text to be placed in the table of contents.
LEVEL	Specifies the level at which to place the text. If this argument is not specified, the value of the previous level is used.

Example

The following example shows the TOC-ENTRY command:

```
toc-entry text = &heading
toc-entry text = &caption level=2
```

See Also

The DECLARE-TOC command.

UNSTRING

Syntax

```
UNSTRING {{src_txt_lit|_var|_col}|{src_date_var|_col}}
BY {delim_txt_lit|_var|_col}
INTO dst_txt_var...
```

Description

Copies portions of a string into one or more text variables.

Each substring is located usingby means of the specified delimiter. The source string must not be included in the list of destination strings.

If more destination strings than substrings are found in the source strings, the extra destination strings are each set to an empty string.

If more substrings are found in the source string than in the destination strings, the extra substrings are not processed. The programmer is responsible for ensuring that enough destination strings are specified.

If the source is a date variable or column, it is converted to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the Default Database Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Parameters

Parameter	*Description*
{src_txt_lit|_var|_col}|{ src_date_var|_col}	Specifies the source field to be parsed.
delim_txt_lit|_var|_col	Specifies one or more characters to be used to delimit the fields within { *src_txt_lit*|*_var*|*_col*}|{*src_date_var*|*_col*}
dst_txt_var	Specifies one or more destination fields to receive the results.

Example

The following example shows the UNSTRING command:

```
unstring $show-info by ' - ' into $name $city $state $zip
unstring $cust-data by ',' into $cust_num $entry-date $total
```

See Also

STRING, EXTRACT

The substr and instr functions in the Miscellaneous Functions table under the LET command.

UPPERCASE

Syntax

```
UPPERCASE txt_var
```

Description

Converts a string variable to uppercase.

Parameters

Parameter	*Description*
txt_var	Specifies a text variable to be converted to uppercase.

Example

The following example shows the UPPERCASE command:

```
input  $state  'Enter state abbreviation'
uppercase  $state     ! Force uppercase.
```

See Also

The upper function in the Miscellaneous Functions table under the LET command.

USE

Syntax

```
USE database
```

Description

Uses the named database rather than the default database associated with your username. (Sybase and Microsoft SQL Server only.)

Use USE in the SETUP section only. When used, it must appear at the top of your report, before any queries are defined.

To reference more than one database in a program, specify secondary databases explicitly. For example:

```
from sqdb.sqr.customers
```

You cannot issue the Sybase or Microsoft SQL Server USE command from within an SQL paragraph.

Parameters

Parameter	*Description*
database	Specifies the name of the database to use.

Example

The following example shows the USE command:

```
begin-setup
 use pubs
end-setup
```

See Also

Chapter 4, "Invoking SQR Execute," page 269

USE-COLUMN

Syntax

```
USE-COLUMN {column_number_int_lit|_var|_col}
```

Description

Sets the current column.

The column must have been defined previously with the COLUMNS command.

To stop printing within columns, use a column number of 0 (zero). Printing returns to normal; however, the columns remain defined for subsequent NEXT-COLUMN or USE-COLUMN commands.

Parameters

Parameter	*Description*
{column_number_int_lit|_var|_col}	Specifies the number of the defined column (not the location on the page). For example, if five columns are defined, the *column_number_int_lit|_var|_col* can be 1 to 5.

Example

The following example shows the USE-COLUMN command:

```
use-column  3     ! Print total in 3rd column.
print  #total  ()  999,999
use-column  0     ! End of column printing.
```

USE-PRINTER-TYPE

Syntax

USE-PRINTER-TYPE *printer-type*

Description

Sets the printer type to be used for the current report.

The USE-PRINTER-TYPE command sets or alters the printer type to be used for the current report. The USE-PRINTER-TYPE command must appear before the first output is written to that report. If output has already been written to the report file, the USE-PRINTER-TYPE command is ignored.

Parameters

Parameter	***Description***
printer-type	Specifies the printer type to be used for the current report. See DECLARE-PRINTER for valid types.

Example

The following example shows the USE-PRINTER-TYPE command:

```
use-report customer_orders
use-printer-type PostScript
print (1, 1) 'Customer Name: '
print () $customer_name
```

See Also

DECLARE-PRINTER, DECLARE-REPORT, USE-REPORT

USE-PROCEDURE

Syntax

```
USE-PROCEDURE
[FOR-REPORTS=(report_name1[,report_namei]...)]
[BEFORE-REPORT=procedure_name[(arg1[,argi]...)]]
[AFTER-REPORT=procedure_name[(arg1[,argi]...)]]
[BEFORE-PAGE=procedure_name[(arg1[,argi]...)]]
[AFTER-PAGE=procedure_name[(arg1[,argi]...)]]
```

Description

Changes the procedure usage.

The USE-PROCEDURE command must be issued in the PROGRAM or PROCEDURE sections of an SQR program. USE-PROCEDURE is a runtime command; its compile-time equivalent is DECLARE-PROCEDURE. You can use the command as often as required to change to the necessary procedures required by the reports. If you issue multiple USE-PROCEDURE commands, each remains in effect for that report until altered by another USE-PROCEDURE command for that report. In this way, you can use one to change common procedures for ALL reports and others to change unique procedures for individual reports. The referenced procedures can accept arguments.

If no FOR-REPORTS is specified, ALL is assumed. Initially, the default for each of the four procedure types is NONE. If a procedure is defined in one DECLARE-PROCEDURE for a report, that procedure is used unless NONE is specified.

You can change the BEFORE-REPORT only before the first output is written to that report, because that causes the BEFORE-REPORT procedure to be executed.

Parameters

Parameter	***Description***
FOR-REPORTS	Specifies the reports that are to use these procedures. This argument is required only for a program with multiple reports. If you are writing a program that produces a single report, you can ignore this argument.
BEFORE-REPORT	Specifies a procedure to execute at the time of execution of the first command, which causes output to be generated. You can use the command, for example, to create a report heading.
AFTER-REPORT	Specifies a procedure to execute just before the report file is closed at the end of the report. This argument can be used to print totals or other closing summary information. If no report was generated, the procedure does not execute.
BEFORE-PAGE	Specifies a procedure to execute at the beginning of every page, just before the first output command for the page. It can be used, for example, to set up page totals.

Parameter	*Description*
AFTER-PAGE	Specifies a procedure to execute just before each page is written to the file. This argument can be used, for example, to display page totals. You can also specify arguments to be passed to the procedure. Arguments can be any variable, column, or literal.

Example

The following example shows the USE-PROCEDURE command:

```
use-procedure               ! These procedures will

for-reports=(all)           ! be used by all reports

before-report=report_heading

after-report=report_footing
use-procedure               ! These procedures will

for-reports=(customer)      ! be used by the customer

before-page=page_setup      ! report

after-page=page_totals
use-procedure               ! The after-report

for-reports=(summary)       ! procedure will be

after-report=none           ! disabled for the

                            ! summary report
```

See Also

DECLARE-PROCEDURE

USE-REPORT

Syntax

```
USE-REPORT {report_name_lit|_var|_col}
```

Description

For programs with multiple reports, enables the user to switch between reports.

The USE-REPORT command specifies which report files the subsequent report output is to be written to. An application can contain several USE-REPORT statements to control several reports.

You must specify the report name and report characteristics in a DECLARE-REPORT paragraph and in the associated DECLARE-LAYOUT and DECLARE- PRINTER paragraphs.

Parameters

Parameter	*Description*
{report_name_lit|_var|_col}	Specifies the report to become the current report. All subsequent PRINT and PRINT-DIRECT statements are written to this report until the next USE-REPORT is encountered.

Example

The following example shows the USE-REPORT command:

```
use-report customer_orders
use-printer-type PostScript
print (1, 1) 'Customer Name: '
print () $customer_name
```

See Also

DECLARE-REPORT, DECLARE-LAYOUT, DECLARE-PRINTER, USE-PRINTER-TYPE

WHILE

Syntax

```
WHILE logical_expression
```

The general format of a WHILE command is:

```
WHILE logical_expression
SQR_commands...
[BREAK]
SQR_commands...
END-WHILE
```

Description

Begins a **WHILE ... END-WHILE** loop.

The WHILE loop continues until the condition being tested is FALSE.

An expression returning 0 (zero) is considered FALSE; an expression returning nonzero is TRUE.

BREAK causes an immediate exit of the WHILE loop; SQR continues with the command immediately following END-WHILE.

WHILE commands can be nested to any level and can include or be included within IF and EVALUATE commands.

Parameters

Parameter	*Description*
logical_expression	A valid logical expression. See the LET command for a description of logical expressions.

Example

This example shows an IF nested within a WHILE:

```
while #count < 50
    do get_statistics
    if #stat_count = 100
       break     ! Exit WHILE loop.
    end-if
    add 1 to #count
end-while
```

You can use single numeric variables in your expression to make your program more readable, for example, when using flags:

```
move 1 to #have_data
...
 while #have_data
   ...processing...
end-while
```

This example sets up an infinite loop:

```
while 1
    ...processing...
    if ...
       break     ! Exit loop
    end-if
end-while
```

Any complex expression can be used in the WHILE command, as shown in this example:

```
while #count < 100 and (not #end-file or isnull(&state))
   ...
end-while
```

See Also

The LET command for a description of expressions.

WRITE

Syntax

```
WRITE {filenum_lit|_var|_col} FROM
{{{txt_lit|_var|_col}|{date_var|_col}|num_col}
[:len_int_lit]}|{num_lit|_var:len_int_lit}}...
[STATUS=status_num_var]
```

Description

Writes a record to a file from data stored in variables, columns, or literals.

The file must already be opened for writing.

If length is specified, the variable is either truncated at that length or padded with spaces to that length. If length is not specified (for string variables or database columns), the current length of the variable is used.

When you are writing numeric variables, the length argument is required. Only 1-byte, 2-byte, or 4-byte binary integers are written. Floating point values are not supported directly in the WRITE command. However, you can first convert floating point numbers to strings and then write the string.

When you are writing binary data, you must open the file using the FIXED or FIXED-NOLF qualifiers. The file is not portable across platforms because binary numbers are represented differently.

When writing a date variable or column, the date is converted to a string according to the following rules:

- For DATETIME columns and SQR DATE variables, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting.

  If this has not been set, SQR uses the first database-dependent format as listed in the Default Database Formats table.

- For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format listed in the DATE Column Formats table.

- For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting.

  If this has not been set, SQR uses the format as listed in the TIME Column Formats table.

Text literals take the length of the literal.

Files opened for writing are treated as having variable-length records. If you need a fixed-length record, specify a length for each variable written to the file.

The total length of the variables and literals being written must not be greater (but can be less) than the record length specified when the file was opened. Records are not padded, but are written with the total length of all variables in the WRITE command.

If STATUS is specified, SQR returns 0 if the write is successful; otherwise, it returns the value of *errno*, which is system-dependent.

Parameters

Parameter	*Description*
filenum_lit|_var |_col	Specifies the number assigned in the OPEN command to the file to be written.
{{txt_lit|_var|_col}| {date_var|_col}|num_col} [:len_int_lit]}| {num_lit|_var:len_int_lit}	Specifies one or more variables to be written. *len_int_lit* specifies the length of each field of data.
STATUS	Specifies an optional variable into which a write status is returned.

Example

The following example shows the WRITE command:

```
write  5  from  $name:20  $city:15  $state:2
write  17  from  $company  ' - '  $city  ' - '  $state  ' '  $zip
write  #j2  from  #rate:2  #amount:4  #quantity:1
move  #total to $tot 99999.99          ! Convert floating point to string.
write  1  from  $tot
let $date1 = datenow()                ! Put the current date and time
                                      ! into DATE variable
write 3 from $date1:20
```

See Also

OPEN, CLOSE, READ

Chapter 3

Generating HTML Output

This chapter describes the procedures that enable SQR for PeopleSoft to generate HTML output. You can publish the output on an internet, intranet, or extranet website. An SQR program without HTML procedures has limited HTML capabilities; therefore, adding HTML procedures to an SQR program enhances the appearance of the HTML output.

This chapter discusses:

- HTML general purpose procedures.
- HTML heading procedures.
- HTML highlighting procedures.
- HTML hypertext link procedures.
- HTML list procedures.
- HTML table procedures.

See Also

PeopleTools 8.52: SQR for PeopleSoft Developers, "Generating and Publishing HTML from an SQR Program"

HTML General Purpose Procedures

The following table describes HTML general purpose procedures:

Procedure	*Description*

html_br	Produces the specified number of line breaks in a paragraph by using the HTML
	 tag. The paragraph continues onto the next line.
	Syntax:
	`html_br(number count, string attributes)`
	Count: The number of HTML
	 tags that are inserted.
	Attributes: The HTML attributes that are incorporated inside the HTML
	 tag.
	Example:
	`print 'Here is some text' ()`
	`do html_br(3,'')`
	`print 'Here is some text three lines down' ()`

html_center	Marks the start of text to be centered in the HTML document by using the HTML <CENTER> tag. You can also accomplish this by using the SQR print statement with CENTER specified in the code.
	Syntax:
	`html_center()`
	Attributes: The HTML attributes that are incorporated inside the HTML <CENTER> tag.
	Example:
	`do html_center('')`
	`print 'Here is some centered text' ()`
	`do html_center_end`

html_center_ end	Marks the end of text that was previously specified as centered. Syntax: `html_center_end`

html_hr	Produces a horizontal divider between sections of text by using the HTML <HR> tag.
	Syntax:
	`html_hr(string attributes)`
	Attributes: The HTML attributes that are incorporated inside the HTML <HR> tag.
	Example:
	`print 'Here is some text' ()`
	`do html_hr('')`
	`print 'Text after a horizonal divider' ()`

Procedure	*Description*

html_img	Inserts an image by using the HTML <IMG> tag. You can also do this by using the PRINT-IMAGE command; however, the html_img procedure enables you to specify the full set of available HTML attributes.
	Syntax:
	`html_img(string attributes)`
	Attributes: The HTML attributes that are incorporated inside the HTML <IMG> tag. Some common attributes include:
	• src: The URL of the image to be inserted.
	Example: src=/images/abc.gif
	• height: The height of the image in pixels.
	Example: height=200
	• width: The width of the image in pixels.
	Example: width=400
	Example:
	`do html_img('src="/images/stop.gif"')`

html_nobr	Marks the start of text that cannot be wrapped by using the HTML <NOBR> tag.
	Syntax:
	`html_nobr`
	Example:
	`do html_nobr('')`
	`print 'Long line of text that shouldn't wrap' ⇒ ()`
	`do html_nobr_end`

html_nobr_end	Marks the end of text that cannot be wrapped by using the HTML </NOBR> tag. Syntax: `html_nobr_end`

html_on	Activates the HTML procedures. Call this procedure at the start of an SQR program; otherwise, the HTML procedures are not activated. After the HTML procedures are activated, format the appearance of the web page by using the various HTML procedures.
	Syntax:
	`html_on`
	Example:
	`do html_on`

Procedure	*Description*

html_p	Marks the start of a new paragraph by using the HTML <P> tag.
	Syntax:
	`html_p(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <P> tag. A common attribute is align = left|right|center, which specifies the alignment of the paragraph.
	Example:
	`do html_p('ALIGN=RIGHT')`
	`print 'Right aligned text' (1,1)`
	`do html_p_end`
	`print 'Normally aligned text' (+1,1)`

html_p_end	Marks the end of a paragraph by using the HTML </P> tag. The end of a paragraph is typically implied and the procedure is technically not needed; however, specifying it for completeness is a good practice. Syntax: `html_p_end`

html_set_body_attributes	Specifies the attributes that are incorporated into the HTML <BODY> tag. Call this procedure at the start of the SQR program.
	Syntax:
	`html_set_body_attributes(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <BODY> tag. Some common attributes include:
	• background: Specifies the image to display in the background of the web page.
	Example: background=/images/logo.gif
	• bgcolor=#rrggbb: Specifies the background color of the web page.
	Example: bgcolor=#80FFF
	Example:
	`do html_set_body_attributes('BACKGROUND="⇒ /images/x.gif"')`

Procedure	Description

html_set_head_tags	Specifies the tags that are incorporated between the HTML <HEAD> and </HEAD> tags. These tags are empty by default. One common tag to set is the HTML <TITLE> tag, which specifies the title to display for the web page. Call this procedure at the start of the SQR program.
	Syntax:
	`html_set_head_tags(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated between the HTML <HEAD> and </HEAD> tags.
	Example:
	`do html_set_head_tags('<TITLE>My Report<⇨ /TITLE>')`

HTML Heading Procedures

The following table describes HTML heading procedures:

Procedure	Description

html_h1	Marks the start of text for heading level one by using the HTML <H1> tag. This heading text appears more prominently than heading level two text.
	Syntax:
	`html_h1(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <H1> tag.
	Example:
	`do html_h1('')`
	`print 'This is a heading' ()`
	`do html_h1_end`

html_h1_end	Marks the end of text for heading level one by using the HTML </H1> tag. Syntax: `html_h1_end`

Procedure	*Description*

html_h2	Marks the start of text for heading level two by using the HTML <H2> tag. This heading text appears less prominently than heading level one text and more prominently than heading level three text.
	Syntax:
	`html_h2(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <H2> tag.
	Example:
	`do html_h2('')`
	`print 'This is a heading' ()`
	`do html_h2_end`

html_h2_end	Marks the end of text for heading level two by using the HTML </H2> tag. Syntax: `html_h2_end`
html_h3	Marks the start of text for heading level three by using the HTML <H3> tag. This heading text appears less prominently than heading level two text and more prominently than heading level four text. **Note.** This heading level is the default value. Syntax: `html_h3(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <H3> tag.
html_h3_end	Marks the end of text for heading level three by using the HTML </H3> tag. Syntax: `html_h3_end`
html_h4	Marks the start of text for heading level four by using the HTML <H4> tag. This heading text appears less prominently than heading level three text and more prominently than heading level five text. Syntax: `html_h4(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <H4> tag.

Procedure	*Description*
html_h4_end	Marks the end of text for heading level four by using the HTML </H4> tag. Syntax: `html_h4_end`
html_h5	Marks the start of text for heading level five by using the HTML <H5> tag. This heading text appears less prominently than heading level four text and more prominently than heading level six text. Syntax: `html_h5(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <H5> tag.
html_h5_end	Marks the end of text for heading level five by using the HTML </H5> tag. Syntax: `html_h5_end`
html_h6	Marks the start of text for heading level six by using the HTML <H6> tag. This heading text appears less prominently than heading level five text. Syntax: `html_h6(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <H6> tag.
html_h6_end	Marks the end of text for heading level six by using the HTML </H6> tag. Syntax: `html_h6_end`

HTML Highlighting Procedures

The following table describes HTML highlighting procedures:

Procedure	*Description*

html_blink	Marks the start of blinking style text by using the HTML <BLINK> tag.
	Syntax:
	`html_blink(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <BLINK> tag.
	Example:
	`do html_blink('')`
	`print 'This is blinking text' ()`
	`do html_blink_end`

html_blink_end	Marks the end of blinking style text by using the HTML </BLINK> tag. Syntax: `html_blink_end`

html_cite	Marks the start of text in citation style by using the HTML <CITE> tag.
	Syntax:
	`html_cite(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <CITE> tag.
	Example:
	`do html_cite('')`
	`print 'This is a citation' ()`
	`do html_cite_end`

html_cite_end	Marks the end of text in citation style by using the HTML </CITE> tag. Syntax: `html_cite_end`

html_code	Marks the start of text in code style by using the HTML <CODE> tag.
	Syntax:
	`html_code(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <CODE> tag.
	Example:
	`do html_code('')`
	`print 'Here is code style text' ()`
	`do html_code_end`

Procedure	*Description*
html_code_end	Marks the end of text in code style by using the HTML </CODE> tag. Syntax: `html_code_end`

html_kbd	Marks the start of text in keyboard input style by using the HTML <KBD> tag.
	Syntax:
	`html_kbd(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <KBD> tag.
	Example:
	`do html_kbd('')`
	`print 'Here is keyboard style text' ()`
	`do html_kbd_end`

html_kbd_end	Marks the end of text in keyboard style by using the HTML </KBD> tag. Syntax: `html_kbd_end`

html_samp	Marks the start of text in sample style by using the HTML <SAMP> tag.
	Syntax:
	`html_samp(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <SAMP> tag.
	Example:
	`do html_samp('')`
	`print 'Here is sample style text' ()`
	`do html_samp_end`

html_samp_end	Marks the end of text in sample style by using the HTML </SAMP> tag. Syntax: `html_samp_end`

Procedure	*Description*

html_strike	Marks the start of text in strikethrough style by using the HTML <STRIKE> tag.
	Syntax:
	`html_strike(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <STRIKE> tag.
	Example:
	`do html_strike('')`
	`print 'Here is strikethrough text' ()`
	`do html_strike_end`

html_strike_end	Marks the end of text in strikethrough style by using the HTML </STRIKE> tag. Syntax: `html_strike_end`

html_sub	Marks the start of text in subscript style by using the HTML <SUB> tag.
	Syntax:
	`html_sub(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <SUB> tag.
	Example:
	`print 'Here is' ()`
	`do html_sub('')`
	`print 'subscript text' ()`
	`do html_sub_end`

html_sub_end	Marks the end of text in subscript style by using the HTML </SUB> tag. Syntax: `html_sub_end`

html_sup	Marks the start of text in superscript style by using the HTML <SUP> tag.
	Syntax:
	`html_sup(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <SUP> tag.
	Example:
	`print 'Here is' ()`
	`do html_sup('')`
	`print 'superscript text' ()`
	`do html_sup_end`

Procedure	Description
html_sup_end	Marks the end of text in superscript style by using the HTML </SUP> tag. Syntax: `html_sup_end`

HTML Hypertext Link Procedures

The following table describes HTML hypertext link procedures:

Procedure	Description

html_a	Marks the start of a hypertext link by using the HTML <A> tag. When the user clicks the area with the hypertext link, the web browser switches to the specified HTML document.
	Syntax:
	`html_a(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <A> tag. At a minimum, you should define the href attribute, which specifies the URL of an HTML document.
	Some common attributes include:
	• href: Indicates where the hypertext link points.
	Example: href=home.html
	• name: Indicates an anchor to which a hypertext link can point.
	Example: name=marker1
	Example:
	The anchor is positioned at the top of the document. The first hypertext link points to the HTML document named otherdoc.html. The second hypertext link points to the anchor named TOP:
	`do html_a('NAME=TOP')`
	`do html_a_end`
	`print 'At the top of document' ()`
	`do html_br(20, '')`
	`do html_a('HREF=otherdoc.html')`
	`print 'Go to other document' ()`
	`do html_a_end`
	`do html_p('')`
	`do html_a('HREF=#TOP')`
	`print 'Go to top of document' ()`
	`do html_a_end`

Procedure	Description
html_a_end	Marks the end of a hypertext link by using the HTML </A> tag. Syntax: `html_a_end`

HTML List Procedures

The following table describes HTML list procedures:

Procedure	Description
html_dd	Marks the start of a definition in a definition list by using the HTML <DD> tag. Syntax: `html_dd(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <DD> tag.
html_dd_end	Marks the end of a definition in a definition list by using the HTML </DD> tag. The end of a definition in a definition list is typically implied and not needed; however, specifying it for completeness is a good practice. Syntax: `html_dd_end`

html_dir	Marks the start of a directory list by using the HTML <DIR> tag.
	Syntax:
	`html_dir(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <DIR> tag.
	Example:
	`do html_dir('')`
	`do html_li('')`
	`print 'First item' ()`
	`do html_li('')`
	`print 'Second item' ()`
	`do html_li('')`
	`print 'Last item' ()`
	`do html_dir_end`

Procedure	*Description*
html_dir_end	Marks the end of a directory list by using the HTML </DIR> tag. Syntax: `html_dir_end`

html_dl	Marks the start of a definition list by using the HTML <DL> tag. A definition list displays a list of terms and definitions. The term appears before and to the left of the definition. Use the html_dt procedure to display a term. Use the html_dd procedure to display a definition.
	Syntax:
	`html_dl(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <DL> tag.
	Example:
	`do html_dl('')`
	`do html_dt('')`
	`print 'A Daisy' ()`
	`do html_dd('')`
	`print 'A sweet and innocent flower.' ()`
	`do html_dt('')`
	`print 'A Rose' ()`
	`do html_dd('')`
	`print 'A very passionate flower.' ()`
	`do html_dl_end`

html_dl_end	Marks the end of a definition list by using the HTML </DL> tag. Syntax: `html_dl_end`
html_dt	Marks the start of a term in a definition list by using the HTML <DT> tag. Syntax: `html_dt(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <DT> tag.
html_dt_end	Marks the end of a term in a definition list by using the HTML </DT> tag. Syntax: `html_dt_end`

Procedure	*Description*
html_li	Marks the start of a list item by using the HTML <LI> tag. Syntax: `html_li(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <LI> tag.
html_li_end	Marks the end of a list item by using the HTML </LI> tag. The end of a list item is typically implied and not needed; however, you should specify it for completeness. Syntax: `html_li_end`

html_menu	Marks the start of a menu by using the HTML <MENU> tag. Use the html_li procedure to identify each item in a list.
	Syntax:
	`html_menu(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <MENU> tag.
	Example:
	`do html_menu('')`
	`do html_li('')`
	`print 'First item' ()`
	`do html_li('')`
	`print 'Second item' ()`
	`do html_li('')`
	`print 'Last item' ()`
	`do html_menu_end`

html_menu_end	Marks the end of a menu by using the HTML </MENU> tag. Syntax: `html_menu_end`

Procedure	*Description*

html_ol	Marks the start of an ordered list by using the HTML <OL> tag. Each item in the list typically appears indented to the right with a number to the left. Use the html_li procedure to identify each item in a list.
	Syntax:
	`html_ol(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <OL> tag.
	Example:
	`do html_ol('')`
	`do html_li('')`
	`print 'First item' ()`
	`do html_li('')`
	`print 'Second item' ()`
	`do html_li('')`
	`print 'Last item' ()`
	`do html_ol_end`

html_ol_end	Marks the end of an ordered list by using the HTML </OL> tag. Syntax: `html_ol_end`

html_ul	Marks the start of an unordered list by using the HTML <UL> tag. Each item in the list typically appears indented to the right with a bullet to the left. Use the html_li procedure to identify each item in a list.
	Syntax:
	`html_ul(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <UL> tag.
	Example:
	`do html_ul('')`
	`do html_li('')`
	`print 'First item' ()`
	`do html_li('')`
	`print 'Second item' ()`
	`do html_li('')`
	`print 'Last item' ()`
	`do html_ul_end`

html_ul_end	Marks the end of an unordered list by using the HTML </UL> tag. Syntax: `html_ul_end`

HTML Table Procedures

The following table describes HTML table procedures:

Procedure	*Description*
html_caption	Marks the start of a table caption by using the HTML <CAPTION> tag. Syntax: `html_caption(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <CAPTION> tag.
html_caption_end	Marks the end of a table caption by using the HTML </CAPTION> tag. The end of a table caption is typically implied and not needed; however, you should specify it for completeness. Syntax: `html_caption_end`

Procedure	*Description*

html_table	Marks the start of a table by using the HTML <TABLE> tag.
	Syntax:
	`html_table(string attributes)`
	Attributes: Defines the HTML attributes that are incorporated inside the HTML <TABLE> tag.
	Some common attributes include:
	• border: Specifies that a border appears around each cell of a table.
	• width: Specifies the width of an entire table in pixels.
	• cols: Specifies the number of columns in a table.
	Example: COLS=4
	Example: Displaying database records in a tabular format. The html_caption_end, html_tr_end, html_td_end, and html_th_end procedures are used for completeness; however, they are typically implied and not needed.
	`! start the table & display the column headings`
	`do html_table('border')`
	`do html_caption('')`
	`print 'Customer Records' (1,1)`
	`do html_caption_end`
	`do html_tr('')`
	`do html_th('')`
	`print 'Cust No' (+1,1)`
	`do html_th_end`
	`do html_th('')`
	`print 'Name" (,10)`
	`do html_th_end`
	`do html_tr_end`
	`! display each record`
	`begin-select`
	`do html_tr('')`
	`do html_td('')`
	`cust_num   (1,1,6) edit 099999`
	`do html_td_end`
	`do html_td('')`
	`name     (1,10,25)`
	`do html_td_end`
	`do html_tr_end`
	`next-listing skiplines=1 need=1`
	`from customers`
	`end-select`
	`! end the table`
	`do html_table_end`

html_table_end	Marks the end of a table by using the HTML </TABLE> tag. Syntax: `html_table_end`

Procedure	*Description*
html_td	Marks the start of a new column in a table row by using the HTML <TD> tag. This tag specifies that the text that follows appears in the column. Syntax: `html_td(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <TD> tag.
html_td_end	Marks the end of a column in a table by using the HTML </TD> tag. The end of a column is typically implied and not needed; however, you should specify it for completeness. Syntax: `html_td_end`
html_th	Marks the start of a new column header in a table row by using the HTML <TH> tag. This tag specifies that the text that follows appears as the header of the column. Syntax: `html_th(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <TH> tag.
html_th_end	Marks the end of a column header in a table by using the HTML </TH> tag. The end of a column header is typically implied and not needed; however, you should specify it for completeness. Syntax: `html_th_end`
html_tr	Marks the start of a new row in a table by using the HTML <TR> tag. Syntax: `html_tr(string attributes)` Attributes: Defines the HTML attributes that are incorporated inside the HTML <TR> tag.
html_tr_end	Marks the end of a row by using the HTML </TR> tag. The end of a row in a table is typically implied and not needed; however, you should specify it for completeness. Syntax: `html_tr_end`

Chapter 4

Invoking SQR Execute

SQR Execute is a runtime program that enables you to run a previously compiled SQR program.

This chapter discusses how to:

- Run SQR Execute
- Use SQR Execute flags

Running SQR Execute

To run SQR Execute, enter the following command. (If you are using Microsoft Windows, invoke SQRWT rather than SQRT.)

```
SQRT [program][connectivity][flags...][args...][@file...]
```

Using SQR Execute Flags

The following table describes the SQR Execute command-line flags. See the SQR command-line arguments section for information about program, connectivity, args, and @file.

See Chapter 1, "Understanding SQR for PeopleSoft," SQR Command-Line Arguments, page 5.

Flag	*Description*
-A	Appends the output to an existing output file of the same name. If the file does not exist, it creates a new one. This flag is useful when you want to run the same report more than once, but you want to create only one output file.
-C	(Microsoft Windows) Specifies that the Cancel dialog box appears while the program is running so that you can easily terminate the program.

Flag	*Description*
-BURST:{xx}	Specifies the type of bursting to be performed. -BURST:T generates the table of contents file only. -BURST:S generates the report output according to the symbolic table of contents entries that are set in the program with the LEVEL argument of the TOC-ENTRY command. In -BURST:S[{l}], *l* is the level at which to burst. The -BURST:S setting is equivalent to -BURST:S1. -BURST:P generates the report output by report page numbers. In -BURST:P[{l} , {s} [, {s}] …]] , *l* is the number of logical report pages that each .htm file contains and *s* is the page selection: *n, n-, m, -m,* or *n-*. The -BURST:P setting is equivalent to -BURST:P0,1- when using -PRINTER:HT or -BURST:P1 when using -PRINTER:EH. **Note.** -BURST:P and -BURST:S require -PRINTER:EH or -PRINTER:HT. The page range selection feature of -BURST:P requires -PRINTER:HT. -BURST:T requires -PRINTER:HT.
-CB	(Microsoft Windows) Forces the communication box.
-Dnn	(Non-Microsoft Windows) Causes SQR to display the report output on the terminal at the same time that it is being written to the output file. The *nn* variable is the maximum number of lines to display before pausing. If you do not enter a number after -D, the display scrolls continuously. The printer type must be LP; otherwise, SQR does not display any output. If the program is producing more than one report, SQR displays only the first report.
-DBdatabase	(Sybase) Causes the SQR program to use the specified database, overriding any USE command in the SQR program.
-E[file]	Directs error messages to the named file or to the default file, program.err. If no errors occur, no file is created.

Flag	*Description*
-EH_APPLETS:dir	Specifies the directory location of the enhanced HTML applets. The default directory for theses applets is IMAGES. **Note.** The -EH flags in this table are applicable only when either the -PRINTER:EH or -PRINTER:EP flag is specified.
-EH_BQD	Generates a {report}.bqd file from the report data. Also associates a BQD icon with {report}.bqd in the navigation bar.
-EH_BQD:file	Associates the BQD icon with the specified file.
-EH_BROWSER:xx	Specifies the target browser. When set to ALL, SQR automatically determines which browser is being used, invokes a browser-specific file, and generates HTML that is designed for that browser. When set to BASIC, SQR generates HTML that is suitable for all browsers. When set to IE, SQR generates HTML that is designed for Microsoft Internet Explorer.
-EH_CSV	Generates a {report}.csv file from the report data.
-EH_CSV:file	Associates the CSV icon with the specified file.
-EH_CSVONLY	Creates a CSV file but does not create an HTML file.
-EH_FULLHTML:xx	Switches between HTML 3.0 and HTML 3.2. When set to TRUE, SQR generates HTML 3.2. When set to FALSE, SQR generates HTML 3.0.
-EH_Icons:dir	Specifies the directory for the referenced icons.
-EH_LANGUAGE:xx	Sets the language that is used for the HTML navigation bar. You can specify English, French, German, Portuguese, Spanish, Japanese, Simplified Chinese, and Korean.
-EH_PDF	Associates a PDF icon with {report}.pdf in the navigation bar.

Flag	*Description*
-EH_Scale:{nn}	Sets the scaling factor from 50 to 200.
-F[file|directory]	Overrides the default output file name, program.lis. The default action places the program.lis file in the same directory as the program.sqr file. To use the current directory, specify -F without an argument. To change the name of the output file, specify -F with the new name. If the new name does not specify a directory, the file is created in the current directory. The output file is not created until data is actually printed on the page. If no data is printed, no output file is created. Specify the file name and directory for different operating systems: • UNIX\Linux • MVS
-GPRINT=YES | NO	(MVS) -GPRINT=YES causes ANSI control characters to be written to the first column of each record of the SQR output file.
-ID	Displays the copyright banner on the console.
-KEEP	In addition to .lis files, creates an .spf file for each report that the program generates.
-NOLIS	Prevents the creation of .lis files. Instead, creates .spf files.
-O[file]	Directs log messages to the specified file or to programt.log if no file is specified. By default, the sqr.log file is used in the current working directory.
-P	(MVS) Suppresses printer control characters from column 1.
-PB	(Informix) Causes column data to retain trailing blanks.

Flag	*Description*
-PRINTER:xx	Causes printer type *xx* to be used when creating output files: • EH • EP • HP • HT • LP • PD • PS • WP Types LP, HP, and PS produce files with the .lis extension. Types EH and HT produce .htm file output. Type HT produces files in HTML version 2.0 with the report content inside the <PRE></PRE> tags. Type EH produces reports in which content is fully formatted with HTML version 3.0 or 3.2 tags. In Microsoft Windows systems, the WP extension sends the output to the default Microsoft Windows printer. To specify a nondefault Microsoft Windows printer, enter -PRINTER:WP:{printer name}. The {printer name} is the name that is assigned to the printer. For example, to send output to a Microsoft Windows printer named NewPrinter, use - PRINTER:WP:NewPrinter. If the printer name has spaces, enclose the entire argument in quotes. If you also want to create an .spf file, use -KEEP.
-S	Requests that the status of all cursors appear at the end of the report run. Status includes the text of each SQL statement, the number of times that each was compiled and run, and the total number of rows selected.
-Tnn	Specifies that you want to test the report for *nn* pages. SQR ignores all Order By clauses in Select statements to save time during testing. If the program is producing more than one report, SQR stops after the specified number of pages that are defined for the first report have been printed.

Flag	*Description*
-T{B|Z|BZ|ZB}	(IBM MVS and DB2): • -TB prevents SQR from removing trailing blanks from database character columns. • -TZ prevents SQR from removing trailing zeros from the decimal portion of numeric columns. • -TBZ or -TZB prevents both. (Windows/DB2, Sybase CT-Lib, and ODBC): -TB trims trailing blanks from database character columns. **Note.** The -TB flag only has an effect if SQR is connecting to either a DB2, Sybase CT LIB, or ODBC (MSS) database. Confusingly, the behavior of the -TB command-line flag varies depending on your platform. If you are using one of the previously mentioned databases and running SQR on z/OS, the -TB flag behaves as follows: If you do not use the -TB flag, trailing blanks are trimmed. If you do use the -TB flag, trailing blanks are not trimmed. If you are running SQR on any other platform, the behavior of -TB is the opposite: If you do not use the -TB flag, trailing blanks are not trimmed. If you do use the -TB flag, trailing blanks are trimmed.
-Vserver	(Sybase) Uses the named server.
-XB	(Non-Microsoft Windows) Suppresses the SQR banner and the SQR.... End of Run message.
-XCB	(Microsoft Windows) Does not use the communication box.

Flag	*Description*
-XL	Prevents SQR from signing in to the database. Programs that are run in this mode cannot contain any SQL statements. -XL enables you to run SQR without accessing the database. You still must supply at least an empty slash (/) in the command line as a placeholder for the connectivity information. For example: `sqr myprog / -xl` Some database files must be available for SQR to run whether SQR signs in to the database or not. See information about your particular operating system and database to determine which files you need.
-XMB	(Microsoft Windows) Disables the error message display so that a program can be run without interruption by error message boxes. Error messages are sent only to an .err file. See the -E flag for more information.
-XNAV	Prevents SQR from creating the navigation bar in .htm files that are generated with -PRINTER:HT. This occurs when only a single .htm file is produced. Multiple .htm files that are generated from a single report always contain the navigation bar.
-XP	(Sybase DBLib) Prevents SQR from creating temporary stored procedures. See BEGIN-SELECT for more information.
-XTB	Preserves the trailing blanks in an .lis file at the end of a line.
-XTOC	Prevents SQR from generating the table of contents for the report. SQR ignores this flag when -PRINTER:EH or - PRINTER:HT is also specified.
-ZIF{file}	Sets the full path and name of the SQR initialization file, sqr.ini.
-ZIV	Invokes the SPF Viewer after generating the program.spf file. This flag implicitly invokes the -KEEP flag to create program.spf. In the case of multiple output files, only the first report file is passed to the viewer.

Flag	Description
-ZMF{file}	Specifies the full path and name of the SQR error message file, sqrerr.dat.

See Also

Chapter 1, "Understanding SQR for PeopleSoft," SQR Command-Line Arguments, page 5

Chapter 5

Using SQR Print

This chapter provides an overview of SQR Print and discusses how to:

- Generate output from the command line.
- Use SQR Print command-line flags.
- Generate output in Microsoft Windows.

Understanding SQR Print

SQR Print enables you to create printer-specific reports for any of the file types that SQR supports. SQR Print converts portable printer-independent files (spfs) into printer-specific files. SQR and SQRT (or SQRWT for Windows) create .spf files when you use the -KEEP and -NOLIS command-line flags.

Generating Output from the Command Line

To begin running SQR Print, enter the following command. (If you are in Microsoft Windows, invoke SQRWP rather than SQRP.)

```
SQRP [spf-file] [flags...]
```

The following table describes the *spf-file* and *flags* variables.

SQR Print writes an .lis file with the same name as the .spf file but with an lis extension. You can override this name with the -F command-line flag.

The -PRINTER command-line flag specifies the printer type. SQR offers these printer type options:

- Line printer
- HP LaserJet
- PostScript
- HTML
- Enhanced HTML
- Adobe PDF
- Enhanced HTML and Adobe PDF

If the report contains graphics and you select a line printer, then SQR Print will ignore graphic elements (such as lines, boxes, and charts) and print only the text.

Using SQR Print Command-Line Flags

The following table describes the SQR Print command-line flags:

Command-line Flag	*Description*
-A	Appends the output to an existing output file of the same name. If the file does not exist, it creates a new one. This flag is useful when you want to run the same report more than once, but you want to create only one output file.
-BURST:{xx}	Specifies the type of bursting to be performed. -BURST:T generates the table of contents file only. -BURST:S generates the report output according to the symbolic table of contents entries that are set in the program with the LEVEL argument of the TOC-ENTRY command. In -BURST:S[{l}], *l* is the level at which to burst. The -BURST:S setting is equivalent to -BURST:S1. -BURST:P generates the report output by report page numbers. In -BURST:P[{l}, {s} [, {s}] …]], *l* is the number of logical report pages that each .htm file contains and *s* is the page selection: *{n}*, *{n}-{m}*, *-{m}*, or *n-*. The -BURST:P setting is equivalent to -BURST:P0,1- when using -PRINTER:HT or -BURST:P1 when using -PRINTER:EH. **Note.** -BURST:P and -BURST:S require -PRINTER:EH or -PRINTER:HT. The page range selection feature of -BURST:P requires -PRINTER:HT. BURST:T requires -PRINTER:HT.
-D*nn*	(Non-Microsoft Windows) Displays the report output on the terminal at the same time that it is being written to the output file. The variable *nn* is the maximum number of lines that appear before pausing. If you do not enter a number after -D, the displayed output scrolls continuously. The printer type must be LP or no output appears. If the program is producing more than one report, then only the first report appears.

Command-line Flag	*Description*
-E[file]	Directs error messages to the named file or to the default file, program.err. If no errors occur, no file is created. **Note.** The following -EH flags are applicable only when either the −PRINTER:EH or −PRINTER:EP flag is specified.
-EH_APPLETS:dir	Specifies the directory location of the enhanced HTML applets. The default directory for theses applets is IMAGES.
-EH_BQD	Generates a {report}.bqd file from the report data. Also associates a BQD icon with {report}.bqd in the navigation bar.
-EH_BQD:file	Associates the BQD icon with the specified file.
-EH_BROWSER:xx	Specifies the target browser. When set to ALL, SQR automatically determines which browser is being used, invokes a browser-specific file, and generates HTML that is designed for that browser. When set to BASIC, SQR generates HTML that is suitable for all browsers. When set to IE, SQR generates HTML that is designed for Microsoft Internet Explorer.
-EH_CSV	Generates a {report}.csv file from the report data.
-EH_CSV:file	Associates the CSV icon with the specified file.
-EH_CSVONLY	Creates a CSV file, but does not create an HTML file.
-EH_FULLHTM L:xx	Switches between HTML 3.0 and HTML 3.2. When set to TRUE, SQR generates HTML 3.2. When set to FALSE, SQR generates HTML 3.0
-EH_Icons:dir	Specifies the directory in which the HTML should find the referenced icons.

Command-line Flag	*Description*
-EH_LANGUAGE:xx	Sets the language that is used for the HTML navigation bar. You can specify English, French, German, Portuguese, Spanish, Japanese, Simplified Chinese, or Korean.
-EH_PDF	Associates a PDF icon with {report}.pdf in the navigation bar.
-EH_Scale:{nn}	Sets the scaling factor from 50 to 200.
-F[file | directory]	Overrides the default output file name, program.lis. The default action places program.lis in the same directory as the program.sqr file. To use the current directory, specify -F without an argument. To change the name of the output file, specify -F with the new name. If the new name does not specify a directory, the file is created in the current directory. The output file is not created until data is actually printed on the page. If no data is printed, no output file is created. Specify these file names and directories for different operating systems: • UNIX\Linux • MVS
-ID	Displays the copyright banner on the console.
-O[file]	Directs log messages to the specified file or to programt.log if no file is specified. By default, the sqr.log file is used in the current working directory.
-P	(IBM MVS) Suppresses printer control characters from column 1.

Command-line Flag	*Description*
-PRINTER:xx	Causes SQR to use printer type *xx* when creating output files: • EH • EP • HP • HT • LP • PD Types LP, HP, and PS produce files with the .lis extension. Types EH and HT produce .htm file output. In Microsoft Windows systems, the WP extension sends output to the default Microsoft Windows printer. To specify a nondefault Microsoft Windows printer, enter -PRINTER:WP:{printer name}. The {printer name} is the name that is assigned to the printer. For example, to send output to a Microsoft Windows printer named NewPrinter, use -PRINTER:WP:NewPrinter. If your printer name has spaces, enclose the entire argument in quotes. If you also want to create an .spf file, use -KEEP.
-XB	(Non-Microsoft Windows) Suppresses the SQR banner and the SQR.... End of Run message.
-XNAV	Prevents SQR from creating the navigation bar in .htm files that are generated with -PRINTER:HT. This occurs when only a single .htm file is produced. Multiple .htm files that are generated from a single report always contain the navigation bar.
-XTB	Preserves the trailing blanks in a .lis file at the end of a line.
-XTOC	Prevents SQR from generating the table of contents for the report. This flag is ignored when -PRINTER:EH or -PRINTER:HT is also specified.
-ZIF{file}	Sets the full path and name of the SQR initialization file, sqr.ini.

Command-line Flag	*Description*
-ZMF{file}	Specifies the full path and name of the SQR error message file, sqr-err.dat.

Generating Output in Microsoft Windows

In Microsoft Windows, the SQR Print graphical user interface enables you to generate output from the Print dialog box. In addition to the previously mentioned SQR Print output options, you can also select a Microsoft Windows printer. This selection spools the SQR output to your default Microsoft Windows printer or print server.

To generate output in Microsoft Windows:

1. Select File, Print.

   The Print dialog box appears.

2. Under Generate output for, select the option next to the type of output that you want.
3. Specify a file path.
4. Select the Print to file check box.
5. Click OK.

Chapter 6

Avoiding Older SQR Commands

This chapter provides an overview of older SQR commands and discusses how to use them.

Understanding Older SQR Commands

Avoid incorporating the commands covered in this chapter in your SQR code. Even though they are technically supported by this release, they do not interact well with the current SQR lexicon and may cause unpredictable results. SQR may not support these commands in future releases, so you should remove these commands from your code as soon as feasible.

If your code still contains older SQR commands, refer to this table as you replace them with their updated alternatives:

Old Command	*Alternative to Use Instead*
BEGIN-REPORT (END-REPORT)	BEGIN-PROGRAM (END-PROGRAM)
DATE-TIME	datenow function
DECLARE PRINTER	DECLARE-PRINTER
DECLARE PROCEDURE	DECLARE-PROCEDURE
DOLLAR-SYMBOL	ALTER-LOCALE
GRAPHIC FONT	ALTER-PRINTER
MONEY-SYMBOL	ALTER-LOCALE
NO-FORMFEED	DECLARE-LAYOUT
PAGE-SIZE	DECLARE-LAYOUT

Old Command	*Alternative to Use Instead*
PRINTER-DEINIT	DECLARE-PRINTER
PRINTER-INIT	DECLARE-PRINTER
PRINT … CODE	PRINT … CODE-PRINTER

Note. Two older commands, DECLARE PRINTER and DECLARE PROCEDURE, do not contain hyphens. The new commands, DECLARE-PRINTER and DECLARE-PROCEDURE, contain hyphens.

Using Older SQR Commands

This section discusses each of the older SQR commands.

BEGIN-REPORT

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use BEGIN-PROGRAM.

Syntax

Use this syntax:

```
BEGIN-REPORT
```

Description

Begins a report.

After processing the commands in the SETUP section, SQR starts running the program at the BEGIN-REPORT section. The PROGRAM section typically contains a list of DO commands, though you can also use other commands. This section is the only required section in an SQR program.

Example

For example:

```
begin-report
   do startup
   do main
   do finish
end-report
```


DATE-TIME

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use the datenow function in the LET command.

Syntax

Use this syntax:

```
DATE-TIME  position [date_format[col_var]]
```

Description

Retrieves the current date and time from the local machine (or from the database for Oracle and some IBM DB2 platforms) and places it in the output file at the specified position or into a column variable.

If *col_var* is specified, then *date_format* must be supplied and the current date and time is retrieved each time this command is run. Otherwise, the date is retrieved only at the program start, and the same date and time is printed each time.

If *date_format* is not specified, the date is returned in the default format for that database. The following table provides the default date-time formats for SQR-supported databases:

Database	*Default Date-Time Format*
Oracle	DD-Mon-YYYY HH:MI PM
Informix	YYYY-MM-DD HH:MI YYYY-MM-DD HH:MI:SS.NNN
IBM DB2	YYYY-MM-DD-HH:MI YYYY-MM-DD-HH:MI:SS.NNNNNN
Sybase	DD-MON-YYYY HH:MI

Some databases have two default formats. The first format prints the date-time, as in the following example:

```
date-time (+1,1)
```

The second format retrieves the date-time into a column variable:

```
date-time () '' &date1
```

For databases with only one default format, that format is always used in either of these cases.

See the table showing miscellaneous functions under the LET command for information about the valid edit mask format codes.

Parameters

position	Specifies the position for printing the date.
date_format	Represents a string literal containing the date format mask.
col_var	Places the retrieved date-time into a column variable rather than in the output file.

Example

For example:

```
date-time (1,50)  MM/DD/YY
date-time (1,1) 'Day Mon DD, YYYY'
date-time ()  HH:MI  &time
date-time (+1,70) 'MON DD YYYY HH24:MI' &datetime
date-time (#i, #j) 'YYYY-MM-DD' &date1
```

See the $current-date reserved and datenow functions that are described in the table showing miscellaneous functions under the LET command.

See ALTER-LOCALE.

DECLARE PRINTER

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use DECLARE-LAYOUT and DECLARE-PRINTER.

Syntax

Use this syntax:

```
DECLARE PRINTER
[TYPE=printer_type_lit]
[ORIENTATION=orientation_lit]
[LEFT-MARGIN=left_margin_num_lit]
[TOP-MARGIN=top_margin_num_lit]
[LINE-SIZE=line_size_num_lit]
[CHAR-SIZE=char_size_num_lit]
[LINES-INCH=lines_inch_int_lit]
[CHARS-INCH=chars_inch_num_lit]
[POINT-SIZE=point_size_num_lit]
[FONT-TYPE=font_type_txt_lit]
[SYMBOL-SET=symbol_set_id_lit]
[STARTUP-FILE=file_name_txt_lit]
[FONT=font_int_lit]
[BEFORE-BOLD=before_bold_string_txt_lit]
[AFTER-BOLD=after_bold_string_txt_lit]
```

Description

Specifies the printer type and sets printer characteristics.

Use the DECLARE PRINTER command either in the SETUP section or in the body of the report. Generally, you should use it in the SETUP section. However, if you do not know what type of printer you will be using until the report is run, or if you need to change some of the arguments depending on user selection, you can put several DECLARE PRINTER commands in the body of the report and run the one that you need.

The following arguments take effect only once, upon execution of the first PRINT command, and thereafter have no effect even if changed:

```
LINE-SIZE
CHAR-SIZE
LINES-INCH
CHARS-INCH
ORIENTATION
```

SQR maps its line and column positions on the page by using a grid that is determined by the LINE-SIZE and CHAR-SIZE (or LINES-INCH and CHARS-INCH) arguments. Each printed piece of text is placed on the page by means of this grid. Because the characters in proportional fonts vary in width, a word or string may be wider than the horizontal space that you have allotted, especially in words containing uppercase letters. To account for this behavior, you can either move the column position in the PRINT statement or indicate a larger CHAR-SIZE value in the DECLARE PRINTER command.

Arguments

The following table describes the arguments for the DECLARE PRINTER command:

Argument	*Choice or Measure*	*Default Value*	*Description*
TYPE	LINE-PRINTER, POSTSCRIPT, HPLASERJET	LINE-PRINTER	SQR creates output that is specific to each printer. Line printer files generally contain ASCII characters and can be viewed by a text editor. PostScript files contain ASCII characters, but you need to know PostScript to understand what will appear on the printer. HP LaserJet files are binary files and cannot be edited or viewed.
ORIENTATION	PORTRAIT, LANDSCAPE	PORTRAIT	Portrait pages are printed vertically. Landscape pages are printed horizontally. Printing in landscape mode on HP LaserJet printers requires landscape fonts.

Argument	*Choice or Measure*	*Default Value*	*Description*
LEFT-MARGIN	inches	0.5	This argument does not apply to line printers. This is the amount of blank space to leave at the left side of the page.
TOP-MARGIN	inches	0.5	This argument does not apply to line printers. This is the amount of blank space to leave at the top of the page.
LINE-SIZE	points	12	This argument does not apply to line printers. This is the size of each SQR line on the page. There are 72 points per inch. If LINE-SIZE is not specified, it follows the value for POINT-SIZE, if specified. The default value of 12 points yields 6 lines per inch.
CHAR-SIZE	points	7.2	This argument does not apply to line printers. This is the size of each SQR horizontal character column on the page (for example, the distance between the locations [1,12] and [1,13]). If CHAR-SIZE is not specified and the point size is less than 8.6, CHAR-SIZE is set to 4.32, which yields 16.6 characters per inch. The default value of 7.2 yields 10 characters per inch.
LINES-INCH	lines	6	This argument does not apply to line printers. This is an alternate way of indicating the, in lines per inch rather than in points (as in LINE-SIZE).

Argument	*Choice or Measure*	*Default Value*	*Description*
CHARS-INCH	characters	10	This argument does not apply to line printers. This is an alternate way of indicating the width of each SQR character column, in characters per inch rather than in points (as in CHAR-SIZE).
POINT-SIZE	points	12	This argument does not apply to line printers. This is the beginning size of the selected font.
FONT-TYPE	PROPORTIONAL, FIXED	Depends on the font	This argument applies only to HP LaserJet printers and must be specified only for font types that are not listed as being available for HP LaserJet printers in SQR in the previous DECLARE-PRINTER section.
SYMBOL-SET	HP defined sets	0U	This argument applies only to HP LaserJet printers. The default value, 0U, is for the ASCII symbol set. Additional symbol sets exist. See *HP LaserJet Technical Reference Manual.*
STARTUP-FILE	filename	POSTSCRI.STR	This argument applies only to PostScript printers. Use it to specify an alternate startup file. Unless otherwise specified, the default startup file is located in the directory that is specified by the environment variable SQRDIR.

Argument	*Choice or Measure*	*Default Value*	*Description*
FONT	font_number	3	This is the font number of the typeface to use. For HP LaserJet printers, this is the typeface value as defined by Hewlett-Packard. See *HP LaserJet Technical Reference Manual.* For PostScript printers, SQR supplies a list of fonts and arbitrary font number assignments in the postscri.str file. The font numbers are the same as those for HP LaserJet printers, wherever possible, so that you can use the same font number for reports to be printed on both types of printers. You can modify the font list in postscri.str to add or delete fonts. Read the postscri.str file for instructions. See the fonts that are listed as being available for HP LaserJet printers in SQR in the previous DECLARE-PRINTER section. See also the table that lists the fonts that are available in the SQR postscri.str file, also in the DECLARE-PRINTER section.

Argument	*Choice or Measure*	*Default Value*	*Description*
BEFORE-BOLD	any string	None	The BEFORE-BOLD and AFTER- BOLD arguments are for line printers only. They specify the character string to turn bold on and off. If the string contains blank characters, enclose it in single quotation marks. To specify nonprintable characters, such as ESC, enclose the decimal value within angle brackets as follows: BEFORE-BOLD=<27>[r ! Turn on bold AFTER-BOLD=<27>[u ! Turn it off These arguments work with the BOLD argument of the PRINT command.
AFTER-BOLD	any string	None	See BEFORE-BOLD.

The font that you choose—its orientation, typeface, and point size—must be an internal font (available in a font cartridge) or downloaded to the printer.

For fonts that are not listed as being available for HP LaserJet printers in SQR in the "DECLARE-PRINTER" section in the "SQR Command Reference" chapter, you must indicate the font style by using the FONT-TYPE argument; otherwise, the printer cannot select the correct typeface.

DECLARE PROCEDURE

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use DECLARE-PROCEDURE.

Syntax

Use this syntax:

```
DECLARE PROCEDURE
[BEFORE-REPORT=procedure_name]
[AFTER-REPORT=procedure_name]
[BEFORE-PAGE=procedure_name]
[AFTER-PAGE=procedure_name]
```

Description

Defines specific event procedures.

Use the DECLARE PROCEDURE command either in the SETUP section or in the body of the report. You can use the command as often as you like.

If you issue multiple DECLARE PROCEDURE commands, the last one takes precedence. In this way, you can turn procedures on and off while a report is running. The referenced procedures do not take any arguments; however, they may be local. In addition, they can print only into the body of the report; that is, they cannot print into the header and footer areas.

Parameters

BEFORE-REPORT	Specifies a procedure to run at the time of the first PRINT command. For example, you use this to create a report heading.
AFTER-REPORT	Specifies a procedure to run just before the report file is closed at the end of the report. Use this to print totals or other closing summary information. If no report was generated, the procedure does not run.
BEFORE-PAGE	Specifies a procedure to run at the beginning of every page, just before the first PRINT command for the page. For example, you use this to set up page totals.
AFTER-PAGE	Specifies a procedure to run just before each page is written to the file. For example, you use this to display page totals.

Example

For example:

```
declare procedure
   before-page=page_setup
   after-page=page_totals
```

DOLLAR-SYMBOL

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use ALTER-LOCALE.

Syntax

Use this syntax:

```
DOLLAR-SYMBOL  new_symbol
```

Description

Redefines the currency symbol within numeric edit masks.

The dollar sign ($) is the default currency symbol for coding edit masks in the program that prints on report listings. DOLLAR-SYMBOL provides a way to change that symbol for both the edit mask and for printing.

To change the symbol that prints on the report, use MONEY-SYMBOL in the PROCEDURE section. Use DOLLAR-SYMBOL and MONEY-SYMBOL together to configure SQR programs and the reports that they produce.

Use this command only in the SETUP section.

Note. The MONEY-SYMBOL command has the same effect as these options of the ALTER-LOCALE command: MONEY-SIGN and MONEY-SIGN-LOCATION=LEFT.

The following table lists the characters that DOLLAR-SYMBOL cannot take:

Type	*Character*
Numbers	0 8 9
Alphabetical	b e n r v B E N R V
Symbols	. , - + ! * _ ` < > ( )

Parameters

new_symbol Specifies a new, single character to be used in edit masks instead of the dollar sign ($).

Example

For example:

```
begin-setup
 dollar-symbol £      ! Define £ as the currency symbol
end-setup
begin-procedure
...
print #amount () edit £££,999.99
...
end-procedure
```

In the previous example, if you used the dollar sign in the edit mask after defining the dollar symbol as £, the following error message appears:

```
Bad numeric 'edit' format: $$$,999.99
```

See the ALTER-LOCALE command for a description of other locale-specific parameters.

GRAPHIC FONT

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use ALTER-PRINTER and DECLARE-PRINTER to set the FONT, FONT-TYPE, POINT-SIZE, and PITCH.

Syntax

Use this syntax:

```
GRAPHIC  ()
FONT  { font_number_int_lit|_var }
[point_size_int_lit|_var[{1|0}
[pitch_int_lit|_var]]]
```

Description

Changes a font.

Parameters

font_number For HP LaserJet printers, the specified font must be installed in the printer. For PostScript printers, the font must be defined in the postscri.str file.

point_size If *point_size* is omitted, the size from the most recent DECLARE-PRINTER or GRAPHIC FONT command is used.

{1|0} This argument is for HP LaserJet printers only. It is needed only if you are using a font that SQR does recognize. (See the fonts that are listed as being available for HP LaserJet printers in SQR in the DECLARE-PRINTER section of the chapter "SQR Command Reference.") A 1 indicates a proportional font, and a 0 indicates a fixed-pitch font. The default is proportional.

pitch If the specified font is fixed pitch, also indicate the pitch in characters per inch.

Example

For example:

```
graphic () font  23  8.5      ! Century Schoolbook, 8.5 points
graphic () font  6  12  0  10 ! Letter Gothic, 12 points,
                              ! fixed, 10 characters per inch
graphic () font :#font_number :#point_size
```

See ALTER-PRINTER and DECLARE-PRINTER for information about setting and changing the FONT, FONT-TYPE, POINT-SIZE, and PITCH.

MONEY-SYMBOL

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use ALTER-LOCALE.

Syntax

Use this syntax:

```
MONEY-SYMBOL new_symbol
```

Description

Redefines the currency symbol to be printed.

To change the symbol that prints on the report, use the MONEY-SYMBOL in the PROCEDURE section. When the MONEY-SYMBOL is set, that value is used until the next MONEY-SYMBOL command runs.

Use DOLLAR-SYMBOL and MONEY-SYMBOL together to configure SQR programs and the reports that they produce.

To indicate a nonedit character, surround its decimal value with angle brackets (<>). See the table under the DOLLAR-SYMBOL command for characters that cannot be used with MONEY-SYMBOL.

Note. The MONEY-SYMBOL command has the same effect as the MONEY-SIGN and MONEY-SIGN-LOCATION=LEFT options of the ALTER- LOCALE command.

Parameters

new_symbol	Specifies a new, single character to replace the dollar sign ($) or DOLLAR-SYMBOL character on the printed report.

Example

For example:

```
begin-setup
 dollar-symbol £! Define £ as the

   ! currency symbol
end-setup
begin-procedure! If #Amount=1234.56
...
money-symbol £
print #Amount () Edit £££,999.99   ! Prints as: £1,234.56
...
money-symbol $
print #Amount () Edit £££,999.99   ! Prints as: $1,234.56
...
money-symbol
print #Amount () Edit £££,999.99   ! Prints as:  1,234.56
...
end-procedure
```

See the DOLLAR-SYMBOL and ALTER-LOCALE commands.

NO-FORMFEED

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use the FORMFEED parameter of the DECLARE-LAYOUT command.

Syntax

Use this syntax:

```
NO-FORMFEED
```

Description

Prevents form-feed characters from being written to the output file.

NO-FORMFEED is useful for certain types of reports; for example, flat file output. It is used only in the SETUP section.

Do not write form-feed control characters directly into the output file between pages.

Example

For example:

```
begin-setup
  no-formfeed
end-setup
```

See Also

See the FORMFEED qualifier in DECLARE-LAYOUT.

PAGE-SIZE

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use the MAX-LINES and MAX-COLUMNS parameters of the DECLARE-LAYOUT command.

Syntax

Use this syntax:

```
PAGE-SIZE  page_depth_num_lit  page_width_num_lit
```

Description

Sets the page size.

If you are printing multiple reports, you must use the PAPER-SIZE parameter of the DECLARE-LAYOUT command.

This command is used in the SETUP section only.

Specify the page depth in lines and the page width in columns. An average report that is printed on 8 1/2 by 11 inch paper might have a page size of 60 lines by 80 columns. A 3-inch by 5-inch sales lead card might have a size of 18 by 50.

If the page size is not specified, the default of 62 lines by 132 columns is used.

For line printers, SQR stores one complete page in a buffer before writing the page to the output file when you issue a NEW-PAGE command or when a page overflow occurs.

You can define a page to be 1 line deep and 4,000 characters wide, which you can use for writing large flat files, perhaps for copying to magnetic tape. Each time a NEW-PAGE occurs, one record is written. Use the NO-FORMFEED command in the SETUP section to suppress form-feed characters between pages.

Use a page width that is at least one character larger than the rightmost position that will be written. This prevents unwanted wrapping when printing. When the last column position on a line is printed, the current position becomes the first position of the next line. This can cause confusion when using relative line positioning with the NEXT-LISTING command. Having a wider page than necessary does not waste any file space because SQR trims trailing blanks on each line before writing the report file.

Determine the size of the internal page buffer that stores a complete page in memory by multiplying the page depth by the width in the PAGE-SIZE command. For personal computers, the page buffer is limited to 64K bytes. On other computers, the page buffer is limited only by the amount of memory that is available.

Example

For example:

```
begin-setup
  page-size  57  132! 57 lines long by 132 columns wide
end-setup
```

See Also

See the PAPER-SIZE parameter of the DECLARE-LAYOUT command.

PRINT … CODE

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use PRINT … CODE-PRINTER.

If you use CODE, the sequence is assumed to be for the printer type that is specified in the DECLARE-REPORT or for the default printer, if none is specified.

Syntax

Use this syntax:

```
PRINT . . . CODE
```


Parameters

CODE	CODE is a qualifier that may be discontinued in a future release. Use CODE-PRINTER instead.

PRINTER-DEINIT

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use the RESET-STRING parameter of the DECLARE-PRINTER command.

Syntax

Use this syntax:

```
PRINTER-DEINIT  initialization_string
```

Description

Sends control or other characters to the printer at the end of a report.

Specify nondisplay characters by placing their decimal values inside angle brackets. For example, <27> is the ESC or escape character.

The PRINTER-DEINIT command is used only in the SETUP section and is designed for use with line printers. It has limited functionality with HP LaserJet and PostScript printers.

Example

For example:

```
begin-setup
  printer-deinit<27>[7J ! Reset the printer
end-setup
```

See Also

See the ENCODE command for another method of printing nondisplay characters. See the chr function in the table listing miscellaneous functions under the LET command in the chapter "SQR Command Reference."

PRINTER-INIT

You should no longer use this command because it may be discontinued in a future release. To use the newer SQR functionality, use the INIT-STRING parameter of the DECLARE-PRINTER command.

Syntax

Use this syntax:


```
PRINTER-INIT initialization_string
```

Description

Sends control or other characters to the printer at the beginning of a report.

Specify nondisplay characters by placing their decimal values inside angle brackets. For example, <27> is the ESC or escape character.

The PRINTER-INIT command is used only in the SETUP section and is designed for use with line printers. It has limited functionality with HP LaserJet and PostScript printers.

Example

For example:

```
begin-setup
  printer-init<27>[7J ! Set the printer
end-setup
```

See Also

See the ENCODE command for another method of printing nondisplay characters. See the chr function in the table listing miscellaneous functions under the LET command in the chapter "SQR Command Reference."

Chapter 7

Using the PSSQR.INI File and the PSSQR Command Line

This chapter discusses the:

- pssqr.ini installation process.
- Default settings section.
- Processing-Limits section.
- Environment sections.
- Locale section.
- Fonts section.
- HTML-Images section.
- PDF Fonts section.
- PDF Fonts: Exclusion Ranges section.
- TrueType Fonts section.
- Enhanced-HTML section.
- Colors section.
- The use of pssqr.exe command-line options.

Installing PSSQR.INI

The pssqr.ini file is the initialization file for SQR for PeopleSoft. SQR uses the settings and parameters in this file during the compile and execution phases.

The installation process installs a default initialization file called pssqr.ini, which is located in the SQR directory under the <PS_CFG_HOME> directory. The installation process also installs files named pssqr<language_cd>.ini, which are used to create language-specific configurations.

Microsoft Windows Platforms

In Microsoft Windows, SQR looks for the initialization file in the following locations in this order:

1. The file name specified by the -ZIF{file} command-line flag.
2. The directory in which the executable image resides.
3. The Microsoft Windows system directory.

z/OS

In z/OS, SQRINI is required during initialization. You must specify the dataset member &PSHLQ..SQRSRC(PSSQRINI) in the JCL to start the SQR process.

All Other Platforms

In all other platforms, SQR looks for the initialization file in the following locations in this order:

1. The file name specified by the -ZIF{file} command-line flag.
2. The current working directory.
3. The directory specified using the SQRDIR environment variable.

   SQR automatically sets up SQRDIR.

You can make changes or additions to the pssqr.ini file.

This example shows the format of the pssqr.ini file:

```
; Comments are lines that start with a semicolon.  The semicolon
; must be the first character of the line and therefore cannot be
; part of another line.
;
; Leading and trailing space characters are ignored.  To preserve
; the space characters you must surround the value with either
; single (') or double (") quote characters.  SQR will remove
; them when the entry is processed.
;
[Section_Name]
Entry = Value
     .
     .

[Another_Section_Name]
Entry = Value
     .
     .
```

Default Settings Section

This table describes the SQR default settings:

Entry	*Value*	*Description*
`ForceSpaceAfterComma=⇒ {TRUE|FALSE}`	TRUE | FALSE	The default setting is FALSE. DB2 only: Forces a space after every comma not in a literal value to support the DECIMAL=COMMA setting on the OS390.
`ShowDBWarnings=⇒ {TRUE|FALSE}`	TRUE | FALSE	The default setting is FALSE. DB2 only: If set to TRUE, SQR displays database warnings in the SQR output file.

Entry	*Value*	*Description*

`AllowDateAsChar=⇒ {TRUE|FALSE}`	TRUE | FALSE	The default setting is FALSE.
		By default, SQR produces an error when a dynamic column specification does not match the database definition of the column. That is, character equals character, date equals date, and numeric equals numeric.
		When this value is set to TRUE, SQR allows character to be equal to either character or date columns.
		When a date column is type cast to be a character, SQR creates the string according to the following rules:
		• For DATETIME columns, SQR uses the format specified by the SQR_DB_DATE_FORMAT setting. If this is not set, SQR uses the first database-dependent format listed in the table showing default database formats in the Edit section under the PRINT command.
		• For DATE columns, SQR uses the format specified by the SQR_DB_DATE_ONLY_FORMAT setting. If this is not set, SQR uses the format listed in the table showing default database formats in the Edit section under the PRINT command.
		• For TIME columns, SQR uses the format specified by the SQR_DB_TIME_ONLY_FORMAT setting. If this is not set, SQR uses the format listed in the table showing the TIME column formats in the Edit section under the PRINT command.
		In the following example, AllowDateAsChar=True. This setting allows $Col1 to be either date or text.
		`Begin-Select`
		`  [$Col1] &col1=Text`
		`  [$Col2] &col2=Date`
		`  [$Col3] &col3=Number`
		`  from MyTable`
		`End-Select`

Entry	*Value*	*Description*
OUTPUT-FILE-MODE	LONG | SHORT	Specifies the file name convention used for HTML output. SHORT specifies DOS style (8.3) and LONG specifies UNIX\Linux style (non 8.3). The default is LONG. This setting is ignored on 16-bit platforms. The DECLARE-TOC and -BURST commands force Output-File-Mode = LONG. The following list items represents the file formats for UNIX\Linux, DOS, and Windows. SQR and SQRT: {Program} is the name of the SQR/SQRT file without the extension.
OUTPUT-FILE-MODE (continued)	LONG | SHORT	(Continued) For Output-File-Mode = SHORT, SQR-generated file names are limited to a DOS 8.3 format. • Output file = {program}.lis for first, and {program}.lnn for multi-reports • SPF file = {program}.spf for first, and {program}.snn for multi-reports • PDF file = {program}.pdf for first; and {program}.pnn for multi-reports • HTM file = {program}.htm for "frame, and {program}.hbb for report bodies • GIF file={program}.gxx for all reports bb ranges from 00 to 99 and represents the report number. nn ranges from 01 to 99 and represents the report number. xx ranges from 00 to ZZ and represents the graphic number.

Entry	*Value*	*Description*
OUTPUT-FILE-MODE (continued)	LONG | SHORT	(Continued) For Output-File-Mode = LONG, SQR-generated file names are not constrained to a DOS 8.3 format. {output}={program} of first report and {program}_nn for multi-reports. • Output file = {output}..lis • SPF file = {output}..spf • PDF file = {output}..pdf • GIF file = {output}_zz..spf • HTM files = {output}..htm, {output}_bb..htm, {output}_frm..htm, {output}_toc./htm, {output}_nav.htm bb ranges from 01 to ZHJOZI and represents the bursted page group number. nn ranges from 01 to 99 and represents the report number. zz ranges from 00 to ZHJOZI and represent the graphic number.
OUTPUT-FILE-MODE (continued)	LONG | SHORT	(Continued) SQRP: {file name} is the name of the SPF file without the extension For Output-File-Mode = SHORT, SQR-generated file names are limited to a DOS 8.3 format. • Output file = {file name}.lis • GIF file = {file name}.gxx • PDF file = {file name}.pdf • HTM file = .htm and {file name}.h00 xx ranges from 00 to ZZ and represents the graphic number.

Entry	*Value*	*Description*
OUTPUT-FILE-MODE (continued)	LONG | SHORT	(Continued) For Output-File-Mode = LONG, SQR-generated file names are not limited to a DOS 8.3 format. • Output file = {file name}.lis • PDF file = {file name}.pdf • GIF file = {file name}_zz.spf • HTM files = {file name}.htm, {file name}_bb.htm, {file name}_frm.HTM, {file name}_toc.htm {file name}_nav.htm bb ranges from 01 to ZHJOZI and represents the bursted page group number. zz ranges from 00 to ZHJOZI and represents the graphic number.
LOCALE	Name of a locale defined in the sqr.ini file or the name SYSTEM.	Specifies the initial locale that SQR loads when the program starts. The value of SYSTEM is used to reference the default locale. See Chapter 2, "SQR Command Reference," ALTER-LOCALE, page 28.
DEFAULT-NUMERIC	INTEGER | FLOAT | DECIMAL[(p)] | V30	Specifies the default numeric type for variables. The command line flag -DNT overrides this setting. Similarly, the DECLARE-VARIABLE command overrides this setting. See Chapter 2, "SQR Command Reference," DECLARE-VARIABLE, page 108.
OutputFormFeedWithDashD=⇒ {TRUE|FALSE}	TRUE | FALSE	The default value is FALSE. When set to TRUE, the -Dnn command line flag outputs the Form-Feed character that denotes a page break.
OutputTwoDigitYearWarningMsg=⇒ {TRUE|FALSE}	TRUE | FALSE	The default value is TRUE. When set to TRUE, SQR generates a warning message (sent to the warning file) when a YY or RR date edit mask is encountered during a program run. This setting affects only SQR code that is processed.

Entry	Value	Description
UseY2kCenturyAlgorithm=⇒ {TRUE|FALSE}	TRUE | FALSE	The default value is FALSE. When set to TRUE, SQR treats the YY date edit mask as though it is an RR edit mask. See the RR edit mask.

Note. Use the setting V30 to process numbers in the same manner as in previous (before V4.0) releases. Specifically, all numeric variables and literals are declared as FLOAT, including integer literals.

Processing-Limits Section

Use the Processing-Limits section to define the sizes and limitations of some of the internal structures used by SQR; these definitions directly affect memory requirements. The entries are the same as those used in the file specified with the -MFILE command-line flag. If the -MFILE command-line flag is used, the Processing-Limits section of the file is not processed.

This table describes some of the internal structures used by SQR:

Entry	Default Value	Maximum Value	Entry Size	Description
BREAKS	100	64K-1	4	Number of BREAK arguments allowed for each EVALUATE or IF command.
DYNAMICARGS	70	32K-1	14	Maximum number of dynamic SQL arguments.
EXPRESSIONSP⇒ ⇒ ACE	8192	64K-1	1	Maximum length, in bytes, of temporary string storage used during LET operations.
FORWARDREFS	200	32K-1	8	Maximum number of column forward references.
LONGSPACE	32K-2	32K-2	1	Maximum buffer size to transfer text and image data in bytes.

Entry	*Default Value*	*Maximum Value*	*Entry Size*	*Description*
ONBREAKS	30	64K-1	8	Maximum number of ON-BREAK LEVEL=values per SET.
POSITIONS	1800	64K-1	14	Maximum number of placement parameters, for example, (10,5,30).
PROGLINEPARS	18000	64K-1	2	Maximum number of arguments for all program lines. This value is generally 3 or 4 times the value set for PROGLINES.
PROGLINES	5000	32K-1	8	Maximum number of program lines (SQR commands).
QUERIES	60	32K-1	60	Maximum number of BEGIN-SQL and BEGIN-SELECT paragraphs. This value is database dependent and can vary. This size is a close approximation.
QUERYARGS	240	64K-1	6	Maximum number of arguments (bind variables) for all SQL or SELECT statements. The number of arguments required is one more than the number used in your report file.
SQLSIZE	4000	64K-1	1	Maximum length of an SQL statement in characters.

Entry	*Default Value*	*Maximum Value*	*Entry Size*	*Description*
STRINGSPACE	15000	64K-1	1	Maximum size of string space for program line arguments, in bytes.
SUBVARS	100	32K-1	8	Maximum number of runtime substitution variables.
VARIABLES	1500	64K-1	18	Maximum number of variables *(string, float, integer, decimal),* literal values, and database columns. Add 4 to the entry size for Informix.
WHENS	70	64K-1	4	Maximum number of WHEN arguments allowed for each EVALUATE command.

The maximum value refers to the number of entries allowed, as shown in the previous table; however, limits are lower for 32-bit machines. In either case, SQR indicates the limit if you exceed it.

In addition to increasing the sizes, you may also decrease them to decrease the amount of memory used. Decreasing them might be advantageous, for example, for certain applications running in the PC environment, where memory is limited.

Environment Sections

The Environment sections { Common | Oracle | Informix | ODBC | DB2 | RDB | Sybase] } define the environment variables used by SQR. You can define an environment variable in multiple environment sections; however, a definition in a database-specific environment section takes precedence over an assignment in the Environment:Common section.

You can set these environment variables: *SQRDIR, SQRFLAGS,* and *DSQUERY.* In Microsoft Windows systems, *SQRDIR* is required and is automatically defined in the appropriate database-specific environment section during the SQR installation. The other environment variables are optional. *SQRFLAGS* specifies the default command-line flags for all invocations of SQR. *DSQUERY* (Sybase only) identifies the default Sybase server to use.

In Microsoft Windows systems only, the SQR Extension section defines DLLs containing new user functions (UFUNC) and user calls (UCALL). UFUNC and UCALL reside inside SQREXT.DLL and other DLLs, or in one or the other.

When SQRW.DLL and SQRWT.DLL are being loaded, they look for SQREXT.DLL in the same directory and for any DLLs specified in the SQR Extension section in sqr.ini, such as:

```
[SQR Extension]
c:\sqrexts\sqrext1.dll=
c:\sqrexts\sqrext2.dll=
c:\sqrexts\sqrext3.dll=
```

Any new extension DLLs containing new user functions must be listed in the SQR Extension section in sqr.ini.

See *PeopleTools 8.52: SQR for PeopleSoft Developers*, "Using Interoperability Features."

For Oracle on Windows, SQR uses dynamic binding of Oracle routines. When SQR attempts to access an Oracle database, it searches for the Oracle DLL as follows:

- The file described by the value of ORACLE_DLL entry in the Environment:Oracle section of the sqr.ini file.
- OCIW32.DLL (Oracle supplied).
- ORANT71.DLL (Oracle supplied).

This table describes SQRParseSQLEnable:

Entry	***Value***	***Description***
SQRParseSQLEnable={TRUE|FALSE}	TRUE | FALSE	The default setting is FALSE. If SQRParseSQLEnable=TRUE is added to the PSSQR.INI file (in the Common or Oracle stanza), SQR performs SQL parsing. **Note.** The function required to perform this parsing is not supported by Oracle 8.X, but it does appear to function. If a customer wants to assume the risk of using a feature that might not be fixed if a problem is found, then they can turn on this parameter. Oracle 9.X customers can use this feature without any problem, although it has a performance hit of between 2-10 percent.

Locale Section

This section specifies the default settings for the locale identified by locale-name (which can consist of letters from A to Z, numbers from 0 to 9, a hyphen, and an underscore). A number of locales are predefined in the pssqr.ini file. Depending on your application, you may have to alter the settings for these locales or add new locales. You can reference or alter a locale at runtime using the ALTER-LOCALE command. The following table describes the entries for a locale section.

Note. The SYSTEM locale is provided for your reference but is commented out. The settings for the SYSTEM locale, if set, are ignored. Use the ALTER-LOCALE command to change the SYSTEM locale settings at runtime.

See Chapter 6, "Avoiding Older SQR Commands," page 283.

Entry	*Description*
`NUMBER-EDIT-MASK`	Specifies the default numeric edit mask format when the keyword **NUMBER** accompanies the DISPLAY, MOVE, PRINT, or SHOW command.
`MONEY-EDIT-MASK`	Specifies the default numeric edit mask format when the keyword **MONEY** accompanies the DISPLAY, MOVE, PRINT, or SHOW command.
`DATE-EDIT-MASK`	Specifies the default date edit mask format when the keyword **DATE** accompanies the DISPLAY, MOVE, PRINT, or SHOW command, or the LET `datetostr()` or `strtodate()` functions.
`INPUT-DATE-EDIT-MASK`	Specifies the default date format to use with the INPUT command when TYPE=DATE is specified with the command or the input variable is a DATE variable.
`MONEY-SIGN`	Specifies the characters to replace the ($) edit character.
`MONEY-SIGN-LOCATION`	Specifies the location of the MONEY-SIGN character. Valid values are LEFT and RIGHT.
`THOUSAND-SEPARATOR`	Specifies the character to replace the comma (,) edit character.
`DECIMAL-SEPARATOR`	Specifies the character to replace the period (.) edit character.

Entry	*Description*
DATE-SEPARATOR	Specifies the character to replace the slash (/) character.
TIME-SEPARATOR	Specifies the character to replace the colon (:) character.
EDIT-OPTION-NA	Specifies the characters to replace the NA option.
EDIT-OPTION-AM	Specifies the characters to replace AM
EDIT-OPTION-PM	Specifies the characters to replace PM.
EDIT-OPTION-AD	Specifies the characters to replace AD.
EDIT-OPTION-BC	Specifies the characters to replace BC.
DAY-OF-WEEK-CASE	Specifies how the case for the DAY-OF-WEEK-FULL or DAY-OF-WEEK-SHORT entries are affected when used with the format codes DAY or DY. Valid values are UPPER, LOWER, EDIT, and NO-CHANGE. Use UPPER and LOWER to force the output to either all uppercase or lowercase, ignoring the case of the format code in the edit mask. Use EDIT to follow the case as specified with the format code in the edit mask. Use NO-CHANGE to ignore the case of the format code and output the day of week as explicitly listed in the DAY-OF-WEEK-FULL or DAY-OF-WEEK-SHORT entries.
DAY-OF-WEEK-FULL	Specifies the full names of the days of the week. SQR considers the first day of the week to be Sunday. You must specify all seven days.
DAY-OF-WEEK-SHORT	Specifies the abbreviated names for the days of the week. SQR considers the first day of the week to be Sunday. You must specify all seven abbreviations.

Entry	*Description*
MONTHS-CASE	Specifies how the case for the MONTHS-FULL or MONTHS-SHORT entries is affected when used with the format codes MONTH or MON. Valid values are UPPER, LOWER, EDIT, and NO-CHANGE. Use UPPER and LOWER to force the output to either all uppercase or lowercase, ignoring the case of the format code in the edit mask. Use EDIT to follow the case as specified with the format code in the edit mask. Use NO-CHANGE to ignore the case of the format code and output the month as explicitly listed in the MONTHS-FULL or MONTHS-SHORT entries.
MONTHS-FULL	Specifies the full names for the months of the year. SQR considers the first month of the year to be January. You must specify all 12 months.
MONTHS-SHORT	Specifies the abbreviated names for the months of the year. SQR considers the first month of the year to be January. You must specify all 12 abbreviations.

Fonts Section

The Fonts section lists the fonts available to SQR when printing on Microsoft Windows printer devices (using the -PRINTER:WP command-line flag), viewing SPF output using SPF Viewer on Microsoft Windows (SQRWV), and creating style sheet for Enhanced HTML output. This section does not apply to PostScript or HP LaserJet printer types.

This section provides an overview of fonts available to SQR and discusses how to:

- Add font entries.
- Specify character sets in Windows.

See Chapter 2, "SQR Command Reference," DECLARE-PRINTER, page 93 and Chapter 6, "Avoiding Older SQR Commands," page 283.

Adding Font Entries

The Fonts section contains a number of predefined font entries. You can add entries by using the font numbers 900 through 999. Each entry consists of a font name, a font style (fixed or proportional), and a bold indicator, all of which are associated with a font number, for example:

```
4=Arial,proportional
  or
300=Courier New,fixed,bold
```


Note. A proportional font style is assumed if the second parameter starts with a *P*. Bold is assumed if a third parameter is supplied.

Using the font number, commands such as ALTER-PRINTER and DECLARE-PRINTER can reference a particular font style.

Specifying Character Sets in Windows

In Microsoft Windows, you can use the CharacterSet entry either to determine the Microsoft Windows default character set or to specify a character set. The CharacterSet entry enables you to print any standard character set to a Windows printer (-PRINTER:WP) or to view an SPF file that displays the appropriate character set.

The syntax is:

```
CharacterSet=DEFAULT|AUTO|character_set
```

The arguments are:

- DEFAULT reflects current SQR functionality.
- AUTO automatically determines the default character set of the Microsoft Windows installation and uses the default set when generating reports.

  CharacterSet specifies one of these keywords: ANSI, ARABIC, BALTIC, CHINESEBIG5, EASTEUROPE, GB2312, GREEK, HANGUL, HEBREW, JOHAB, MAC, OEM, RUSSIAN, SHIFTJIS, SYMBOL, THAI, TURKISH, VIETNAMESE.

HTML-Images Section

The HTML-Images section defines the parameters that SQR uses when generating HTML report output files. This table describes the parameters SQR uses when generating HTML report output files:

Entry	*Value*	*DefaultValue*	*Description*
FIRST-PAGE	HEIGHT, WIDTH, NAME	60,60,firstpg.gif	Specifies the NAME of the graphic image file that accesses the first page of the report. The HEIGHT and WIDTH values are specified in pixels.
PREV-PAGE	HEIGHT, WIDTH, NAME	60,60,prevpg.gif	Specifies the NAME of the graphic image file that accesses the previous page of the report. The HEIGHT and WIDTH values are specified in pixels.

Entry	*Value*	*DefaultValue*	*Description*
NEXT-PAGE	HEIGHT, WIDTH, NAME	60,60,nextpg.gif	Specifies the NAME of the graphic image file that accesses the next page of the report. The HEIGHT and WIDTH values are specified in pixels.
LAST-PAGE	HEIGHT, WIDTH, NAME	60,60,lastpg.gif	Specifies the NAME of the graphic image file that accesses the last page of the report. The HEIGHT and WIDTH values are specified in pixels.
WALLPAPER	NAME		Specifies the NAME of the graphic image file used as the background image for the report.
Navbar Background	NAME		Specifies the background image of the navigation bar.

Note. SQR does not validate any of the graphic image file names provided. The user is responsible for ensuring that the graphic image files are in a location that the browser can access.

PDF Fonts Section

The PDF Fonts section lists the available fonts for SQR when printing using the -PRINTER:PD command-line flag. Fonts specified are case sensitive.

Unlike the Fonts section, the PDF Fonts section uses a list of fonts mapped to a single font number. You can specify up to 10 fonts for a single font number. Depending on the character to print, SQR determines the font to use from the list. The following is the syntax of this font list:

```
Font Number=Font1, Font2, Font3 ...
```

The list is ordered by priority from the left. If Font1 has a glyph (an image of a character) for the character you want to print, then used; but if Font1 does not have a glyph, then Font2 is checked and used if it has a glyph for that character.

Font Number is a decimal number that specifies fonts in the SQR program using statements like DEFINE-PRINTER or ALTER-PRINTER.

Font numbers that are multiples of 100 (300, 3200, and so on) are recognized as a bold version of the base font. For example, font 300 is the bold version of Font3.

Font numbers 30 to 39 are recognized as italic fonts, and multiples of 30 to 39 (3000, 3100, and so on) are bold italic versions of the font. If you assign nonbold fonts for a font number recognized as bold, such as 600, the text is printed in bold.

Font1, Font2, and Font3 represent the font name. You can use Adobe Reader core fonts, Adobe Reader Asian Font Pack fonts, or TrueType fonts.

Adobe Reader core fonts are the fonts that Adobe Reader natively supports. The core fonts are:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats

Adobe Reader Asian Font Pack fonts are the font packages Adobe provides as an add-on to Adobe Reader for the purpose of viewing Asian text. You can download these fonts from the Adobe website or, if you use Adobe Reader 6, the Asian Font Pack automatically downloads when you open a PDF file that contains a font from the Asian Font Pack. Alternatively, you can use the Adobe Reader localized to any of the Japanese, Chinese, or Korean languages. These localized versions come with the same fonts that are included in the Asian Font Pack. These Asian Font Pack fonts are used with SQR:

- HeiseiKakuGo-W5 (Japanese)
- HeiseiMin-W3 (Japanese)
- MHei-Medium (Traditional Chinese)
- MSung-Light (Traditional Chinese)
- STSong-Light (Simplified Chinese)
- HYGoThic-Medium (Korean)
- HYSMyeongJo-Medium (Korean)

In addition, you can use TrueType/OpenType fonts that exist on the machine where you run SQR. If you use TrueType fonts for PDF output, they will be always embedded as subset of the original font. To use TrueType fonts for PDF output, you need to specify the mapping between the font name and the font file path in the [TrueType Fonts] section, and then include the font name in the font list under the [PDF Fonts] section. More information about TrueType fonts is included in a later section in this chapter.

See Chapter 7, "Using the PSSQR.INI File and the PSSQR Command Line," TrueType Font Section, page 318.

PDF Fonts: Exclusion Ranges Section

The PDF Fonts: Exclusion Ranges section defines character ranges you want to exclude from the range a font covers. Without an exclusion range, SQR uses all of the character ranges a font covers, and then checks the font listed in the next priority only when it does not cover a character. By specifying an exclusion range, you can use a font in a lower priority for a specific character or a range of characters. This mechanism allows users to control which font to use down to a single character level.

The syntax in this section is as follows:

```
Font Number=Font1 Exclusion Range, Font2 Exclusion Range, ...
```

Exclusion ranges apply to the list of fonts defined in the PDF Fonts section. For example, if you have a PDF Fonts section like the following:

```
3=Courier,Cumberland,HeiseiKakuGo-W5,...
```

In addition, if you define the PDF Fonts: Exclusion Ranges section like this

```
 3=0x20AC,0x3070-0x30FF,,
```

then you are defining that Euro currency symbol (0x20AC in Unicode) is rendered using the Cumberland font, even though Courier has it. In addition, Greek characters (from 0x3070 to 0x30FF in Unicode) are rendered using HeiseiKakuGo-W5 font, even though Cumberland has them.

Each Exclusion range is specified using Unicode (UCS-2) codepoint, in decimal or in hexadecimal, in the following ways:

0x20AC (single character)

160-255 (range)

0x20AC|0x00A0-0x00FF (multiple characters and ranges)

Hexadecimal numbers need to be preceded by 0x; otherwise, they are recognized as decimal.

TrueType Font Section

The TrueType Font section defines the mapping of font name, which is an alias SQR uses internally to look up a font and physical font file name. You need to specify the font in the full path, unless you have placed fonts in a directory specified in the Font Path parameter or in a Windows font folder if you are running SQR on Windows.

The syntax is:

```
font name=fontfile path
```

For example, if you have the font courier.ttf in the c:\user\fonts directory, you set the following in this section:

```
CourierNew=c:\user\fonts\courier.ttf
```

Font name can be any string that is convenient for you to identify the font, and it does not need to correspond to the internal name of the font. You should not use the same font name that is used for Adobe core fonts or for Asian Font Pack fonts. If you set the same name for any of Adobe core fonts or Asian Font Pack fonts, then the TrueType font is used.

Note. TrueType font embedding is now supported by SQR running on z/OS. However, the Font Path parameter under the TrueType Font section is not supported for z/OS. If you are on z/OS, you need to specify the full path to each entry.

If you have a TrueType Collection (TTC) file, you will also need to specify the font number to access the specific font included in the collection. For example, if MS P Mincho is included in the TrueType Collection file msmincho.ttc, then you will need to specify:

```
MSPMincho=c:\winnt\fonts\msmincho.ttc,1
```

The font number within the TrueType collection starts at 0. msmincho.ttc contains MS Mincho and MS P Mincho in this order; thus the number 0 represents MS Mincho and 1 represents MS P Mincho. If you do not specify a font number, SQR uses font with font number 0 from the TrueType Collection file.

Note. You can use only Microsoft type of Unicode-based TrueType/OpenType fonts with SQR. SQR requires TrueType/OpenType font to have CMAP table with Platform ID 3 (Microsoft), Encoding ID 0 (Symbol), 4 (UCS-2), or 10 (UCS-4) and table format 4 or 12. SQR does not support OpenType fonts with CFF (Postscript) outline. You can use OpenType fonts with TrueType outlines, but SQR does not make use of advanced layout features provided with OpenType font.

Enhanced-HTML Section

The Enhanced-HTML section is used to define various default actions that SQR takes when generating HTML output using the -EH command-line flag. This table describes the default actions that SQR takes when generating HTML output using the -EH command-line flag:

Entry	*Value*	*Description*
Browser={ALL|IE|NETS CAPE}	ALL|IE|NETSCAPE	When set to ALL, the generated HTML automatically determines which browser is being used and invokes the proper browser-specific file. When set to IE, the generated HTML is designed for Internet Explorer. The default action is to generate HTML suitable for all browsers. **Note.** NETSCAPE is not supported in PeopleTools 8.52
Language={English|French| German| Portuguese|Spanish|Japanese| Simplified Chinese|Korean}	English|French| German| Portuguese|Spanish|Japanese| Simplified Chinese|Korean	Sets the language used for the HTML navigation bar. The default setting is English.
FullHTML={TRUE | FALSE}	TRUE | FALSE	When set to TRUE, HTML 3.2 is generated. When set to FALSE, HTML 3.0 is generated. The default setting is FALSE.

Colors Section

The Colors section defines the default colors that you can use in your SQRs. Enter the default colors in this format:

```
[Colors]
color_name = ({rgb])
color_name = ({rgb])
...
color_name = ({rgb])
```

The default colors implicitly installed are:

- black=(0,0,0)
- white=(255,255,255)
- gray=(128,128,128)
- silver=(192,192,192)
- red=(255,0,0)
- green=(0,255,0)

- blue=(0,0,255)
- yellow=(255,255,0)
- purple=(128,0,128)
- olive=(128,128,0)
- navy=(0,0,128)
- aqua=(0,255,255)
- lime=(0,128,0)
- maroon=(128,0,0)
- teal=(0,128,128)
- fuchsia=(255,0,255)

Using PSSQR.EXE Command-Line Options

The PeopleSoft system provides a shell called pssqr.exe that extends SQR to handle the submission of SQR programs under Microsoft Windows and UNIX/Linux operating systems. Pssqr.exe implements a process that ensures that output is sent to the appropriate destination in a way that is consistent across platforms.

Note. PeopleSoft does not support running pssqr.exe directly from a DOS or Unix/Linux command line. Pssqr.exe is a wrapper program used by PeopleSoft Process Scheduler to run SQR reports; it is not designed to run manually outside of Process Scheduler.

Pssqr.exe does not run on OS/390, however, all output formats are supported on all platforms.

Pssqr.exe provides the following features:

- Expanded output formats: SQR Viewer, HTML, PDF, CSV (Spreadsheet Standard), HP, Postscript, Line Printer.
- Expanded printer format: Microsoft Windows Default Printer (Win32 only), HP, Postscript, Line Printer.
- Enhanced delivery of reports to printers: PSSQR sends reports to the printer instead of SQR, which resolves issues that can be encountered on non-Microsoft networks.
- File output and logs with unique names. If the process instance is sent, the file names will be <SQR Program>_<Instance>.xxx.

  If not, then the log names will be <SQR Program>_<timestamp>.xxx.
- Common command line interface for both Microsoft Windows and UNIX/Linux.
- Capability to read Configuration Manager settings to determine the flags (SQRFLAGS) to use.
- Support for multiple report output.

  Default values come from Configuration Manager if run from the command line.

This table describes the command-line parameters:

Command-Line Parameter	***Description***
-CT	Database type. Valid values are ORACLE, DB2, DB2390, INFORMIX, MICROSFT, and SYBASE.
-CS	Server name.
-COWN	Database owner. This parameter applies to OS/390 only.
-CD	Database name.
-CA	Access ID.
-CAP	Access password.
-RP	Program name.
-I	Process instance.
-R	Run control ID.
-CO	User ID.
-OT	Output type: • 2 (File) • 3 (Printer) • 4 (Window)

Command-Line Parameter	*Description*
-OF	Output format. Column headings refer to the -OT parameter. • 2 (Adobe Acrobat) • 3 (Comma delimited) • 4 (HP format) • 5 (HTML documents) • 6 (Line printer) • 10 (Postscript) • 13 (SQRiBE portable) • 14 (Text files) • 15 (Windows default printer)
-OP	Output destination.
-MR	Multiple report output. Values: # (maximum 99)
-AF	Additional flags.
-AP	Additional information.
-TR	Enable trace.
-DL	Display log file.
-FILE	Enter the name of a parameter file containing the parameters you want to pass to PSSQR. The system deletes the file immediately after use.
-LG	Language Code. Takes a three-letter language code used across PeopleSoft systems. This flag is used to determine which language version of the PSSQR.INI file to use. If you specify LG JPN, then pssqr.exe picks up the Japanese configuration file (PSSQRJPN.INI) instead of PSSQR.INI. This parameter takes effect only when the %LANGUAGE_CD% variable is used for specifying the PSSQR.INI file in PSSQRFLAGS entry of the Process Scheduler configuration or Configuration Manager.

Command-Line Parameter	*Description*
-LPFLAGS "<flags>"	UNIX\Linux only: Overrides the flags passed to the lp command.
-DEBUGLP ON	Displays command used to print the SQR report.

Note. Avoid using a hyphen (-) or @ sign in the PSSQR.EXE command line, for instance, as part of the run control ID. The hyphen (-) and @ sign characters are SQR-reserved characters.

See Also

PeopleTools 8.52: SQR for PeopleSoft Developers, "Using the SQR Command Line," Using Reserved Characters

Appendix A

Understanding SQR Messages

This chapter discusses all the messages produced by SQR for the PeopleSoft application:

- Unnumbered messages
- Numbered messages

Note. An apostrophe and two digits (`nn) appear as replacement markers in the messages. Descriptions of these replacement markers are listed in the message itself. The messages contain the proper value when they appear on the screen.

Unnumbered Messages

This table describes all unnumbered SQR messages:

Message	*Description*
Out of memory.	This error occurs when a call to the C routine 'malloc()' fails. (32-bit machines) Use the -Mfile command-line flag to reduce some of the different memory requirements. Remove unneeded TSRs. (UNIX\Linux) Increase the size of the system swap file.
No cursors defined.	This message is issued from the -S command-line flag. The SQR program did not contain any commands that required a database cursor.
Not processed due to report errors.	This message is issued from the -S command-line flag. SQR cannot provide information about the cursor due to errors in the program.
Enter `01`02	This message prompts the user to type the value to be assigned to the specified variable. `01 = First character of the variable name. `02 = Rest of the variable name.

Message	*Description*
NOPROMPT used - Enter value below	(Microsoft Windows) This message appears when an INPUT command is defined with the NOPROMPT argument.
Enter `01	This message prompts the user to enter the value to be assigned to the specified substitution variable. `01 - Name of the substitution variable
Enter this run's parameters:	This message prompts the user to enter the values for the parameters defined in the program.
Error on line `01: `02	SQR detected an error while processing the report file. Correct the error and rerun. `01 - Source line number. `02 - Source line.
Error in include file "`01" on line `02: `03	SQR detected an error while processing the report file. Correct the error and rerun. `01 - Name of the include file. `02 - Source line number. `03 - Source line.
Warning on line `01: `02	SQR detected a nonfatal error while processing the report file. `01 - Source line number. `02 - Source line.
Warning in include file "`01" on line `02: `03	SQR detected a nonfatal error while processing the report file. `01 - Name of the include file. `02 - Source line number. `03 - Source line.
Type RETURN for more, C to continue w/o display, X to exit run:	This informational message is used in conjunction with the -D command-line flag.
Error at: `01 Loading Oracle DLL Failed!!!	(Oracle) This title for the dialog box informs the user that SQR could not load the Oracle DLL.

Message	*Description*
Errors were found in the program file.	This error message tells the user that there were errors in the program. The user can correct the errors and rerun.
Errors were found during the program run.	This error message tells the user to there were errors running the program. The user can correct the errors and rerun.
`01: End of Run.	This is an informational message. `01 - Image name (for example, SQR)
Enter report name:	This message prompts the user to enter the name of the report (.sqr or .sqt) to run.
Enter database name:	This message prompts the user to enter the name of the database.
Enter database[/username]: Enter Username:	This message prompts the user to enter the user name to log onto the database.
Enter Password:	This message prompts the user to enter the password. For security reasons, the password is not echoed.
Customer ID:	This is a text message.
Press Enter to close...	This is a text message.
`01: Program Aborting.	This is an informational message. `01 - Image name (for example, SQR)
*** Internal Coding Error ***	This is an informational message.
SQL DataServer Message	(Microsoft Windows) This message is a title for the error message dialog box.
Operating-System error	(Microsoft Windows) This message is a title for the error message dialog box.
DB-Library error	(Microsoft Windows) This message is a title for the error message dialog box.

Message	*Description*
`01 is running. Click the Cancel button to interrupt it.	(Microsoft Windows) This message is the body of the -C cancel dialog box. The user can quit the program run by clicking the Cancel button.
Table of Contents	This message is the text for HTML driver.
Previous	This message is the text for HTML driver.
Next	This message is the text for HTML driver.
First Page	This message is the text for HTML driver.
Last Page	This message is the text for HTML driver.
PAGE	This message is the text for HTML driver.

Numbered Messages

This tabledescribes numbered messages:

Error Number	*Error Message*	*Suggestion/Interpretation*
000001	Error while opening the message file: '`01' (`02): `03	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Name of the error message file. `02 = System error code. `03 = System error message.
000002	Error while reading the message file. (`01): `02	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Name of the error message file. `02 = System error code. `03 = System error message.

Error Number	*Error Message*	*Suggestion/Interpretation*
000003	Error while closing the message file. (`01): `02	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Name of the error message file. `02 = System error code. `03 = System error message.
000004	Error while seeking the message file. (`01): `02	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Name of the error message file. `02 = System error code. `03 = System error message.
000005	Corrupt message file: Invalid header information.	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support.
000006	Corrupt message file: Invalid count (Got `01, Should be `02).	The header contains an invalid entry count. 1. Make sure SQRDIR points to the correct directory. 2. Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = The value read from the header. `02 = The correct value.

Error Number	*Error Message*	*Suggestion/Interpretation*
000007	Cannot handle message file version `01.	This message occurs when the release of SQR does not support the header version. 1. Make sure SQRDIR points to the correct directory. 2. Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Unsupported version read from the header.
000010	Invalid SEMCode encountered: `01.	An invalid code was passed to the error message handler. Try reloading the files from the release media. If the error persists, contact technical support. `01 = Invalid code.
000011	Unknown conversion type (`01) for code `02.	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Invalid type. `02 = Internal error code.
000012	Message `01 must be either Preload or BuiltIn.	The type error code is not correct. Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Error code.
000013	Cannot point to message `01.	The error handler cannot position to the desired error code. Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Error code.
000014	The required environment variable `01 has not been defined.	Define the named environment variable and restart SQR. `01 = Environment variable name.

Error Number	*Error Message*	*Suggestion/Interpretation*
000015	The Meta ESC characters do not match (Got '`01', Should be '`02').	The meta escape character defined in the header does not match what the error message handler expects. Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = What was found in the header. `02 = What was expected to be found
000016	`01() called to process (`02) and the message file is not open.	The specified error routine was called but the error message file was not open. Try reloading the files from the release media. If the error persists, contact technical support. `01 = Name of the routine `02 = Error code
000017	Message `01 must be ReportParameters or CopyrightNotice.	Try reloading the sqrerr.dat file from the release media. If the error persists, contact technical support. `01 = Error code
000018	Allocation header does not point to a valid heap.	(Microsoft Windows) This message is the result of a memory overwrite. Record the steps leading up to the error and contact technical support.
000019	Allocation header has an invalid size.	(Microsoft Windows) This message is the result of a memory overwrite. Record the steps leading up to the error and contact technical support.
000020	GLOBAL header has an invalid size.	(Microsoft Windows) This message is the result of a memory overwrite. Record the steps leading up to the error, and contact technical support.
000021	Cannot free GLOBAL allocation.	(Microsoft Windows) This message is the result of a memory overwrite. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000028	Cannot access the initialization file: `01 (`02): `03	The initialization file specified by the -ZIF command line flag cannot be accessed. `01 = Name of the file. `02 = System error code. `03 = System error message.
000202	DPUT: Bad field number.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000203	DARRAY: Unknown command number.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000204	`01: Cannot find `02 command.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the routine. `02 = Name of the command.
000205	DDO: DO arguments do not match procedure's.	<obsolete>
000206	SDO: Bad params for DO command.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000207	SDO: Bad params for BEGIN-PROCEDURE command.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000208	SGOTO: Bad command numbers.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000209	SGOTO: Bad goto function parameters.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000210	SGOTO: Could not find beginning of section or paragraph.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000211	SGOTO: Bad label: from parameters.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000212	COMPAR: Unknown relational (numeric) operator.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000213	COMPAR: Unknown relational (string) operator.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000214	DONBRK: Unknown case for putlin.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000215	`01: Bad length case for numeric `02.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the routine. `02 = name of the variable.
000216	GARRAY: Unknown command number.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000217	GCMDS: No Gfunc found.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000218	GDOC: Unknown document type.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000219	GLET: Bad operator.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000220	GLET: Stack incorrect for expression - arg `01.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Number of the argument.
000221	GLET: Unknown operator type.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000222	GLET: Unknown operator in expression.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000223	GPARS: Column not SCOL, TCOL or NCOL type.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000224	GPARS: Bad parameter format: `01 =`02=	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string. `02 = Bad format field found.
000225	GPARS: No end of required word in parfmt: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string.
000226	GPARS: Bad parfmt entry: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string.
000227	GPARS: Bad parameter string.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000228	GPARS: Repeat count bad: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string.
000229	GPARS: Only a,b,8,9 allowed for repeats: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string.
000230	GPARS: Missing required x: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal command format string.
000231	GPARS: Bad type in 'ckvrpr()'.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000232	GPROC: No Gfunc found.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000233	GRDWRT: Unknown command number.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000234	GSHOW: Unknown SHOW option.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000235	PGMPARS: 'addvar()' passed maxlen but not column.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000238	PGMPARS: '`01' passed invalid parameter number: `02.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Routine name. `02 = Invalid parameter number.
000239	PGMPARS: 'fxclrf()' encountered bad column reference type: `01.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal variable type code.
000240	PLCMNT: 'getplc()' passed invalid element number: `01.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Invalid element number.
000241	RDPGM: Command array size exceeded (change COMDMAX to at least `01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Maximum internal command number supported.
000242	RDPGM: Bad match adding internal variable: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Internal variable name.

Error Number	*Error Message*	*Suggestion/Interpretation*
000243	RDPGM: No cmdget function found for BEGIN_S.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000244	Function `01 not included in run-time package.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the SQR routine.
000245	SETSQL: Could not find variable ''`01', in Run Time.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Variable name.
000248	SIFWHL: Command number incorrect.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000249	SPINIT: Bad parameters.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000251	DBFFIX: DBDATLEN returned out of range status.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000252	DPRPST: Error converting Sybase type for EXECUTE.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
000254	SETSQL: Could not find variable entry in list.	(Oracle) This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000255	DBDESC: SQLD not = number of select columns.	(DB2, Informix) This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
000256	DBFETCH: Unknown variable dbtype encountered: `01 (`02)	(DB2, Informix) This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Variable name. `02 = Unknown database type.
000257	WRITE_SPF: Unknown code encountered: `01	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Unknown SPF code.
000258	`01: Cannot find LOAD-LOOKUP table: `02	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the routine. `02 = Name of the table.
000259	PGMPARS: ''`01' called with wrong variable '`02'	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the routine. `02 = Name of the variable.

Error Number	*Error Message*	*Suggestion/Interpretation*
000260	SQTMGT: Could not find 'vars' entry with 'nvars' index of `01'.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Index into nvars table.
000261	MODIFYVAR: Attempt to change variable which is not xVAR (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = name of the variable.
000262	MODIFYVAR: Incompatible variable types (`01) and (`02).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Variable type (from). `02 = Variable type (to).
001100	Out of query arguments; use -Mfile to increase QUERYARGS.	This is the total number of variable references ($Var, #Var, &Col) allowed in the context of a BEGIN-SQL or BEGIN- SELECT command. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
001201	Cannot open the argument file: '`01'. (`02): `03	Depends on the system error message. `01 = Name of the file. `02 = System error code. `03 = System error message.
001202	Cannot close the argument file. (`01): `02	Depends on the system error message. `01 = System error code. `02 = System error message.

Error Number	*Error Message*	*Suggestion/Interpretation*
001203	Cannot open the - Mfile: '`01'. (`02): `03	Depends on the system error message. `01 = Name of the file. `02 = System error code. `03 = System error message.
001204	Minimum value for '`01' in the -Mfile is `02.	Correct the -Mfile entry. `01 = Keyword in question. `02 = Minimum value allowed.
001205	Maximum value for '`01' in the -Mfile is `02.	Correct the -Mfile entry. `01 = Keyword in question. `02 = Maximum value allowed.
001206	Invalid -Mfile entry: '`01'.	Correct the -Mfile entry. `01 = The line from the -Mfile.
001207	Cannot close the - Mfile. (`01): `02	Depends on the system error message. `01 = System error code. `02 = System error message.
001209	The minimum value for '`01' (`02) is `03.	Value out of range. `01 = Entry name. `02 = Specified value. `03 = Minimum value.
001210	The maximum value for '`01' (`02) is `03.	Value out of range. `01 = Entry name. `02 = Specified value. `03 = Maximum value.
001211	The value for '`01' (`02) is not an integer number.	Value must be a integer value. `01 = Entry name. `02 = Specified value.

Error Number	*Error Message*	*Suggestion/Interpretation*
001300	Bind list does not match query (do not use '@__p' string).	SQR reserves the variable names that start with "@__p" for internal use. Edit the source code and use different variable names.
001301	Forward references not permitted in select list bind variables.	Within the body of BEGIN-SQL paragraphs, forward references to &column names are not permitted. Move the BEGIN-SQL paragraph after the &column definition.
001302	SQL buffer too small; use -Mfile to increase SQLSIZE.	The SQL statement exceeds the size of the internal SQL buffer. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
001303	Error in SQL (perhaps missing &name after expression):	The database server has determined that the SQL statement is in error. The actual error text from the server follows this message. Correct the SQL statement.
001304	Check SELECT columns, expressions and 'where' clause for syntax.	The database server has determined that the SQL statement is in error. The actual error text from the server follows this message. Correct the SQL statement.
001305	CMPSQL: Unknown data type in database: `01.	Contact technical support with the version of the database you are connected to. `01 = Datatype in question.
001306	Bind value too large (IMAGE, TEXT not allowed).	IMAGE and TEXT data types cannot be used as bind variables. Modify your SQL statement to use other columns to perform the same selection logic.
001307	CMPSQL: DBDEFN failed.	(Oracle, ODBC, Informix) This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
001308	`01: Could not bind column `02.	(Oracle, ODBC, Informix) This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Name of the SQR routine. `02 = Name of the column.
001309	The type for '&`01' (`02) does not match the type from the database (`03).	Correct the source code. `01 = Name of the column/expression pseudonym. `02 = User specified type. `03 = Database type.
001400	Only numerics allowed for arithmetic.	Only #numeric variables, &columns, and literals are permitted in the arithmetic commands. Correct the source code.
001401	Optional qualifier is ROUND=n (0-`01).	Correct the syntax. `01 = Maximum value for ROUND.
001402	Optional qualifiers for DIVIDE are ON- ERROR={HIGH|ZER O} and ROUND=n.	Correct the syntax.
001403	Attempting division by zero.	Use the ON-ERROR = HIGH | ZERO option to prevent this error from halting the program.
001404	Bad number of digits to ROUND or TRUNC (0-15).	Correct the syntax.
001405	WARNING: The ROUND or TRUNC qualifier is greater than the number's precision.	Correct the syntax.
001500	Array element out of range (`01) for array '`02' on line `03.	Correct the source logic. `01 = Element number passed. `02 = Name of the array. `03 = Program line number.

Error Number	*Error Message*	*Suggestion/Interpretation*
001501	Field element out of range (`01) for array '`02', field '`03', on line `04.	Correct the source logic. `01 = Element number passed. `02 = Name of the array. `03 = Name of the field. `04 = Program line number.
001502	WARNING: Attempting division by zero on line `01. Array field '`02' unchanged. Run continuing...	The ARRAY-DIVIDE command has attempted division by zero. The division has been ignored; the result field is unchanged. Add logic to account for this possibility. `01 = Program line number. `02 = Name of field.
001601	'FILL' not appropriate for numeric data.	The FILL argument to the PRINT command may be used only for text fields. Move the #numeric variable to a $string variable, and then print the string variable.
001700	Report '`01': Columns must be between 1 and the page width (`02).	The specified value is wider than the width of the page. Correct the source line. `01 = Name of the current report. `02 = Page width.
001702	Report '`01': GOTO- TOP=`02 must be between 0 and the page depth (`03).	The value specified on the GOTO-TOP argument of the NEXT-COLUMN command was either less than 1 or greater than the page depth. Correct the source line. `01 = Name of the current report. `02 = Goto-Top value. `03 = Page width.
001703	Report '`01': ERASE- PAGE=`02 must be between 0 and the page depth (`03).	The line number specified on the ERASE- PAGE argument of the NEXT-COLUMN command is greater than the page depth. Correct the source line. `01 = Name of the current report. `02 = Erase-Page value. `03 = Page width.

Error Number	*Error Message*	*Suggestion/Interpretation*
001704	Report '`01': The NEXT-COLUMN command is not legal in the `02 section with the qualifier AT-END=NEWPAGE.	Correct the source line. `01 = Name of the current report `02 = Name of the section
001705	Report '`01': Column number `02 is not defined.	The column number specified with the USE- COLUMN command is greater than the highest column defined in the COLUMNS command. Correct the source line. `01 = Name of the current report `02 = Column number
001800	Format for CONNECT: username/password [ON-ERROR=procedure[(arg1[,argi]...)]]	Correct the syntax.
001801	Cannot use CONNECT while SQL statements are active.	Correct the program logic to ensure that all BEGIN-SELECT paragraphs have completed before executing the CONNECT command.
001802	Logoff failed prior to CONNECT.	The database server returned an error while trying to log off from the database. SQR ends the program run since it cannot continue.
001803	CONNECT failed. Perhaps user name/password incorrect.	The specified connectivity information is incorrect or there might have been a network failure. Use the ON-ERROR flag to trap any errors during the program run; otherwise SQR ends the program run.
001804	Sybase extensions SET and SETUSER not permitted in SQR.	Remove SET and SETUSER from the source.
001805	USE allowed once in SETUP section only, not in BEGIN-SQL. Elsewhere, specify db.[user].table...	Correct the source.

Error Number	*Error Message*	*Suggestion/Interpretation*
001806	Out of query space. Use -Mfile to increase QUERIES.	The number of SQL statements has been exceeded. Use the -Mfile flag on the command line specify a file that contains an entry that increases a greater value than is currently defined.
001807	The requested database connection (`01) is already active.	The -Cnn value specified is being used by another BEGIN-SELECT paragraph that is currently selecting data. Use another connection number. `01 = Connection number
001808	Cannot find inactive database cursor. Program too large.	Too many BEGIN-SELECT and BEGIN-SQL paragraphs are active at the same time. Reduce the complexity of the program.
001809	Database commit failed.	(Oracle, DB2, ODBC) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator.
001810	Database rollback failed.	(Oracle, DB2, ODBC) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator.
001811	Cannot open database cursor.	(Oracle, ODBC) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator.
001901	Variable for date-time must begin with '&'.	Correct the syntax.
001913	Format code must be SYYYY when specifying signed year.	Correct the edit mask.
001914	Bad input data (`01) for edit mask: '`02'.	Correct the input. `01 = Data being converted `02 = Edit mask

Error Number	*Error Message*	*Suggestion/Interpretation*
001915	Year cannot be zero.	Correct the date.
001916	Year must be between -4713 and 9999 inclusive.	Correct the date.
001917	Ambiguous date-time.	Correct the date.
001918	'`01' is not a valid date part.	Correct the date part. `01 = Date part.
001919	Invalid day of week.	Correct the date.
001920	Format code cannot appear in date input format: '`01'.	Correct the edit mask. `01 = Improper format characters.
001921	Bad date mask starting at: '`01'.	Correct the edit mask. `01 = Improper format characters.
001922	Seconds past midnight must be between 0 and 86399.	Correct the date.
001923	Seconds must be between 0 and 59.	Correct the date.
001924	Minutes must be between 0 and 59.	Correct the date.
001925	Month must be between 1 and 12.	Correct the date.
001926	Day must be between 1 and `01.	Correct the date.
001927	Hour must be between 1 and 12.	Correct the date.
001928	Hour must be between 0 to 23.	Correct the date.
001929	HH24 precludes the use of meridian indicator.	Correct the edit mask.
001930	HH12 requires meridian indicator.	Correct the edit mask.

Error Number	*Error Message*	*Suggestion/Interpretation*
001931	Day of year must be between 1 and 365 (366 for leap year).	Correct the date.
001932	Date string too long.	Correct the date.
001933	The month (`01) is not valid for the current locale or database.	Correct the date. `01 = Name of the month.
001934	The format mask must be a literal when the date-time is not loaded into a variable.	Correct the format mask. The format mask must be a literal when the date-time is not loaded into a variable.
001935	Date-time format too long.	Correct the format mask.
001936	Bad date-time format.	Correct the format mask.
001937	Bad SQL for default date-time. (Table DUAL required for syntax.)	(Oracle) The format mask needs to be corrected or there is a problem with the database server.
001937	Bad SQL for default date-time. (Table DUAL required for syntax.)	(DB2) The format mask needs to be corrected or there is a problem with the database server.
001938	Cannot recompile sql.	A fatal error relating to the SQL statement used to retrieve the date-time was encountered. Record the steps leading up to the error and contact your system administrator.
001939	Problem executing cursor.	A fatal error relating to the SQL statement used to retrieve the date-time was encountered. Record the steps leading up to the error and contact your system administrator.
001940	Error fetching row.	A fatal error relating to the SQL statement used to retrieve the date-time was encountered. Record the steps leading up to the error and contact your system administrator.

Error Number	*Error Message*	*Suggestion/Interpretation*
001941	Cannot redefine variable addresses.	A fatal error relating to the SQL statement used to retrieve the date-time was encountered. Record the steps leading up to the error and contact your system administrator.
001942	The date '`01' is not in the format SYYYYMMDD[HH24[MI[SS[NNNNNN]]]].	When specifying an SQR date, at a minimum, the date must be specified. The time is optional. `01 = The invalid date.
001943	The date '`01' is not in one of the accepted formats listed below: MM/DD/YYYY [BC | AD] [HH:MI[:SS[.NNNNN N]] [AM | PM]] MM-DD-YYYY [BC | AD] [HH:MI[:SS[.NNNNN N]] [AM | PM]] MM.DD.YYYY [BC | AD] [HH:MI[:SS[.NNNNN N]] [AM | PM]] SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	The date specified with the INPUT command was not in one the default formats. Reenter the date in a valid format. `01 = The invalid date.
001944	The date '`01' is not in the format specified by SQR_DB_DATE_FORMAT or in one of the accepted formats listed below: DD-MON-YY SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	(Oracle) The date was not in one of the expected formats for this database. `01 = The invalid date.

Error Number	*Error Message*	*Suggestion/Interpretation*
001944	The date '`01' is not in the format specified by SQR_DB_DATE_FORMAT or in one of the accepted formats listed below: Mon DD YYYY [HH:MI[:SS[.NNN]][AM | PM]] Mon DD YYYY [HH:MI[:SS[:NNN]][AM | PM]] YYYYMMDD [HH:MI[:SS[.NNN]][AM | PM]] YYYYMMDD [HH:MI[:SS[:NNN]][AM | PM]] SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	(Sybase) The date was not in one of the expected formats for this database. `01 = The invalid date.
001944	The date '`01' is not in the format specified by SQR_DB_DATE_FORMAT or in one of the accepted formats listed below: Mon DD YYYY [HH:MI[:SS[.NNN]][AM | PM]] Mon DD YYYY [HH:MI[:SS[:NNN]][AM | PM]] YYYYMMDD [HH:MI[:SS[.NNN]][AM | PM]] YYYYMMDD [HH:MI[:SS[:NNN]][AM | PM]] SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	(ODBC) The date was not in one of the expected formats for this database. `01 = The invalid date.
001944	The date '`01' is not in the format specified by SQR_DB_DATE_FORMAT or in one of the accepted formats listed below: YYYY-MM-DD HH:MI:SS.NNN SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	(Informix) The date was not in one of the expected formats for this database. `01 = The invalid date.

Error Number	*Error Message*	*Suggestion/Interpretation*
001944	The date '`01' is not in the format specified by SQR_DB_DATE_FORMAT or in one of the accepted formats listed below: YYYY-MM-DD[-HH.MI.SS[.NNNNNN]] MM/DD/YYYY DD.MM.YYYY SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	(DB2) The date was not in one of the expected formats for this database. `01 = The invalid date.
001945	SQR does not support dates before '`01'.	SQR does not support dates before the one specified in the message. `01 = Smallest date
001946	The date variables are incompatible with each other.	The SQR function references two date variables that cannot logically be used together, for example, DateDiff of 'date- only' and 'time-only' dates.
002000	Procedure name used more than once: '`01'.	Specify a unique name for the procedure. `01 = Procedure name
002001	Could not find procedure: '`01'.	Verify the spelling of the procedure name. `01 = Procedure name
002002	DO arguments do not match procedure's.	The argument lists for the DO and BEGIN- PROCEDURE commands must match in both type and count. Correct the source line.
002003	DO argument must be $string or #number to accept returned value.	Correct the syntax.
002100	Edit string too long.	The edit mask must be less than 255 characters. Reduce the length of the edit mask.

Error Number	*Error Message*	*Suggestion/Interpretation*
002101	Bad numeric 'edit' format: `01	The numeric edit mask contains an invalid character. See the PRINT command for the valid numeric edit mask characters. `01 = Invalid character
002103	DOLLAR-SYMBOL must be a single alphanumeric character or its decimal value enclosed in brackets: <nnn>.	Correct the syntax.
002104	DOLLAR-SYMBOL cannot be any of the following characters: `01	Correct the syntax. `01 = List of invalid characters
002106	MONEY-SYMBOL must be a single alphanumeric character or its decimal value enclosed in brackets: <nnn>.	Correct the syntax.
002107	MONEY-SYMBOL cannot be any of the following characters: `01	Correct the syntax.
002200	ENCODE string too large; maximum is `01.	Break up the ENCODE command. `01 = Maximum length of an ENCODE string supported by this version of SQR for PeopleSoft.
002300	EXIT-SELECT failed.	The database command to cancel the query returned an error. Try running the SQR program again. The error could be related to a network or server problem. If the error persists, contact your system administrator.
002301	EXIT-SELECT valid only within SELECT paragraph.	Remove the EXIT-SELECT command.
002400	Duplicate label's - do not know which one to GOTO.	Labels must be unique within the section or paragraph where they are defined. Give each label a unique name.

Error Number	*Error Message*	*Suggestion/Interpretation*
002401	(Labels must be in same section or paragraph as GOTO.) Cannot find a matching label for GOTO command.	Check the source code.
002500	Error getting INPUT.	The C routine "fgets()" returned an error and SQR ends the program run.
002501	Unknown INPUT datatype: type={char|number|integer|date}	Correct the syntax.
002502	INPUT STATUS= must reference #variable.	Correct the syntax.
002503	Unknown qualifier for INPUT.	Correct the syntax.
002506	Too long. Maximum `01 characters.	The response to the INPUT statement was too long. Re-enter the data. `01 = Maximum characters allowed
002507	Incorrect. Format for floating point number: [+|-]99.99[E99]	Invalid number was entered for an INPUT request. Re-enter the data.
002508	Incorrect. Format for integer: [+|-]999999	Invalid integer was entered for an INPUT request. Re-enter the data.
002510	A format mask can only be specified when TYPE=DATE is used.	Correct the syntax.
002511	The format mask cannot be stored in a date variable.	Correct the syntax.
002512	The input variable type does not match the TYPE qualifier.	Correct the syntax.
002513	Number too large for INTEGER. Valid range is -2147483648 to 2147483647.	The number was too large to be stored as an integer. Values are from -2147483648 to 2147483647. Re-enter the data.

Error Number	*Error Message*	*Suggestion/Interpretation*
002514	Enter a date in one of the following formats: MM/DD/YYYY [HH:MI[:SS[.NNNNN N]] [AM | PM]] MM-DD-YYYY [HH:MI[:SS[.NNNNN N]] [AM | PM]] MM.DD.YYYY [HH:MI[:SS[.NNNNN N]] [AM | PM]] SYYYYMMDD[HH24[MI[SS[NNNNNN]]]]	The date cannot be blank. Enter a date in one of the specified formats.
002515	`01 required user interaction but user interaction was disabled by the -XI command line flag.	The specified command required user interaction, but user interaction was disabled by the -XI command line flag. `01 = Name of the command
002600	LOAD-LOOKUP table '`01' has not been defined.	Add a LOAD-LOOKUP command. `01 = Load lookup table name
002601	Missing value for `01= in LOAD-LOOKUP.	Correct the syntax. `01 = Name of missing required parameter
002602	Bad value for `01= in LOAD-LOOKUP.	Correct the syntax. `01 = Name of the parameter
002603	LOAD-LOOKUP `01= cannot reference a variable in the Setup section.	Either move the LOAD-LOOKUP command from the Setup section or remove the variable reference. `01 = Name of the parameter
002604	LOAD-LOOKUP names must be unique.	Give each LOAD-LOOKUP array a unique name.

Error Number	*Error Message*	*Suggestion/Interpretation*
002605	Cannot compile SQL for LOAD-LOOKUP table '`01'.	The database server returned an error while trying to compile the SQL statement needed to process the LOAD-LOOKUP command. Check the column and table names. Also check the WHERE= clause for errors. `01 = Load lookup table name
002606	Could not set up cursor for LOAD-LOOKUP table '`01'.	The database server returned an error while trying to compile the SQL statement needed to set up the LOAD-LOOKUP command. Verify the column and table names. Review the WHERE= clause for errors. `01 = Load lookup table name
002607	Problem executing the cursor for LOAD- LOOKUP table '`01'.	The database server returned an error while trying to execute the SQL statement needed to process the LOAD-LOOKUP command. `01 = Load lookup table name
002609	Integers only allowed in numeric lookup keys.	Correct the source line.
002610	Numeric lookup keys must be <= `01 digits.	Correct the source line. `01 = maximum length supported
002611	Bad return fetching row from database in LOAD-LOOKUP table '`01'.	The database server returned an error while fetching the data. `01 = Load lookup table name
002613	Loading '`01' lookup table ...	This message can be inhibited by using the QUIET argument on the LOAD-LOOKUP command. `01 = Name of the load lookup table `02 = Number of rows loaded

Error Number	Error Message	Suggestion/Interpretation
002615	Warning: `01 duplicate keys found in '`02' lookup table.	This message can be inhibited by using the QUIET argument on the LOAD-LOOKUP command. `01 = Number of duplicate keys `02 = Name of the load lookup table
002616	LOAD-LOOKUP `01= must reference a numeric variable or literal.	Correct the source line. `01 = Name of the parameter
002617	LOAD-LOOKUP `01= must reference a string variable or literal.	Correct the source line. `01 = Name of the parameter
002618	LOAD-LOOKUP `01= variable '`02' has not been defined.	Correct the source line. `01 = Name of the parameter `02 = Name of the undefined variable
002619	LOAD-LOOKUP cannot support `01 rows; maximum is `02.	Reduce the ROWS= value. `01 = ROWS= value `02 = Maximum value allowed
002620	`01 command not allowed with -XL option in effect.	Either use the #IF command to conditionally compile the program when -XL is being used or do not execute this SQR report with the -XL option. `01 = SQR command
002700	Line to stop erasing for 'NEW-PAGE' is larger than the page depth.	Correct the source line.
002800	'ON-BREAK' not appropriate for numeric data.	The ON-BREAK argument to the PRINT command may be used only for text fields. Move the #numeric variable to a $string variable, and then print the $string variable.
002801	SET= and LEVEL= must be >= zero when indicated.	Correct the source line.

Error Number	*Error Message*	*Suggestion/Interpretation*
002802	Cannot use old style PROCEDURE= with BEFORE= or AFTER=.	Correct the syntax.
002803	Out of ON-BREAKS; use -Mfile to increase ONBREAKS.	Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
002804	SET= must be same for all ON-BREAKs in Select.	All the ON-BREAKS in a query must belong to the same SET. Use SET= to differentiate between ON-BREAKs in different queries. Correct the source line.
002805	ON-BREAK with BEFORE or AFTER must be inside Select.	Correct the source line.
002806	SAVE= must be a $string variable.	Correct the syntax.
002900	Record :types are FIXED, VARY or FIXED_NOLF (default is VARY).	Correct the syntax.
002901	STATUS variable for `01 must be #Numeric.	Correct the syntax. `01 = SQR command affected
002902	OPEN missing required qualifiers: RECORD={rec_len} FOR-READING|FOR- WRITING|FOR-APPEND	Correct the syntax.
002903	Too many external files opened; maximum is `01.	Reduce the number of open external files needed by the program. `01 = Maximum number of open external files supported by this version of SQR
002904	File number already opened.	Verify the program logic.

Error Number	*Error Message*	*Suggestion/Interpretation*
002905	Cannot open file '`01' AS `02.(`03): `04	SQR stops. `01 = Filename `02 = File number `03 = System error code `04 = System error message
002906	Cannot close file `01. (`02): `03	SQR stops. `01 = File number `02 = System error code `03 = System error message
002907	Problem closing user file(s) at the end of run.	This message may indicate system problems.
002908	Warning: Cannot CLOSE file `01 -- file not opened.	While not an error, this message indicates a problem with your SQR code. `01 = File number
003000	PAGE-NUMBER strings too long.	The pre-and post-PAGE-NUMBER strings must be less than 74 characters. Correct the source line.
003100	Cannot find document marker referenced in POSITION command.	Defines the specified @ marker in a BEGIN- DOCUMENT paragraph. Verify that the @ marker names are spelled correctly.
003101	Only 'COLUMNS nn...' allowed after document marker in POSITION command.	Correct the syntax.
003200	Specified file number not opened for reading.	Files must be opened for reading in order to use the READ command with them. Correct the program logic.
003201	Line `01: Error reading the file. (`02): `03	`01 = Program line number `02 = System error code `03 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
003202	Specified file number not opened for writing.	Files must be opened for writing in order to use the WRITE command with them. Correct the program logic.
003203	Line `01: Error writing the file. (`02): `03	`01 = Program line number `02 = System error code `03 = System error message
003204	Length of variables exceeds record length.	The total of the lengths indicated in the command must be less than the RECORD= argument used on the OPEN command. Search for a typographical error or recalculate the RECORD= value.
003205	Numeric binary transfer allowed with FIXED or FIXED_NOLF records only.	By default, all files are opened in VARY (variable length) mode, thus prohibiting the transfer of numeric binary data. Add the:FIXED or FIXED_NOLF option to the RECORD= argument on the appropriate OPEN command.
003206	Command not complete.	Correct the syntax.
003207	File number must be a numeric literal, variable, or column.	Correct the syntax.
003208	Missing required length in READ command.	Correct the syntax.
003209	Bad length for READ or WRITE command.	Correct the syntax.
003210	$String or #numeric variables required for READ.	Correct the syntax.
003211	#Numeric variables and literals must have :length of 1, 2 or 4 bytes.	Correct the syntax.
003212	#Numeric variables and literals on CDC may only have :length of 1 or 3 bytes.	Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
003213	:Length not allowed for $date variables, length of 18 is assumed.	Correct the syntax.
003300	Unknown qualifier for STOP.	Correct the syntax.
003301	Program stopped by user request.	This is an informational message.
003400	Wrap not appropriate for numeric data.	The WRAP argument to the PRINT command may be used only for text fields. Move the #numeric variable to a $string variable first, and then print the $string variable.
003401	Max `01 chars/line for reverse WRAP.	Reduce the number of characters specified. `01 = Maximum number of characters supported by this version of SQR.
003402	Max `01 chars/line for WRAP with ON= or STRIP=	Reduce the number of characters specified. `01 = Maximum number of characters supported by this version of SQR
003403	Bad <number> in WRAP qualifier.	The number inside the angled brackets must be a valid number (1 - 255). Correct the source line.
003404	Missing '>' in WRAP qualifier.	A leading "<" in the ON= or STRIP= qualifier indicates that a numeric value is following, which must be ended by a closing ">". Correct the source line.
003405	The value for '`01' (`02) must be `03 0.	The value specified for the specified qualifier is invalid. Correct the program logic. `01 = Qualifier name `02 = Value encountered `03 = Relation to zero (<,<=,=,>=,>)

Error Number	*Error Message*	*Suggestion/Interpretation*
003500	PUT, GET or ARRAY- xxxx command incomplete. Required word missing.	Correct the syntax.
003501	Did not find end of literal.	The ending quote character (') was not found at the end of the literal. Add the ending quote character.
003502	Literal too long.	Literal strings can be up to 256 characters long. Break up the literal into smaller pieces and combine using the LET command.
003503	Unknown variable type.	Variable names must begin with $, #, or &. Correct the source line.
003504	Cannot find 'array_name (#element)'.	The element number was not specified. Correct the source line.
003505	'(#Element)' variable not found for array.	Each GET or PUT command must indicate the element or row number to access in the array. Correct the source line.
003506	Array specified not defined with CREATE-ARRAY.	Use the CREATE-ARRAY command to define each array before referencing that array in other commands. Verify that the array names are spelled correctly.
003507	Bad element reference for array (#variable|123).	The element number is larger than the number of rows defined in the CREATE-ARRAY command. Review the program logic to make sure that the element number was not inadvertently changed.
003508	Did not find ending ')' for field.	The "occurs" number for an array field is missing a right parenthesis. Correct the source line.
003509	Field not defined in array: `01	Verify that there are no misspelled field names in the CREATE-ARRAY command. `01 = Undefined field name

Error Number	*Error Message*	*Suggestion/Interpretation*
003510	More variables than fields specified in array command.	The ARRAY command must not have more variables listed to the left of the array name than there are matching fields defined for the array. Check against the CREATE-ARRAY command.
003511	More variables in command than fields in array.	The ARRAY command must not have more variables listed to the left of the array name than there are matching fields defined for the array. Check against the CREATE-ARRAY command.
003512	Only numeric variables and fields allowed with array arithmetic commands.	The ARRAY-ADD, ARRAY-SUBTRACT, ARRAY-MULTIPLY, and ARRAY-DIVIDE commands may have only numeric variables or literals as the source fields. Move the string data into a #numeric variable and then reference the #numeric variable.
003513	GET can only be used with $string or #numeric variables.	You can move array fields only into $string variables or #numeric variables. Correct the source line.
003514	PUT and GET variables must match array field types.	When moving data into or out of arrays, the source or destination variables must match the array fields in type. CHAR fields can be stored into/from strings, NUMBER fields into/from numeric variables. Check the CREATE-ARRAY command.
003515	More fields than variables found in array command.	The ARRAY command must not have more variables listed to the left of the array name than there are matching fields defined for the array. Check against the CREATE-ARRAY command.
003516	Too many arrays defined; maximum is `01.	Reduce the number of arrays needed by the program. `01 = Maximum number of arrays supported by this version of SQR

Error Number	*Error Message*	*Suggestion/Interpretation*
003517	Missing '=specifier' in qualifier: `01	Correct the syntax. 01 = Name of missing required parameter
003518	Duplicate array name: `01	Change the name of the array. `01 = Array name in question
003519	Too many fields defined; maximum is `01.	Reduce the number of fields. `01 = Maximum number of fields allowed per array
003520	Missing ':type' in CREATE-ARRAY FIELD= `01	Correct the syntax. `01 = The name of the field
003521	Duplicate FIELD name: `01	Change the name of one of the fields. `01 = The name of the field
003522	Optional :nn for FIELD must be between 1 and 64K.	Correct the source line.
003523	CREATE-ARRAY FIELDS :type must be one of the following: `01	Correct the syntax.
003525	Missing NAME= in CREATE-ARRAY.	Correct the syntax.
003526	Missing or incorrect SIZE= in CREATE-ARRAY.	Correct the syntax.
003527	Missing FIELD= statements in CREATE-ARRAY.	Correct the syntax.
003528	Array dimensioned too large for PC in CREATE-ARRAY.	On 32-bit systems, the maximum allocation that can be made is 65520 characters. The array as specified would exceed this limit. Reduce the number of entries.
003529	Missing or invalid initialization value for field `01.	Correct the syntax. 01 = Name of the field

Error Number	*Error Message*	*Suggestion/Interpretation*
003600	Missing 'ask' variable name.	Correct the syntax.
003601	Out of substitution or #DEFINE variables; use -Mfile to increase SUBVARS.	Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
003603	WARNING: Substitution variables do not vary when saved with run-time.	This is an informational message.
003605	No substitution variable entered.	The C routine "fgets()" returned an error and SQR ends the program run.
003700	Did not find end of paragraph: `01	The END-paragraph command to match the specified paragraph is missing. Correct the source file. `01 - BEGIN-paragraph in question
003701	Invalid command.	Verify the spelling of the command.
003702	Command not allowed in this section: `01	Correct the syntax. `01 = Offending command name
003703	Paragraph not allowed inside procedure.	The BEGIN-paragraph command is not allowed here. Review your SQR code for a misplaced paragraph.
003704	Missing procedure name.	Correct the syntax.
003705	Extra argument found.	Correct the syntax.
003706	Missing Comma.	Correct the syntax.
003707	Bad Argument List.	The DO or BEGIN-PROCEDURE command has an error in its argument list, possibly extra characters after the final right parentheses. Correct the source line.

Error Number	*Error Message*	*Suggestion/Interpretation*
003708	Empty Argument.	The DO or BEGIN-PROCEDURE command has an error in its argument list, possibly two commas in a row inside the parentheses. Correct the source line.
003709	Only $string and #number variables allowed for BEGIN-PROCEDURE parameters.	Correct the syntax.
003710	Unknown argument type.	An argument in a DO or BEGIN-PROCEDURE command is incorrect. Verify the spelling of the variable types.
003711	Indicate :$string or :#number returned values in BEGIN-PROCEDURE only.	Correct the syntax.
003712	Missing).	Correct the syntax.
003713	`01 paragraph not allowed with -XL option in effect.	Either use the #IF command to conditionally compile the program when -XL is being used or do not execute this SQR report with the -XL option. `01 = Name of the BEGIN-paragraph
003714	Bad database connection number.	The -Cnn value must be a non-zero value. Correct the source line.
003715	Did not find end of paragraph: `01 (No 'from...' clause found.)	Correct the source code. `01 = BEGIN-command in question
003716	Error in SQL statement.	The database server has determined that the SQL statement is in error. The actual error text from the server follows this message. Correct the SQL statement.
003717	Extra characters after expression continuation.	Remove the extra characters after the dash.

Error Number	*Error Message*	*Suggestion/Interpretation*
003718	Did not find end of expression.	An expression in a SELECT list must end with either a &column variable or a position parameter "(Row,Col,Len)". Correct the source line.
003719	Columns names and expressions must be unique or be given unique pseudonyms (&name).	You are trying to select the same &column name more than once. Change the assigned &column name by using an alias after the name. Columns retrieved from the database are assigned names by prepending an "&" to the beginning of the name.
003720	Bad number specified for 'LOOPS=' on 'BEGIN-SELECT; Maximum is 32767'.	If your program logic requires that you stop processing after more than 32767 rows have been retrieved, you could count the rows manually and use the EXIT-SELECT command to break out of the SELECT loop.
003721	Bad param found on 'BEGIN-SELECT' line; Format is: BEGIN-SELECT [DISTINCT] [-Cnn] [-Bnn] [LOOPS=nn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(DB2) Correct the syntax.
003721	Bad param found on 'BEGIN-SELECT' line; Format is: BEGIN-SELECT [DISTINCT] [-Cnn] [LOOPS=nn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Informix) Correct the syntax.
003721	Bad param found on 'BEGIN-SELECT' line; Format is: BEGIN-SELECT [DISTINCT] [-Cnn] [LOOPS=nn] [ON-ERROR=procedure[(arg1[,argi]...)]] [-DB=database]	(ODBC) Correct the syntax.
003721	Bad param found on 'BEGIN-SELECT' line; Format is: BEGIN-SELECT [DISTINCT] [-Cnn] [-Bnn] [LOOPS=nn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Oracle) Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
003721	Bad param found on 'BEGIN-SELECT' line; Format is: BEGIN-SELECT [DISTINCT] [-Cnn] [-XP] [LOOPS=nnn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Sybase) Correct the syntax.
003722	Could not set up cursor.	An error occurred while trying to compile the SQL statement. Review any $string variable references. Correct the SQL statement or use the ON-ERROR= option to trap the error during the program run.
003723	Problem executing cursor.	An error occurred while trying to execute the SQL statement. Review the $string variable references. Correct the SQL statement or use the ON-ERROR= option to trap the error during the program run.
003724	Could not exit query loop.	The database command to cancel the query returned an error. Try running the SQR program again. The error could be related to a network or server problem. If the error persists, contact your system administrator.
003725	Bad return fetching row from database.	The database returned an error status for the last row that was fetched. This is commonly due to the buffer not being large enough. If selecting expressions, make sure that the length of the first expression is adequate for all rows selected.
003726	Literal in SQL expression missing closing quote.	Literals must be surrounded by single quotes ('). To embed a quote within a literal use two single quotes in sequence ('). Correct the source line.
003727	SQL expression not ended, perhaps parentheses not balanced.	An expression in a SELECT list must end with either a &column variable or a position parameter "(Row,Col,Len)". Correct the source line.

Error Number	*Error Message*	*Suggestion/Interpretation*
003728	SQL expression not ended, perhaps missing &name.	An expression in a SELECT list must end with either a &column variable or a position parameter "(Row,Col,Len)". Correct the source line.
003729	SQL expression is missing &name or has unbalanced parentheses.	An expression in a SELECT list must end with either a &column variable or a position parameter "(Row,Col,Len)". Correct the source line.
003730	Incorrect arguments for BEGIN-SQL: [-Cnn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(DB2) Correct the syntax.
003730	Incorrect arguments for BEGIN-SQL: [-Cnn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Informix) Correct the syntax.
003730	Incorrect arguments for BEGIN-SQL: [-Cnn] [-NR] [ON-ERROR=procedure(arg1[,argi]...)]] [-DB=]	(ODBC) Correct the syntax.
003730	Incorrect arguments for BEGIN-SQL: [-Cnn] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Oracle) Correct the syntax.
003730	Incorrect arguments for BEGIN-SQL: [-Cnn] [-XP] [ON-ERROR=procedure[(arg1[,argi]...)]]	(Sybase) Correct the syntax.
003731	Did not find 'END- SQL' after 'BEGIN- SQL'.	Correct the source file.
003732	ON-ERROR= for 'BEGIN-SQL' in SETUP section must be STOP, WARN or SKIP.	Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
003733	Could not create procedure for SQL.	(Sybase) SQR could not create a stored procedure for the SQL statement. The most likely cause for failure is that the user name you are using to run the report under does not have the proper privileges. Either grant the user CREATE PROCEDURE privilege or use the -XP command line option to inhibit SQR from creating temporary stored procedures for SQL statements.
003734	Could not compile SQL.	Correct the SQL statement or use the ON-ERROR= option to trap the error during the program run.
003735	Could not execute SQL.	An error occurred while trying to compile the SQL statement. Correct the SQL statement or use the ON-ERROR= option to trap the error during the program run.
003736	Please use BEGIN-SELECT - END-SELECT section for SELECT statements.	(Oracle, Informix, ODBC) Correct the source code.
003737	Bad fetch buffer count.	(Oracle, Sybase) The -B flag specifies an illegal value. Correct the source code.
003738	Report interrupted by request.	This is an informational message.
003741	Dynamic column must be $string variable.	Correct the syntax.
003742	Dynamic column missing '`01'.	Correct the syntax. `01 = Missing character
003743	Dynamic columns must have a &pseudonym.	Correct the syntax.
003744	&pseudonym =type must be 'char', 'number', or 'date'.	Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
003745	Only a variable name may be between the '`01' and '`02' characters.	Correct the syntax. `01 = Leading character `02 = Trailing character
003746	When dynamic columns are used all non-dynamic columns and expressions must be defined with &name=type.	Add &name=type to all expressions and non-dynamic columns.
003747	When the table name is dynamic each column and expression must be defined with &name=type.	Add &name=type to all expressions and non-dynamic columns.
003800	Too many document paragraphs; maximum is `01.	There are too many BEGIN-DOCUMENT paragraphs. Reduce the number of DOCUMENT paragraphs needed by the program. `01 = Maximum number supported by this version of SQR
003801	Too many document markers; maximum is `01.	There are too many BEGIN-DOCUMENT paragraphs. Reduce the number of DOCUMENT paragraphs needed by the program. `01 = Maximum number supported by this version of SQR
003802	Duplicate document marker.	Specify a unique name for the the document marker.
003803	Did not find 'END- DOCUMENT' after 'BEGIN-DOCUMENT'.	The BEGIN-DOCUMENT paragraph must end with END-DOCUMENT. Correct the source code.
003900	EXECUTE command is incomplete.	Correct the syntax.
003901	Bad -Cnn connection number for EXECUTE.	The -Cnn value must be a nonzero value. Correct the source line.
003902	@#Return_status must be #numeric (missing #).	Correct the source line.

Error Number	*Error Message*	*Suggestion/Interpretation*
003903	Missing '=' after `01.	Correct the source line. `01 = The parameter in question
003904	Unknown variable type.	Variable names must begin with $, #, or &. Correct the source line.
003905	OUT[PUT] variables for EXECUTE may only be $variable or #variable.	Correct the syntax.
003906	The only EXECUTE option is WITH RECOMPILE.	Correct the syntax.
003907	You must EXECUTE ... INTO &columns.	Correct the syntax.
003908	Unknown datatype for EXECUTE...INTO &columns.	Verify the spelling of the data type. If the data type is correct, then contact customer technical support so SQR can be updated.
003909	EXECUTE...INTO &columns must be unique.	The &column name assigned to the column must be unique throughout the report. Specify a unique name for the column.
003910	Missing (length) for datatype in EXECUTE.	Correct the source line.
003911	Datatype should not have (length) in EXECUTE.	Correct the source line.
003912	DO= in EXECUTE requires INTO... variables.	Correct the syntax.
003913	Could not EXECUTE stored procedure.	Record the database error message displayed with this message. If needed, contact your system administrator.

Error Number	*Error Message*	*Suggestion/Interpretation*
003914	Bad return fetching row from database.	Record the database error message displayed with this message. If necessary, contact your system administrator.
003915	Could not set up EXECUTE cursor.	The database server returned an error while trying to compile the SQL statement needed to set up the EXECUTE command.
004000	Result #variable or $variable or '=' missing in expression.	The LET command is not properly formatted. Correct the source line.
004001	Expression too complex.	The expression is either too long or is too deeply nested. Break the expression into smaller expressions.
004002	Parentheses unbalanced in expression.	A left or right parenthesis is missing. Correct the source line.
004003	Too many variables; maximum is `01.	Break the expression into smaller expressions. `01 = Maximum number supported by this version of SQR
004004	Empty expression.	The expression is invalid. Correct the source line.
004005	Extra comma in expression.	An argument is missing after a comma in the expression. Correct the source line.
004006	Unknown operator '`01'. Do you mean `02 ?	The concatenation operator is ||. Correct the source line.
004007	Too many &column forward references in expression; maximum is `01.	The expression contains too many forward references. Break the expression into smaller expressions. `01 = Maximum number supported by this version of SQR

Error Number	*Error Message*	*Suggestion/Interpretation*
004008	Unknown function or variable in expression: `01	The specified function is not an SQR built-in function nor does it exist in the user-modifiable file UFUNC.C. Verify that the function names are spelled correctly. `01 = Function name
004009	Function '`01' missing parentheses.	All functions in an expression must be followed by their arguments enclosed in parentheses. Correct the source line.
004010	Empty parentheses or expression.	A pair of parentheses were found with nothing inside them. Remove the () in question from the source line.
004011	User function '`01' has incorrect number of arguments.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004012	Function '`01' has incorrect number of arguments.	Correct the syntax of the function. Functions are described under the LET command. `01 = SQR function name
004013	Missing operator in expression.	Correct the source line.
004014	Operator '`01' missing argument.	Correct the syntax of the function. Functions are described under the LET command. `01 = Operator
004015	Function '`01' missing argument.	Correct the syntax of the function. Functions are described under the LET command. `01 = SQR function name
004016	Function or operator '`01' missing arguments.	Correct the syntax of the function. Functions are described under the LET command. `01 = SQR function name

Error Number	*Error Message*	*Suggestion/Interpretation*
004017	User function '`01' requires character argument.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004018	User function '`01' requires numeric argument.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004019	User function '`01' requires $string variable.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004020	User function '`01' requires #numeric variable.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004021	User function '`01' has incorrect argument type list. Must be of: c,n,C,N	The UFUNC.C file has a bad definition for the specified function. Correct the UFUNC.C program file, recompile UFUNC.C, and recreate the SQR executable. `01 = User function name
004022	User function '`01' missing arguments.	Look at the file UFUNC.C to determine the correct number and type of arguments required for the specified function. `01 = User function name
004023	User function '`01' has incorrect return type. Must be c or n.	The UFUNC.C file has a bad definition for the specified function. Correct the UFUNC.C program file, recompile UFUNC.C, and recreate the SQR executable. `01 = User function name

Error Number	*Error Message*	*Suggestion/Interpretation*
004024	'isnull' requires a &column, $string or $date argument.	#numeric variables cannot be NULL. Correct the source line.
004025	'nvl' requires a &column, $string or $date as its first argument.	#numeric variables cannot be NULL. Correct the source line.
004026	Function or operator '`01' requires character argument.	Correct the source line. `01 = Function or operator
004027	Function or operator '`01' requires numeric argument.	Correct the source line. `01 = Function or operator
004028	IF or WHILE expression must return logical result.	The expression used must evaluate a statement that will be TRUE or FALSE. Correct the source line.
004029	Attempting division by zero in expression.	The expression tried to divide a number by zero. Use the COND() function to determine whether the divisor is zero; then divide by something else (for example, 1).
004030	Attempting division by zero with '%'.	An attempt was made to divide a number using the " %" operator. Use the COND() function to determine whether the divisor is zero; then divide by something else (for example, 1).
004031	The number used with '%' (`01) is out of range.	The "%" operator works only with integers. Correct the program logic. `01 = Maximum value allowed
004032	User function has unknown return type -- expecting n or c -- need to recompile Run-Time file?	SQR detected an error while processing a user defined function. If you are running an .sqt file, it probably needs to be recompiled because the user function has changed its definition. If you are running an .sqr file, you need to correct the UFUNC.C program file, recompile UFUNC.C, and recreate the SQR executable.

Error Number	Error Message	Suggestion/Interpretation
004033	In user function use C type with allocated string to change $variable.	SQR detected an error while processing a user defined function. Correct the UFUNC.C program file, recompile UFUNC.C, and recreate the SQR executable.
004034	Could not find array '`01' in ARRAY function.	Verify the spelling of the array name. `01 = Array name
004035	Could not find array field '`01' in ARRAY function.	Verify the spelling of the array field name. `01 = Array field name
004036	Math error in expression (usually over- or under-flow).	Most of the SQR mathematical built-in functions have a corresponding C library routine. One returned an error. Break the expression into discrete expressions in order to identify the function that caused the error.
004037	Error executing expression.	Record the steps leading up to the error and contact technical support.
004038	Out of space while processing expression; Use -Mfile to increase EXPRESSIONSPACE.	The expression requires more temporary string storage than is currently allocated. Use the -Mfile flag on the command line to specify a file that contains an entry that increases by a greater value than is currently defined.
004039	'`01' assumed to be a variable name, not an expression.	Warning message. `01 = Expression in question
004040	The array '`01' has not been defined.	Define the array using the CREATE-ARRAY command. `01 = Array name
004041	The field '`01' is not valid for array '`02'.	Correct the source code. `01 = Field name `02 = Array name

Error Number	*Error Message*	*Suggestion/Interpretation*
004042	The array reference '`01' has an incorrect number of parameters specified.	Correct the source code. `01 = Array name
004043	The array reference '`01' requires numeric parameters for the element and occurs arguments.	Correct the source code. `01 = Array name.
004045	Function or operator '`01' requires date argument.	Correct the source code. `01 = Array name
004046	Incompatible types between expression and variable.	Correct the source code.
004047	The field '`01' is must be 'char' or 'float'.	Correct the source code. `01 = Field name
004048	Function or operator '`01' must be a string or date argument.	Correct the source line. `01 = Function or operator
004100	Use 'print' command to format data outside SELECT query.	You must precede PRINT command arguments (WRAP, ON-BREAK.) with an explicit PRINT command when outside of a BEGIN-SELECT paragraph. Correct the source line.
004101	Cannot find required parameter.	Correct the syntax.
004102	Bad number found.	A command expecting a numeric literal or :#numeric variable reference found an illegal number definition or a reference to a string variable or column. Correct the source line.
004103	Cannot find required numeric parameter.	Correct the syntax.
004104	Cannot find placement parameters.	The position qualifier "(Row,Col,Len)" was not found. Search for a missing parentheses.

Error Number	*Error Message*	*Suggestion/Interpretation*
004105	Placement parameter incorrect.	The "Row", "Column" or "Length" fields are invalid or ill-formed. Correct the source line.
004106	Invalid second function on line.	An SQR command used as a qualifier for a primary command (for example, PRINT) is incorrect. Correct the source line.
004107	Second function must be FORMAT type.	The PRINT command may have format command qualifiers such as WRAP, CENTER, or FILL. Other qualifier commands are not permitted.
004108	Missing operator =, <, >, ...	Correct the source line.
004109	Invalid operator.	Correct the source line.
004110	Missing variable.	Correct the syntax.
004111	Please give this expression a &pseudonym.	Expressions selected in a BEGIN-SELECT paragraph should be given an &Name or be followed by a print position "(Row,Col,Len)". Correct the source line.
004112	Wrong variable type.	Correct the syntax.
004113	Command incomplete, expected '`01'.	Correct the syntax. `01 = What was expected
004114	Expecting '`01', found '`02'.	Correct the syntax. `01 = What was expected `02 = What was encountered
004115	Unknown command or extra parameters found (missing quotes?).	Correct the syntax.
004116	Duplicate references to parameter '`01'.	Correct the syntax. `01 = Duplicated parameter

Error Number	*Error Message*	*Suggestion/Interpretation*
004117	Unexpected equal sign found with ''`01'.	Correct the syntax. `01 = Parameter name
004118	Qualifier ''`01' cannot be used with the following qualifiers:	Correct the syntax. `01 = Qualifier name
004119	Expecting numeric column, found string column.	Correct the syntax.
004120	Date variables (`01) cannot be used with this command.	Correct the syntax. `01 = Parameter name
004200	Page width and depth must be > 0 and < 32767.	The values specified with the PAGE-SIZE command are out of specified range. Specify legal values.
004201	Page buffer must be < 65536 on PC SQR.	The maximum page buffer allocation on a PC is 65536. The Page-Depth * Page-Width cannot exceed this value. Reduce the Page-Depth or Page-Width.
004202	Cannot generate line printer output for this report because position qualifier(s) may be out of range. If you are running this report, specify PRINTER:{HP,EH,HT ,PS,WP} for graphical printer output.	The report output cannot be generated for a Line Printer. If your report was designed for a graphics printer, specify -PRINTER:{HP,EH,HT,PS,WP} for graphical printer output.
004300	Missing end of placement (...) in SHOW.	The placement parameter is ill-formed. Correct the source line.
004301	Bad (...) location in SHOW.	Screen positions must be valid numbers. Correct the source line.
004302	Missing literal or variable name to EDIT in SHOW.	A literal or variable name must immediately precede the EDIT, NUMBER, MONEY, or DATE keywords.

Error Number	*Error Message*	*Suggestion/Interpretation*
004303	Missing edit mask in SHOW.	The word EDIT must be followed by a valid edit mask. Correct the source line.
004304	Only string variable allowed for dynamic edit mask.	Dynamic edit masks may only be stored in $variables. Correct the line.
004305	Unknown option for SHOW.	Correct the syntax.
004400	Program too large; use -Mfile to increase PROGLINES.	The SQR program contains too many SQR command lines. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004401	Out of param storage; use -Mfile to increase PROGLINEPARS.	The SQR program contains too many SQR command line parameters. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004402	Out of string storage; use -Mfile to increase STRINGSPACE.	The space allocated to hold the static string variables ('...') has been used. Use the - Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004403	Out of variables; use - Mfile to increase VARIABLES.	There are too many variables (string, numeric), literals and database columns. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004405	Out of forward &column or $variable references; use -Mfile to increase FORWARDREFS.	A forward referenced variable is a variable that is referenced before it is defined. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.

Error Number	*Error Message*	*Suggestion/Interpretation*
004406	Number `01 not allowed.	Use a different value. `01 = Internal number
004407	Referenced variables not defined:	References were made to column variables (&var) that are not defined in the program. The list of variable names follows this message.
004500	Out of Print positions; use -Mfile to increase POSITIONS.	A print position is the "(Row,Col,Len)" parameter. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004501	Use '+' and negate variable for reverse relative placement.	The use of "-#variable" is not legal here. Negate the #variable value and use "+#variable".
004503	Fixed line placement #variable must be > 0. Use relative positioning, (+#line,10,0).	Correct the source line as indicated.
004504	Fixed column placement #variable must be > 0. Use relative positioning, (5,+#col,0).	Correct the source line as indicated.
004505	Length placement #variable must be >= 0.	The length field cannot be a negative value. Correct the source line.
004600	CODE not appropriate for numeric data.	The CODE qualifier to the PRINT command may only be used for text fields. Move the "#Variable" to a "$Variable" first and then print the "$Variable".
004601	Unknown option for GRAPHIC command: BOX, HORZ-LINE, VERT-LINE or FONT	Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
004602	GRAPHIC BOX out of bounds. Row: `01, Column: `02, Width: `03, Depth: `04	SQR ends the program run. `01 = Row `02 = Column `03 = Width `04 = Depth
004603	GRAPHIC VERT- LINE out of bounds. Row: `01, Column: `02, Length: `03	SQR ends the program run. `01 = Row `02 = Column `03 = Length
004604	GRAPHIC HORZ- LINE out of bounds. Row: `01, Column: `02, Length: `03	SQR ends the program run. `01 = Row `02 = Column `03 = Length
004700	Cannot open the program file: '`01' (`02): `03	Depends on the system error message. `01 = Name of the program file `02 = System error code `03 = System error message
004701	Cannot logon to the database.	The connectivity information is either incorrect or the database server is unavailable. Verify the connectivity information and the server availability.
004702	Line found outside paragraph.	All commands must be within BEGIN-... END-statements. Correct the source code.
004703	Cannot close the program file. (`01): `02	Depends on the system error message. `01 = System error code `02 = System error message
004704	#ENDIF not found for #IF.	Missing an #ENDIF to complete conditional compilation. Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
004705	Program line too long; maximum is `01.	Break the program line into smaller lines. `01 = Maximum line length supported by this version of SQR
004706	Substitution variable {`01} would cause this line to exceed the maximum line length of `02 characters.	The substitution variable value would cause this line to exceed the maximum line size. Break the program line into smaller lines. `01 = Name of the substitution variable `02 = Maximum line length supported by this version of SQR
004707	No value found for substitution variable: {`01}	An empty value was found for the substitution variable. Verify the spelling of the substitution variable. `01 = Name of the substitution variable
004708	#ELSE without preceding #IF.	Missing an #IF or #IFDEF or #IFNDEF to begin conditional compilation. Correct the source code.
004709	#ENDIF without preceding #IF.	Missing an #IF or #IFDEF or #IFNDEF to begin conditional compilation. Correct the source code.
004710	#IF's nested too deeply; maximum is `01.	Reduce the number of nested #IF directives. `01 = The maximum depth supported by this version of SQR
004711	#INCLUDE files nested too deeply; maximum is `01.	Reduce the number of nested #INCLUDE directives. `01 = The maximum depth supported by this version of SQR

Error Number	*Error Message*	*Suggestion/Interpretation*
004712	Include file name too long; Modify -I flag.	The combined -I directory name with the #INCLUDE file name exceeds the maximum length permitted for a complete path name. Verify the spelling of both the -I command flag and the #INCLUDE filename.
004713	Cannot open the #INCLUDE file: '`01' (`02): `03	`01 = Include file name `02 = System error code `03 = System error message
004714	Cannot close the #INCLUDE file: '`01' (`02): `03	`01 = Include file name `02 = System error code `03 = System error message
004716	'BEGIN-REPORT' command not found in program.	This section is required for all reports. Correct the source code.
004717	Cannot open the report output file: '`01' (`02): `03	`01 = Output file name `02 = System error code `03 = System error message
004719	Cannot logoff the database.	The database server returned an error while trying to log off from the database. SQR ends the program run.
004720	Cannot open the run-time file: '`01'. (`02): `03	SQR ends the program run. `01 = Run-Time file name `02 = System error code `03 = System error message
004721	Cannot close the run-time file. (`01): `02	SQR ends the program run. `01 = System error code `02 = System error message
004722	Error reading the run-time file. (`01): `02	SQR ends the program run. `01 = System error code `02 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
004723	Run time file must be recreated for this version of SQR.	The runtime file was created by a earlier version of SQR and is incompatible with the current version. Recreate the .sqt (runtime) file.
004724	The -XL option cannot be specified with this run-time file because access to the database is required.	Do not use the -XL option.
004725	Cannot open cursor.	The database server returned an error indicating that a new database cursor or logon could not be completed. See the error message from the database server.
004726	Cannot create procedure for SQL statement.	(Sybase) SQR could not create a stored procedure for the SQL statement. The most likely cause for failure is that the user name you are running the report under does not have the proper privileges. Either grant the user CREATE PROCEDURE privilege or use the -XP command line option to inhibit SQR from creating temporary stored procedures for SQL statements.
004727	Error writing the runtime file. (`01): `02	`01 = System error code `02 = System error message
004728	You must specify a Partitioned Data Set name and member to build a .sqt(member) run-time file. Could not create the run-time file.	(MVS) Use the proper format to specify the name of the .sqt file.
004729	Cannot find inactive database cursor. Program too large.	(Oracle, DB2) The program has too many concurrent database cursors. Reduce the complexity of the program.
004730	Run-time saved in file: `01	This is an informational message. `01 = Name of the .sqt file created

Error Number	*Error Message*	*Suggestion/Interpretation*
004735	Unknown variable type encountered in run-time file: `01	SQR ends loading the runtime file. `01 = Variable type
004736	Unexpected End-Of-File while processing the run-time file.	SQR ends loading the runtime file.
004737	Cannot load the run- time file because it was built for the `01database and `02 is built for the `03 database.	SQR ends loading the runtime file. `01 = Database name from runtime file `02 = SQR image name `03 = Database that SQR is built for
004738	'END-REPORT' not paired with 'BEGIN- REPORT'.	Correct the source code.
004739	'END-PROGRAM' not paired with 'BEGIN- PROGRAM'.	Correct the source code.
004743	#INCLUDE filename must be enclosed in quotation marks.	Correct the syntax.
004744	#INCLUDE command format is: #Include 'filename'.	Correct the syntax.
004745	Array field (`01.`02) specification exceeds the PC 64K limit.	Reduce the size of the field requirements. `01 = Array name `02 = Field name
004746	Layout ``01' specifications exceeds the PC 64K limit.	The layout is too large for the 32–bit version of SQR to handle. `01 = Layout name
004747	The SQT file is corrupted and cannot be processed.	SQR ends loading the runtime file.

Error Number	*Error Message*	*Suggestion/Interpretation*
004748	The user function '`01' needs to be defined as entry `02 in the user function table. It requires a definition of: Return Type = '`03' Arg Count = `04 Arg Types = "`05"	The SQT file requires that the specified user function be defined. `01 = User function name `02 = Entry in the user function table `03 = Return type `04 = Argument count `05 = Argument types
004749	An attempt was made to move `01 characters into '`02'. The maximum allowed is `03 characters.	An attempt was made to move too much data into an SQR string variable. `01 = Number of characters to be moved `02 = Variable name `03 = Maximum characters allowed
004802	PRINTER TYPE must be HTML, HPLASER-JET, POSTSCRIPT, or LINEPRINTER.	Correct the syntax.
004805	Both BEFORE-BOLD and AFTER-BOLD must be specified.	Correct the syntax.
004807	Unknown DECLARE qualifier.	Correct the syntax.
004900	Out of dynamic SQL arguments [$...]; use -Mfile to increase DYNAMICARGS.	Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
004901	Date variables (`01) cannot be used in BEGIN-SQL or BEGIN-SELECT paragraphs.	Correct the source code. `01 = Variable name
005000	Report '`01' heading section size exceeds the page depth.	Reduce the size of the heading or increase the page depth.
005001	Report '`01' footing location must be less than the page depth.	Reduce the size of the footing or increase the page depth.

Error Number	*Error Message*	*Suggestion/Interpretation*
005002	Check 'BEGIN- HEADING' commands: Discovered 2nd page-initialization while heading in progress.	The BEGIN-HEADING procedure either caused an overflow of the current page or it issued a command that caused a page eject to occur. Review any procedure invoked by the BEGIN-HEADING section to ensure that the commands do not overflow the page or cause a page eject.
005003	Check 'BEGIN-FOOTING' commands; perhaps number of footing lines is too small. Discovered 2nd page- write while footing in progress.	The BEGIN-FOOTING procedure either caused an overflow of the current page or it issued a command that caused a page eject to occur. Review any procedure invoked by the BEGIN-FOOTING section to ensure that the commands do not overflow the page or cause a page eject.
005004	Attempt to execute the `01 command while processing the `02 section.	Change the SQR program logic to prevent the command from executing while the specified section is active. `01 = Command name `02 = Section name
005005	Report '`01' already has been assigned a `02 section.	Correct the source code. `01 = Report name `02 = Duplicated section name
005006	You cannot define more than one default '`01' section.	Correct the source code. `01 = Duplicated section name
005007	Report '`01' has overlapping heading and footing sections.	Correct the source code. `01 = Report name
005008	TOC '`01' already has been assigned a `02 section.	Correct the source code. `01 = Table of Contents name `02 = Duplicated section name

Error Number	*Error Message*	*Suggestion/Interpretation*
005100	'IF', 'WHILE', 'EVALUATE' commands nested too deeply; maximum is `01.	Reduce the nested commands. `01 = Maximum depth allowed by this version of SQR
005101	'BREAK' found outside 'WHILE' or 'EVALUATE' statement.	The BREAK command is valid only in the context of a WHILE or EVALUATE statement. Correct the source code.
005102	Out of Break commands; Use -Mfile to increase BREAKS.	This is the number of BREAK commands allowed per EVALUATE command. Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
005103	END-WHILE found without matching 'WHILE'.	Correct the source code.
005104	'IF' or 'EVALUATE' command not completed before 'END-WHILE'.	Correct the syntax.
005105	'ELSE' found without matching 'IF'.	ELSE can be used only within the context of an IF command. Correct the source code.
005106	Single 'ELSE' found inside 'WHILE' or 'EVALUATE' statement.	ELSE can be used only within the context of an IF command. Correct the source code.
005107	Only one 'ELSE' allowed per 'IF'.	Rewrite the source code to use nested IF statements.
005108	Found 'END-IF' without matching 'IF'.	Each IF command must have a matching END-IF command. Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
005109	'WHILE' or 'EVALUATE' command not completed before 'END-IF'.	You are missing a closing END-WHILE or END-EVALUATE command before END- IF. IF, WHILE, and EVALUATE statements can be nested, but they cannot cross each other's boundaries. Each inner statement must be complete before a closing statement is ended. Correct the source code.
005110	EVALUATE statements nested too deep; maximum is `01.	Reduce the number of nested statements. `01 = Maximum depth supported by this version of SQR
005111	'WHEN' found outside 'EVALUATE' clause.	WHEN may be used only in the context of an EVALUATE clause. Correct the source code.
005112	'IF' or 'WHILE' not completed before 'WHEN' statement.	Correct the syntax.
005113	Out of When commands; Use -Mfile to increase WHENS.	Use the -Mfile flag on the command line to specify a file containing an entry that increases the currently defined value.
005114	Incorrect types for comparison. Both must be of the same type (string, numeric or date).	Correct the source line.
005115	'When-other' found outside 'Evaluate' statement.	WHEN can be used only in the context of an EVALUATE statement. Correct the source code.
005116	'IF' or 'WHILE' not ended before 'WHEN- OTHER' command.	Correct the syntax.
005117	Only one 'WHEN- OTHER' allowed per 'EVALUATE'.	Correct the syntax.
005118	Found 'END-EVALUATE' without matching 'EVALUATE'.	Each EVALUATE command must have a matching END-EVALUATE command. Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
005119	'IF' or 'WHILE' command not completed before 'END-EVALUATE'.	Correct the syntax.
005120	'WHEN-OTHER' must be after all 'WHEN's.	Correct the syntax.
005121	No 'WHEN's found inside 'EVALUATE' statement.	Correct the syntax.
005122	'IF', 'EVALUATE' and 'WHILE' statements cannot cross sections or paragraphs.	These commands must be contained within a single section or paragraph. Correct the source code.
005200	Did not find '>' after <....	A leading left angled bracket (<) indicates that you are beginning an decimal value representing a character, which must be ended by a right angled bracket (>). Correct the source line.
005201	Bad character in <...>.	Numbers in angled brackets (<>) must be between 1 and 255. Correct the source line.
005202	Bad number in <...>.	Numbers in angled brackets (<>) must be between 1 and 255. Correct the source line.
005203	<...> string is too long; maximum is `01 characters.	Reduce the length of the string. If this is not possible, use a PRINT-DIRECT command in a BEGIN-REPORT or END-REPORT procedure. `01 = Maximum number of characters supported by this version of SQR
005300	Did not find '=' after qualifier: `01	Correct the syntax. `01 = Qualifier name
005301	Qualifier '`01' requires a numeric value.	Correct the syntax. `01 = Qualifier name

Error Number	*Error Message*	*Suggestion/Interpretation*
005302	Incorrect value for qualifier '`01'. Valid values are:	Correct the source line. `01 = Qualifier name
005303	Invalid qualifier '`01'. Valid qualifiers are:	Correct the source line. `01 = Qualifier name
005304	Qualifier '`01' requires a numeric literal, variable, or column.	Correct the source line. `01 = Qualifier name
005305	Qualifier '`01' references a numeric variable that has not been defined.	Correct the source line. `01 = Qualifier name
005306	Qualifier '`01' requires a string literal, variable, or column.	Correct the source line. `01 = Qualifier name
005307	List not terminated.	Correct the syntax.
005308	Missing comma in list.	Correct the syntax.
005309	Required argument '`01' was not specified.	Correct the source line. `01 = Qualifier name
005310	Qualifier '`01' has already been specified.	Correct the source line. `01 = Qualifier name
005311	Qualifier '`01' requires a string literal.	Correct the source line. `01 = Qualifier name
005312	Qualifier '`01' requires a list of values: (val [,val]...).	Correct the source line. `01 = Qualifier name
005313	Qualifier '`01' requires a integer value.	Correct the source line. `01 = Qualifier name
005314	Invalid character in variable name '`01'.	Correct the source line. `01 = Invalid character

Error Number	*Error Message*	*Suggestion/Interpretation*
005315	Qualifier '`01' references a string variable that has not been defined.	Correct the source line. `01 = Qualifier name
005316	Qualifier '`01' uses an invalid Unit-Of-Measure suffix. Valid suffixes are: dp pt mm cm in	Correct the source line. `01 = Qualifier name
005400	Second page write attempted while writing current page. Check BEFORE- PAGE, AFTER-PAGE procedures.	Review any procedure invoked by the BEFORE-PAGE or AFTER-PAGE procedures to ensure that the commands do not overflow the page or cause a page eject.
005402	String cannot be placed on page: `01 -- placement specified is out of range. (`02,`03,`04)	Ensure the values are within the page limits. `01 = Text value `02 = Row `03 = Column `04 = Length
005403	Error writing the output file. (`01): `02	`01 = System error code `02 = System error message
005404	Cannot open the Postscript startup file: `01 (`02): `03	`01 = Name of the file `02 = System error code `03 = System error message
005405	SQR trial copy exiting after `01 pages.	`01 = Number of pages.
005406	Exiting after requested number of test pages (`01).	`01 = Number of pages.
005408	Program stopped by user request.	This is an informational message.
005500	Cannot set parse_only option.	(Sybase) The DB-Library routine dbsetopt() returned an error. This error should never occur. Contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
005501	Cannot reset parse_only option.	(Sybase) The DB-Library routine dbclropt() returned an error. This error should never occur. Contact technical support.
005502	Cannot drop SQR generated stored procedure: `01.	(Sybase) See the database server error message that was also output. This error should never occur. Contact technical support. `01 = Stored procedure name
005503	Cannot use `01 datatype as bind variable.	(Sybase) Use another database column. `01 = The database datatype.
005504	Unknown datatype for bind variable: `01 Cannot create stored procedure.	(Sybase) Contact technical support. `01 = Unknown database datatype
005505	SQL too large to create stored procedure.	(Sybase) The size of the SQL text needed to create the stored procedure is too large for SQR to process. Add the -XP option to the BEGIN-SQL or BEGIN-SELECT command.
005506	SQR's EXECUTE command not available for this version of Sybase.	(Sybase) Some early versions of Sybase SQL Server or Microsoft SQL Server do not support Remote Procedure Calls (RPCs). Update your database server.
005507	Could not add param to remote procedure call.	(Sybase) A DB-Library routine returned an unexpected error. See the error message from the database.
005508	The number of EXECUTE...INTO &columns does not match the procedure.	(Sybase) Verify the definition for the stored procedure you are referencing.
005509	Incorrect number of INTO &columns defined in EXECUTE.	(Sybase) Verify the definition for the stored procedure you are referencing.

Error Number	*Error Message*	*Suggestion/Interpretation*
005510	Error converting OUTPUT Sybase type for EXECUTE.	(Sybase) The DB-Library routine dbconvert() failed to convert the data from the stored procedure. Contact technical support.
005511	Number of OUTPUT parameters from EXECUTE is incorrect.	(Sybase) Verify the definition for the stored procedure you are referencing.
005512	Missing default database name for USE.	(Sybase) Correct the syntax.
005512	Missing default database name for USE.	(ODBC) Could not connect to the specified datasource.
005513	You may only specify 'USE db' once, before any SQL statements are executed.	(Sybase) Only one USE command is allowed in a report. Place the SETUP section at the beginning of the SQR report.
005515	Undefined variable referenced in -DB flag: `01	(ODBC) Verify that there are no misspellings. `01 = Variable name
005523	Database commit failed.	The database command to perform a commit returned an error. Try running the SQR program again. The error could be related to a network or server problem. If the error persists, contact your system administrator.
005524	Cannot close database cursor.	The database command to close the database cursor returned an error. Try running the SQR program again. The error could be related to a network or server problem. If the error persists, contact your system administrator.
005528	DB2 SQL `01 error `02 in cursor `03:	(DB2) `01 = Routine name `02 = Error code `03 = SQR cursor number

Error Number	*Error Message*	*Suggestion/Interpretation*
005528	INFORMIX SQL `01 error `02 (ISAM: `03) in cursor `04: `05	(Informix) `01 = Routine name `02 = Error code `03 = ISAM code `04 = SQR cursor number `05 = Error message from database
005528	ODBC SQL `01 error `02 in cursor `03: `04	(ODBC) `01 = Routine name `02 = Error code `03 = SQR cursor number `04 = Error message from database
005528	ORACLE `01 error `02 in cursor `03: `04	(Oracle) `01 = Routine name `02 = Error code `03 = SQR cursor number `04 = Error message from database
005528	Sybase `01 error in cursor `02: `03	(Sybase) `01 = Routine name `02 = SQR cursor number `03 = Error message from database
005532	System 10 files are missing.	(Sybase) Contact your system administrator.
005533	Not a System 10 SQL Server.	(Sybase) The CT-Library version of SQR can only connect to a System 10 server. Use the DB-Library version of SQR to connect to a version earlier than System 10 server.
005534	SQL too long for PREPARE/DECLARE; maximum `01 characters.	(DB2) The SQL statement is too large. `01 = Maximum number of characters supported by this version of SQR

Error Number	*Error Message*	*Suggestion/Interpretation*
005536	Unknown error message number: `01.	(DB2) `01 = Error message number
005537	Empty error message returned from system for error number: `01.	(DB2) `01 = Error message number
005538	Invalid SELECT statement; COMPUTE clauses are not supported.	(Sybase) The select statement contains a COMPUTE clause that is not supported.
005539	Could not connect to datasource specified in -db variable: '`01'.	(ODBC) Could not connect to the specified datasource.
005540	Not connected to a database, database access is not allowed.	The SQR program is no longer connected to a database. Commands that access the database can no longer be used. This situation can occur if the CONNECT fails and the ON-ERROR option was used.
005543	Specify the Oracle DLL name in the sqr.ini file in [Environment:Oracle] section for ORACLE_DLL entry, such as ORACLE_DLL=orant 71.dll	(Oracle) SQR was unable to load the Oracle DLL. By default, SQR looks first for ociw32.dll or the DLL specified by the ORACLE_DLL entry in the [Environment:Oracle] section of the sqr.ini file. If that DLL could not be loaded, then SQR attempts to load orant71.dll.
005600	GETWRD: Word too long; maximum is `01.	Reduce the length of the "word". `01 = Maximum size of a "word" supported by this version of SQR
005700	Cannot call SQR recursively.	SQR cannot be called recursively. This error can only occur if a User Function from either UFUNC.C or UCALL.C calls the sqr() routine. Do not call sqr() from a UFUNC.C or UCALL.C routine.

Error Number	*Error Message*	*Suggestion/Interpretation*
005701	Too many SQR command line arguments; maximum is `01	To pass more than this number of arguments, use a @file argument file containing one argument per line. `01 = Maximum number supported by this version of SQR.
005702	Log file name specified is too long.	Reduce the length of the log file name.
005703	Error opening the SQR log file: ''`01' (`02): `03	`01 = Name of the file `02 = System error code `03 = System error message
005704	Missing program name.	The name of the program file was not found on the command line. The program name must be the first parameter on the command line.
005705	Program file name specified is too long.	Reduce the length of the program file name.
005707	Error opening the -E error file: ''`01' (`02): `03	`01 = Name of the file `02 = System error code `03 = System error message
005708	Cannot find `01 in SQRDIR, PATH or \SQR.	The specified file cannot be located in any of the directories pointed to by the mentioned environment variables or default directories. Make sure the "file" is present in one of the locations searched. `01 = File name
005709	`01 environment variable is not defined.	As of version 2.5, the environment variable SQRDIR must be defined. `01 = Name of the environment variable

Error Number	*Error Message*	*Suggestion/Interpretation*
005710	`01 path too long.	The length of the directory path plus the length of the file name to be opened is too long for SQR to handle. Reduce the length of the directory path. `01 = Environment variable name
005711	Bad number in -T test flag.	The number specified must be > zero. Correct the value.
005716	Unknown flag on command line: `01	Correct the syntax. `01 = Unknown command line flag
005720	Error opening tty. (`01): `02	(Tru64, UNIX/Linux) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message
005721	Error with 'ioctl()'. (`01): `02	(Tru64, UNIX/Linux) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message
005722	Error reading tty. (`01): `02	(Tru64, UNIX/Linux) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message
005723	Error closing tty. (`01): `02	(Tru64, UNIX/Linux) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
005724	Bad number in -B flag.	(Oracle, Sybase) The number specified must be greater than zero. Correct the value.
005734	No program name given.	The report name must be the first command line argument.
005737	Unknown printer type specified with - PRINTER: switch.	The printer type can be EH, HT, LP, HP, PS, or WP. WP is valid only with PC/Windows.
005738	Database name needs to be included with - DB switch.	(ODBC) Could not connect to the specified datasource.
005738	Database name needs to be included with - DB switch.	(Sybase) Supply the database name.
005739	Too many -F switches; maximum is `01.	Reduce the number of -F switches. `01 = Maximum number allowed
005740	-F and outfile name are required with DDN or DD style SQR {program} name.	(MVS) Correct the JCL stream.
005741	Attempting to use SQR {program} file for outfile.	(MVS) Correct the JCL stream.
005742	Attempt to invoke viewer (using WinExec) failed; error code = `01.	(Windows) `01 = System error code
005743	Unknown numeric type specified with - DNT: switch.	Correct the command line.
005744	-DNT:Decimal precision (`01) is out of range (`02 - `03).	Correct the command line. `01 = Specified precision `02 = Minimum allowed `03 = Maximum allowed

Error Number	*Error Message*	*Suggestion/Interpretation*
005745	The specified default numeric type '`01 = `02' is invalid.	Correct the sqr.ini file entry. `01 = Entry `02 = Value
005746	The decimal precision '`01 = `02' is out of range (`03 - `04).	Correct the sqr.ini file entry. `01 = Entry `02 = Value `03 = Minimum allowed `04 = Maximum allowed
005747	The following error(s) occurred while processing the [`01] section from the sqr.ini file.	See the error message(s) that follow. `01 = Name of the section
005750	The -Burst switch is not properly formatted.	The "Burst" command line flag is not properly formatted.
005751	The -Burst switch cannot be used with the -NOLIS switch.	The "Burst" command line flag cannot be specified when the -NOLIS command line flag is also specified.
005752	The -Burst switch requires either the - Printer:HT or - Printer:EH switch to be specified.	The "Burst" command line flag is applicable only when HTML code is produced. You must specify either the -PRINTER:HT or -PRINTER:EH switch.
005753	The -Burst:S and - Burst:T switches can only be used against an SPF file which was generated with SQR v4.1 and above.	The "Burst" command line flag can only be specified when processing a SPF file that was generated by SQR v4.1 and above. Older SPF files do not contain the proper information that permits bursting.
005754	The -Burst switch caused no output to be generated.	The "Burst" command line flag was specified with a set of page ranges that prevented any output to be created. Change the page ranges.
005900	Bad number in -`01	(Windows) Specify a valid number. `01 = Command line option

Error Number	*Error Message*	*Suggestion/Interpretation*
005901	Bad filename in -`01	(Windows) Specify a valid file name. `01 = Command line option
005902	Bad directory in -`01	(Windows) Specify a valid directory path. `01 = Command line option
005903	Cannot access the @ parameter file (`01): `02	(Windows) Depends on the system error message. `01 = System error code `02 = System error message
005904	The argument list is too long; maximum is `01.	(Windows) To pass more than this number of arguments, use a @file argument file containing one argument per line. `01 = Maximum number supported by this version of SQR.
005905	Cannot open the report file (`01): `02	(Windows) Depends on the system error message. `01 = System error code `02 = System error message
005906	Invalid filename entered.	(Windows) Re-enter with a valid file name.
006000	Error writing the printer file. (`01): `02	This is an error that can occur during normal operations due to the system environment (for example, file locking and permissions). Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
006001	Error reading the printer file. (`01): `02	This is an error that can occur during normal operations due to the system environment (for example, file locking and permissions). Record the steps leading up to the error and contact your system administrator. `01 = System error code `02 = System error message
006002	Cannot open the printer file: `01 (`02): `03	This is an error that can occur during normal operations due to the system environment (for example, file locking and permissions). Record the steps leading up to the error and contact your system administrator. `01 = Name of the file `02 = System error code `03 = System error message
006003	Unexpected End-Of- File while processing the printer file.	The file might be corrupted. Try to recreate the .spf file. If the error persists, contact technical support.
006004	Encountered unknown SPF code (`01) while reading the printer file.	The file might be corrupted. Try to recreate the .spf file. If the error persists, contact technical support. `01 = Unknown SPF code
006100	Duplicate chart (`01).	Each chart must be given a unique name. `01 = Chart name
006101	Unknown chart (`01).	Chart could not be found. `01 = Chart name
006102	Number of chart data- array columns specified (`01) exceeds the number of array columns (`02).	Correct the source code. `01 = Number of data-array columns `02 = Number of array columns

Error Number	*Error Message*	*Suggestion/Interpretation*
006103	Number of chart data- array rows specified (`01) exceeds the number of array rows (`02).	Correct the source code. `01 = Number of data-array rows `02 = Number of array rows
006104	Too many pie segments (`01). Max is `02.	Correct the source code. `01 = Number of segments `02 = Maximum allowed segments
006105	Chart module is not initialized.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006106	XY charts may have only numeric columns.	Correct the syntax.
006107	The 3rd column in the data array must be a character column to specify USE-3RD- DATA-COLUMN.	Correct the syntax.
006108	Invalid chart size or placement.	Correct the source code.
006120	INTERNAL: Bad chart index from stack (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Chart index
006121	INTERNAL: Unknown SQR BG Interface message (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Message code
006122	INTERNAL: Unsupported Grafsman chart type (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Chart type

Error Number	*Error Message*	*Suggestion/Interpretation*
006123	INTERNAL: Unsupported pie-explode setting (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Setting value
006124	INTERNAL: Unsupported tick-mark placement (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Placement value
006125	Grafsman interface message (`01) not supported.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Message code
006126	Unrecognized return code (`01) from Grafsman command message (`02).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Return code `02 = Message code
006127	Cannot fit Chart/Image into the current page. Position: (`01, `02) Size: (`03, `04)	Correct the source code. SQR ends the program run. `01 = Row `02 = Column `03 = Width `04 = Depth
006128	Check coordinate values.	Correct the syntax.
006140	Duplicate image (`01).	Images must be given unique names. `01 = Image name
006141	Unknown image (`01).	Image name could not be found. `01 = Image name

Error Number	*Error Message*	*Suggestion/Interpretation*
006142	Cannot open image file (`01). (`02): `03	`01 = Name of the file `02 = System error code `03 = System error message
006150	INTERNAL: Bad image index from stack (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Image name
006200	This report has already been defined.	Each report must be given a unique name.
006201	This layout has already been defined.	Each layout must be given a unique name.
006202	This printer has already been defined.	Each printer must be given a unique name.
006203	The values for '01' must be > 0.	Correct the syntax. `01 = Qualifier name
006204	Qualifiers '01' and '02' are mutually exclusive.	Correct the syntax. `01 = Qualifier name `02 = Qualifier name
006205	Qualifier '01' is not applicable with a 'default' printer.	Correct the syntax. `01 = Qualifier name
006206	The list must contain report names or ALL.	Correct the syntax.
006207	'ALL' must be specified by itself.	Correct the syntax.
006208	No report name was specified.	Correct the syntax.
006209	No layout name was specified.	Correct the syntax.

Error Number	*Error Message*	*Suggestion/Interpretation*
006210	No printer name was specified.	Correct the syntax.
006211	The name cannot be 'ALL'.	Correct the syntax.
006212	The name can only contain characters [0-9 A-Z _ -].	Correct the syntax.
006213	Report '01' is referenced by multiple '`02' printers.	Correct the syntax. `01 = Report name `02 = Printer type
006214	Qualifier '01' is not allowed with a '02' printer.	Correct the syntax. `01 = Qualifier name `02 = Printer type
006215	The value for '01' must be '02 0.	Correct the syntax. `01 = Qualifier name `02 = Relation to zero (<,<=,=,>=,>)
006216	Report '01' does not exist.	Correct the syntax. `01 = Report name
006217	The report name can be a string literal, variable, or column.	Correct the syntax. `01 = Report name
006218	Referenced layouts not defined:	A list of undefined layouts follows this message.
006219	Referenced reports not defined:	A list of undefined reports follows this message.
006220	Referenced printers not defined:	A list of undefined printers follows this message.

Error Number	*Error Message*	*Suggestion/Interpretation*
006221	The following SQR commands (listed below) cannot be used when any of the following NEW SQR commands are also used in the same report:	Correct the syntax.
006224	No printer type was specified.	Correct the syntax.
006225	Incorrect value for printer type. Valid values are:	Correct the syntax. A list of valid printer types follows this message.
006226	Attempt to execute the `01 command while processing the `02 procedure.	SQR ends the program run. `01 = SQR command `02 = Procedure name
006227	Incorrect value for 'paper-size'. Specify the actual dimensions or one of the following names:	Correct the syntax. A list of valid predefined paper-size names follows this message.
006228	Referenced TOC (Table Of Contents) not defined:	A list of undefined Table of Contents follows this message.
006229	This TOC (Table Of Contents) has already been defined.	Each Table of Contents must be given a unique name.
006230	The list must contain TOC (Table of Contents) names or ALL.	Correct the syntax.
006231	The TOC (Table Of Contents) entry cannot be positioned given the LEVEL (`01) and INDENTATION (`02) values.	The Table of Contents entry will not fit given the specified level and current indentation values. `01 = Specified LEVEL= value `02 = Current INDENTATION= value
006232	`01 command not allowed while generating the Table of Contents.	The specified command cannot be used while the Table of Contents is being generated. `01 = SQR command

Error Number	*Error Message*	*Suggestion/Interpretation*
006233	The TOC (Table of Contents) entry "A" cannot be processed because the existing entry "B" is positioned below it. A: Line = `01, Level = `02, Text = '`03' B: Line = `04, Level = `05, Text = '`06'	Correct the program logic to eliminate the conflict between the two TOC (Table of Contents) entries. `01 = A: Line number `02 = A: Level value `03 = A: Text value `04 = B: Line number `05 = B: Level value `06 = B: Text value
006300	Unknown parameter (`01).	Correct the syntax. `01 = Parameter name
006301	Value not valid for parameter (`01).	Correct the syntax. `01 = Parameter name
006302	Invalid option (`02) for parameter (`01).	Correct the syntax. `01 = Parameter name `02 = Option
006303	Parameter (`01) is required, but has not been specified.	Correct the syntax. `01 = Parameter name
006304	Parameter (`01) already specified.	Correct the syntax. `01 = Parameter name
006305	Parameter (`01) does not support &columns.	Correct the syntax. `01 = Parameter name
006306	Parameter (`01) requires equal sign.	Correct the syntax. `01 = Parameter name
006307	Parameter (`01) has an unquoted string.	Correct the syntax. `01 = Parameter name

Error Number	*Error Message*	*Suggestion/Interpretation*
006308	Missing part of specification for parameter (`01).	Correct the syntax. `01 = Parameter name
006309	Parameter (`01) requires literal.	Correct the syntax. `01 = Parameter name
006310	Parameter (`01) requires valid numeric value.	Correct the syntax. `01 = Parameter name
006311	Parameter (`01) requires integer value.	Correct the syntax. `01 = Parameter name
006312	Parameter (`01) does not support type supplied.	Correct the syntax. `01 = Parameter name
006313	Parameter (`01) requires valid string. Perhaps quote or $ is missing.	Correct the syntax. `01 = Parameter name
006314	Parameter (`01) does not accept 'NONE' in this context.	Correct the syntax. `01 = Parameter name
006315	Parameter (`01) requires proper object name.	Correct the syntax. `01 = Parameter name
006316	Parameter (`01) requires array name.	Correct the syntax. `01 = Parameter name
006317	Parameter (`01) does not accept 'AUTOSCALE' in this context.	Correct the syntax. `01 = Parameter name
006318	Parameter (`01) has improper value list.	Correct the syntax. `01 = Parameter name
006320	Parameter (`01) does not support relative values.	Correct the syntax. `01 = Parameter name

Error Number	*Error Message*	*Suggestion/Interpretation*
006350	Conversion [(`01) to (`02)] is not supported.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = From type `02 = To type
006352	INTERNAL: Unsupported option/request (`01) in (`02).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Option/request code `02 = Function name
006354	INTERNAL: Unknown data type, (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Data type
006355	INTERNAL: Unable to retrieve parameter value, (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Parameter name
006356	INTERNAL: Data type (`02) not valid for parameter (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Parameter name `02 = Data type
006357	INTERNAL: Data location (`02) not valid for data type (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Data location `02 = Data type

Error Number	*Error Message*	*Suggestion/Interpretation*
006358	INTERNAL: Cannot decode string (`01) to index.	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = String to decode
006359	INTERNAL: Cannot set bit value (`02) for parameter (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = Parameter name `02 = Value
006360	INTERNAL: Unknown program state (`01).	This is an internal error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = State
006400	Unsupported background color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006401	Unsupported border color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006402	Border width out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006403	X position out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006404	Y position out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006405	X size out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006406	Y size out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006407	Unsupported font.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006408	Unsupported font style.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006409	Unsupported font color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006410	Unsupported horizontal text justification value.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006411	Unsupported vertical text justification value.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006412	Unsupported font path.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006413	Unsupported font rotation.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006414	Font size out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006415	Text line id# out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006416	Unsupported chart type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006417	Unsupported chart sub-type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006418	Unsupported chart orientation (not H or V).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006419	Unsupported perspective (not 2D or 3D).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006420	Unsupported axis (not X or Y).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006421	Unsupported axis label data type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006422	Dataset id# out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006423	Unsupported dataset type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006424	Unsupported dataset color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006425	Unsupported dataset line style.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006426	Unsupported dataset fill pattern.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006427	Unsupported dataset marker.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006428	Chart type does not support Y-axis datasets.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006429	Pie-chart segment id# is out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006430	Unsupported pie-segment color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006431	Unsupported pie-segment border color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006432	Unsupported pie-segment pattern.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006433	Unsupported pie- segment explode setting.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006434	Command only valid for charts of type 'pie'.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006435	Pie-chart radius out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006436	Pie-chart starting angle out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006437	Unsupported pie-chart fill direction. Must be clockwise or counter-clockwise.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006438	Unsupported pie- segment label position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006439	Unsupported pie- segment quantity display position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006440	Unsupported pie- segment per-cent display position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006441	Unsupported legend style.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006442	Unsupported legend horizontal position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006443	Unsupported legend vertical position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006444	Text charts do not support legend.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006445	Number of datasets specified does not match data.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006446	Unsupported axis label position.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006447	Unsupported axis type (not LINEAR or LOG).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006448	Pie and text charts do not support axis control.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006449, 006450	Unsupported axis min scaling.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006451	Unsupported axis max scaling.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006452	Beginning of tickmarks is after end.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006453	Unsupported tickmark type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006454	Unsupported grid type.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006455	Unsupported grid color.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006456	Grid line width out of range.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006457	Unable to open grafcap file.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006458	Unsupported grafcap device.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006459	Error in grafcap entry specification.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006460	Unable to open chart output destination.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006461	Internal error during ggDraw.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006462	Improper parameters passed to gscale.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006463	The shared library specified in the grafcap file could not be found.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006464	A function called from the shared library specified in the grafcap file could not be found.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006500	The bar code could not be positioned on the page. Row: `01, Column: `02, Height: `03	Correct the source code. `01 = Row `02 = Column `03 = Height
006501	Unknown BCL error (`01) encountered.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support. `01 = BCL error code
006502	Invalid bar code type (`01): Valid values are from 1 to 15.	Correct the source code. `01 = Bar code type.
006503	The length of the bar code text '01' must be between 1 and 30 characters.	Correct the source code. `01 = Bar code text

Error Number	*Error Message*	*Suggestion/Interpretation*
006504	The length of the caption text '01' must be between 1 and 30 characters.	Correct the source code. `01 = Caption text
006505	Invalid printer type (`01): Valid values are from 0 to 13.	Correct the source code. `01 = Printer type
006506	Invalid offset: Valid values are from 0 to 250.	Correct the source code.
006507	Invalid height (`01): Valid values are from 0.1 to 2.0 inches.	Correct the source code. `01 = Height
006508	Invalid checksum: Valid values are from 0 to 2.	Correct the source code.
006509	Invalid pass: Valid values are from 1 to 6.	Correct the source code.
006510	The bar code text '01' is not valid for the type of bar code (`02) selected.	Correct the source code. `01 = Bar code text `02 = Bar code type
006511	Internal error: Could not generate the bar code.	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006512	Internal error: Bar code buffer required too large (>32K).	This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.
006601	Cannot allocate the device context for the default printer.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.

Error Number	*Error Message*	*Suggestion/Interpretation*
006602	Failed to start printing the document.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator.
006603	New-page (start) failed on page `01.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator. `01 = Page number
006604	New-page (end) failed on page `01.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator. `01 = Page number
006605	End document failed.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator.
006606	Error reading font information from the [Fonts] section in sqr.ini. Using the default font.	(Windows) Correct the [Fonts] section in the sqr.ini file.
006607	Failed to create a brush for shading.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006608	Failed to select font `01.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator. `01 = Font name

Error Number	*Error Message*	*Suggestion/Interpretation*
006609	Failed to modify font `01.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator. `01 = Font name
006610	Failed to create a pen that was required to draw a box.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006611	Failed to create a pen that was required to draw a horizontal line.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006612	Failed to create a pen that was required to draw a vertical line.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006613	Failed to open the image bitmap file (`01). (`02): `03	(Windows) This is an error that can occur during normal operations due to the system environment (file locking, permissions). Record the steps leading up to the error and contact your system administrator. `01 = Name of the file `02 = System error code `03 = System error message
006614	The file (`01) does not contain a valid bitmap.	(Windows) Specify a valid bitmap file. `01 = Name of the file
006615	Failed to create the palette for image (`01).	(Windows) This is an error that can occur due to lack of system resources or an invalid bitmap. Record the steps leading up to the error and contact your system administrator. `01 = Name of the file

Error Number	*Error Message*	*Suggestion/Interpretation*
006616	Failed to load RLE into memory for image (`01).	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator. `01 = Name of the file
006617	Failed to convert DIB to DDB for image (`01).	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator. `01 = Name of the file
006618	Failed to draw the bitmap image (`01).	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator. `01 = Name of the file
006619	Cannot access the default printer's driver.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator.
006620	Cannot select the charting clip area onto the printers DC.	(Windows) This is an error that can occur due to lack of system resources or a problem with the printer. Record the steps leading up to the error and contact your system administrator.
006621	Cannot select create a metafile required for business graphics.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006622	Cannot create a region required for business graphics.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.

Error Number	*Error Message*	*Suggestion/Interpretation*
006623	Cannot create a DC required for business graphics.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006624	Cannot create a bitmap required for business graphics.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006625	Business graphics failed while setting up the device (ggWinDevice).	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006626	Cannot draw business graphics.	(Windows) This is an error that can occur due to lack of system resources or due to a damaged LIBSTI.INI file. The LIBSTI.INI file resides in the Windows main directory. Make sure that the GPATH= and IPT= entries point to a valid SQR BINW directory. Record the steps leading up to the error and contact your system administrator.
006700	SQRDIR is not defined.	(Windows) The variable SQRDIR must be defined in the sqr.ini file.
006701	Could not allocate memory while attempting to register the .spf filename extension.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006702	Could not allocate memory for the page cache.	(Windows) This is an error that should never occur during normal operations. Record the steps leading up to the error and contact technical support.

Error Number	*Error Message*	*Suggestion/Interpretation*
006704	Cannot open or read file (`01) (`02): `03	(Windows) This is an error that can occur during normal operations due to the system environment (for example, file locking and permissions). Record the steps leading up to the error and contact your system administrator. `01 = Name of the file `02 = System error code `03 = System error message
006705	File (`01) is not in SPF packet format.	(Windows) The file was not produced by SQR or has been corrupted. `01 = Name of the file
006706	Failed to identify the start of the report (`01).	(Windows) The file was not produced by SQR or has been corrupted. `01 = Name of the file
006707	An invalid seek was made for page `01.	(Windows) This is an internal error which should not occur under normal operations. Contact technical support. `01 = Page number
006708	Too many errors were encountered while processing the file. Processing has been stopped.	(Windows) This is an error that can occur due to lack of system resources. Record the steps leading up to the error and contact your system administrator.
006709	Failed to open the image bitmap file (`01). (`02): `03 This message is displayed only once per SPF file.	(Windows) This is an error that can occur during normal operations due to the system environment (for example, file locking and permissions). Record the steps leading up to the error and contact your system administrator. `01 = Name of the file `02 = System error code `03 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
006800	`01: Detected internal program error.	This is an internal error that should never occur during normal operation. Record the steps leading up to the error and contact technical support. `01 = Name of the routine
006801	`01: Null Operand Passed as input.	This is an internal error that should never occur during normal operation. Record the steps leading up to the error and contact technical support. `01 = Name of the routine
006802	`01: Decimal Exponent Under/Overflow.	Exponent Under/Overflow: Exponent of decimal number has exceeded the valid boundaries established for the decimal type. Review the documentation for the current upper and lower bounds of a decimal object. `01 = Name of the routine
006803	`01: Decimal to Integer Conversion Under/Overflow.	Integer Under/Overflow: Cannot convert input decimal object into a valid integer number. Decimal object exceeds the established integer boundaries for this machine architecture. Review the magnitude and sign of the decimal object to ensure that it falls within the upper and lower bounds of an integer number. `01 = Name of the routine
006804	`01: Decimal to Float Conversion Under/Overflow.	Floating Point Under/Overflow: Cannot convert input decimal object into a valid floating point number. The decimal object exceeds the established floating point boundaries for this machine architecture. Review the magnitude and sign of the decimal object to ensure that it falls within the upper and lower bounds of a floating point number. `01 = Name of the routine

Error Number	*Error Message*	*Suggestion/Interpretation*
006805	`01: Decimal Precision Under/Overflow.	Decimal Precision Under/Overflow: Attempt made to initialize decimal object with an invalid precision. Verify the input precision value against the documented upper and lower boundaries for a decimal object. `01 = Name of the routine
006806	`01: String to Decimal Object Conversion Error.	String To Decimal Conversion Error: The length of input string is greater than the precision of underlying decimal object. Either increase the precision of the decimal object or reduce the size of the input mantissa to match the decimal object precision. `01 = Name of the routine
006807	`01: Truncation/Rounding Error - Outside Valid Range for Decimal Object.	Truncation/Rounding Error: Input truncation or round value is outside the valid range for this decimal object. Ensure that the truncation/round value is greater than or equal to zero and less than the precision of the underlying decimal object. `01 = Name of the routine
006808	`01: Decimal Error: Cannot Divide by Zero.	Decimal Math Divide by Zero Error: Attempt made to divide a decimal object by zero. Ensure that the divisor does not equal zero before attempting to divide. `01 = Name of the routine
006900	There is no default printer set up on your system. Use the Control Panel "Printers" applet to define it.	(Windows) SQR Print requires that a default printer be defined. Use the "Printers" applet in the Control Panel to define one.
007000	The locale '`01' is not defined in the sqr.ini file.	Verify the spelling of the locale name or the sqr.ini file. `01 = Locale name
007001	At least one qualifier must be specified.	Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
007002	The value for '01' must be a list of 02 string literals, variables or columns.	Correct the source code. `01 = Qualifier `02 = Number of entities in list
007003	The values for '01' and '02' cannot be the same.	Correct the source code. `01 = Qualifier `02 = Qualifier
007004	The value for '01' (`02) must be a single character which is not in the list: "03".	Correct the source code. `01 = Qualifier `02 = Value `03 = List of invalid characters
007005	The value for '`01' (`02) is invalid. Valid values are:	Correct the source code. `01 = Qualifier `02 = Value
007006	The last character of the '`01' value (`02) cannot be a digit or the minus sign or the same as either of the separators.	Correct the source code. `01 = Qualifier `02 = Invalid character
007007	The first character of the '`01' value (`02) cannot be a digit or the minus sign or the same as either of the separators.	Correct the source code. `01 = Qualifier `02 = Invalid character
00700	The following errors occurred while processing the (`01) locale from the sqr.ini file.	This message precedes error messages encountered while processing the sqr.ini file. `01 = Locale name
007009	The value for '`01' cannot be 'DEFAULT' or 'SYSTEM'.	Correct the syntax. `01 = Qualifier
007010	The value for '`01' (`02) is not properly formatted: Did not find the '>' for the '<nnn>' construct.	Correct the syntax. `01 = Qualifier `02 = Value

Error Number	*Error Message*	*Suggestion/Interpretation*
007011	The value for '`01' (`02) is not properly formatted: The value of an '<nnn>' construct must be from 1 to 255.	Correct the syntax. `01 = Qualifier `02 = Value
007012	The default locale (`01) specified in the [`02] section of the sqr.ini file has not been defined.	Correct the syntax. `01 = Locale name `02 = Section name
007013	The value for '`01' (`02) must be a list of `03 quoted string literals.	Correct the syntax. `01 = Qualifier `02 = Value `03 = Number of entities in list
007014	The entry (`01 = `02) is not valid.	Correct the sqr.ini entry. `01 = Qualifier from the sqr.ini file `02 = Qualifier's value
007100	The use of an edit mask or the keywords NUMBER, MONEY or DATE is not legal when storing numeric variables.	Correct the source code.
007101	The last keyword is not '`01'.	Correct the source code. `01 = Keyword
007102	Incompatible source and destination variable types.	Correct the source code.
007103	The keyword (`01) is not compatible with the variable (`02).	Correct the source code. `01 = Keyword `02 = Variable name
007104	The use of an edit mask or the keyword DATE is not legal if both variables are date variables.	Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
007200	The specified precision (`01) is out of range (`02 - `03).	Correct the source code. `01 = Specified precision `02 = Minimum precision `03 = Maximum precision
007201	The precision is specified by a value from `01 to `02 surrounded by parentheses.	Correct the source code. `01 = Minimum precision `02 = Maximum precision
007202	Variable (`01) is not a decimal variable and cannot have a precision associated with it.	Correct the source code. `01 = Variable name
007203	A string variable name is required here.	Correct the source code.
007204	A numeric variable name is required here.	Correct the source code.
007205	The variable (`01) has already been defined as '`02' and may not be redefined.	Correct the source code. `01 = Variable name `02 = Variable type
007206	The variable type has not been specified.	Correct the source code.
007207	This command is only allowed within local procedures.	Correct the source code.
007208	This command must be before all other commands in the procedure.	Correct the source code.
007209	Only string ($) and numeric (#) variables may be declared.	Correct the source code.
007210	Invalid variable name specified.	Correct the source code.
007211	You cannot declare a global variable from within a procedure.	Correct the source code.

Error Number	*Error Message*	*Suggestion/Interpretation*
007400	The specified character is invalid in the current character set.	Correct the program logic.
007401	'`01' is not a valid value for the ENCODING environment variable.	The specified encoding scheme is not known by SQR. `01 = ENCODING environment variable setting.
007402	The Double-Byte LET function '`01' is not supported in this version of SQR.	The SQT file contains a reference to a LET function, which is not supported by this version of SQR. `01 = LET function name
007403	The Double-Byte SQR command '`01' is not supported in this version of SQR.	The SQT file contains a reference to an SQR command, which is not supported by this version of SQR. `01 = SQR command name
007404	Double-Byte .sqt files are not supported by this version of SQR.	The runtime file was created by the double-byte version of SQR and is incompatible with the current version.
007405	The barcode text '`01' cannot contain double- byte characters.	Correct the source code. `01 = Bar code text
007501	Using `01 edit mask from (`02) against (`03)	A date edit mask element, which could cause date data to be incorrectly interpreted, was detected. This warning message can be turned off by setting the "OutputTwoDigitYearWarningMsg " entry in the [Default- Settings] section of the sqr.ini file to FALSE. `01 = Edit mask element `02 = Edit mask being used `03 = Value being applied to the edit mask
007601	Cannot access the Java file (`01) (`02): `03	SQR cannot access the required file. `01 = Name of the file `02 = System error code `03 = System error message

Error Number	*Error Message*	*Suggestion/Interpretation*
007602	-EH_Scale: value (`01) is out of range (`02 - `03).	Correct the command line. `01 = Specified scale `02 = Minimum allowed `03 = Maximum allowed
007603	-Printer:EH functionality is not available on this platform.	Enhanced HTML functionality is not available on this platform.
007604	-Printer:PD functionality is not available on this platform.	PDF functionality is not available on this platform.
007701	Did not find end of paragraph: `01 (No 'end-execute' clause found.)	Correct the source code. `01 = BEGIN-command in question.
007702	Invalid entry for keyword, '`01=`02'	Correct the source code.
007703	May only specify either PROCEDURE=, or COMMAND=, or GETDATA=, exclusive.	Correct the source code.
007704	Must specify a SCHEMA.	Correct the source code.
007705	Must specify either a PROCEDURE, COMMAND, or GETDATA.	Correct the source code.
007706	CONNECTION '`01' not found. No such connection.	Correct the source code.
007707	The returned set of Procedure parameters (INOUT and OUT) (length = `01 items) did not include one or more of the specified items.	Stored procedure error.
007708	Encountered a parameter of type '`01'. Valid types are either IN, OUT, or INOUT. If no type is entered, the type defaults to IN.	Stored procedure error.

Error Number	*Error Message*	*Suggestion/Interpretation*
007709	The datasource failed to provide the expected return status value. Verify the query metadata.	Datasource error.
007710	The datasource failed to provide the expected number of elements in the return status list.	Datasource error.
007711	Failed to login to the requested datasource (Connection='`01', username='`02'). DETAILS: `03	Logon failed.
007712	The requested rowset (`01) was not available. Verify the query metadata.	Not enough row sets.
007713	Missing or invalid Registry.properties file. Verify that the CLASSPATH includes SQRDIR, that SQRDIR contains the folder with the Registry.properties file, and that the Registry.properties file is valid.	Incorrect environment setup.
007714	The datasource ('`01') does not support the requested capability ('`02'). Check the capabilities list for the datasource, located in the Properties folder.	Invalid query for datasource.
007715	Failed to start the Java Virtual Machine (JVM). Possible causes are: missing or invalid jdk files, incorrect CLASSPATH, or insufficient resources.	Incorrect environment setup.
007716	The current rowset (`01) contained no rows. Check the return status and/or metadata for the requested service to determine the cause.	No data.
007717	The query failed. DETAILS: `01	Query failed.
007718	Failure setting property '`01'. DETAILS: `02	Property-set failed.

Error Number	*Error Message*	*Suggestion/Interpretation*
007719	The value for keyword '`01' exceeds the maximum length of '02 characters.	Keyword value too long.
007720	A fatal error occurred while fetching against the current rowset: DETAILS: `01	A failure occurred during row fetch.
007721	Parameter `01 (`02) was passed to the PROCEDURE as data type `03; expected (`04) type `05. Verify the query metadata.	A failure occurred during row fetch.
007722	Invalid query parameter: Reason: `01	Bad procedure parameter.
007723	Too many parameters (= `01) were supplied to the query. Verify the query metadata.	Bad procedure parameter.
007724	Parameter `01 (`02) was passed to the PROCEDURE as type `03; expected type `04. Verify the query metadata.	Bad procedure parameter.
007725	Parameter `01 ('`02', JDO-type `03), specified 'NULL', is a required-parameter. Specify a value or variable name.	Bad procedure parameter.
007726	The list-variable parameter to the query is too long. Maximum number of elements is 30.	List too long.
007727	Unable to retrieve metadata for Procedure=`01, Schema=`02. DETAILS: `03	Metadata check failed.
007728	Parameter list type mismatch (#`01, SQR type = `02). The datasource expected a parameter of type `03. Verify the query metadata.	Parameter list mismatch.

Error Number	*Error Message*	*Suggestion/Interpretation*
007729	List size mismatch detected while fetching data of type ROW, `01 items, into SQR list- variable, `02 items. Fetching will proceed to the smaller size.	List size mismatch.
007730	Incorrect syntax for BEGIN-SELECT ... FROM. Options are: FROM ROWSETS=... FROM PARAMETER= $strvar | strlit	Bad BEGIN-SELECT syntax.
007731	Attempted to pass as INOUT or OUT a parameter which was of type ROWSET (`01). Use of such parameters is supported as IN only, after which they may be used in a BEGIN- SELECT construct.	Bad parameter keyword.
007732	Attempt to use a scalar SQR variable ('01') to reference a ROWSET procedure parameter ('02'). Use either the keyword 'NULL', or an SQR LIST variable (%var). Verify the query metadata.	Bad proc parameter.
007733	The list of keywords entered to the PARAMETERS keyword must be terminated with a semi-colon.	Bad proc parameter. Correct the source code.
007734	Datasource '01' not found. The Connection being used by this query specifies a data-source which is not listed in the DDO Registry ('02'). DETAILS: `03.	<obsolete>
007735	Missing one or more DDO {fname} .jar files. Verify the location of the original- installation files, and that they are accessible. Error code: `01. Classpath: `02.	<obsolete>
007736	Unable to open Connection ('01') to data-source ('02'). Possible causes: (a) the Declare-Connection specification is invalid, or (b) the datasource is no longer available. DETAILS: `03.	Bad environment.

Error Number	*Error Message*	*Suggestion/Interpretation*
007737	Unable to locate one or more entry points in an SQR {fname}.jar file. Verify that the original-installation files have not become corrupted.	Bad environment.
007738	At least one JNI method pointer was lost. This should never occur: record the steps leading up to this failure, and contact Technical Support. DETAILS: Schema='`01', Proc='`02'.	Bad environment.
007739	Unable to locate query object '`01' in the specified schema (`02). DETAILS: `03.	Bad environment.
007740	Invalid &pseudonym or 'TYPE=' data-type specified for a begin-select column-variable. Valid types are: CHAR, TEXT, DATE, NUMBER.	Correct the syntax.
007741	Illegal attempt to fetch a non-scalar field into a column variable. Correct the query.	Correct the syntax.
007742	The output parameter specified in 'Begin- Select ... From Parameter = `01' is not available. Available parameters: `02.	Bad command.
007743	The output parameter specified in 'Begin- Select ... From Parameter = `01' is not of type ROWSET. Verify the query metadata.	Bad command.
007744	Illegal attempt to assign an SQR variable ('01') of type '02' the value from a DDO object ('03') of type '04'. Verify the query metadata.	<obsolete>
007745	Illegal attempt to assign an SQR column variable ('01') of type '02' the value from a DDO object of type '03'. Verify the query metadata.	<obsolete>

Error Number	*Error Message*	*Suggestion/Interpretation*
007746	Failed to locate the requested Rowset (`01) while processing the query. The last available Rowset number is `02. Verify the query metadata.	Not enough row sets.
007747	The query raised a DDO exception. DETAILS: `01.	<obsolete>
007748	A BEGIN-SELECT paragraph was coded, but the query returned no Rows.	No data warning.
007749	Invalid syntax for PARAMETERS=(...) statement. Use: PARAMETERS=(%v | $v | #v | &v | NULL | SKIP | numlit | datelit | textlit [IN | INOUT], ...) All parameters must be specified. Optional parameters which are to be ignored may be specified by the keyword 'NULL' or 'SKIP'. Correct the syntax.	Incorrect syntax.
007750	FATAL: Failure creating Java object.	General failure.
007751	Attempt to create a List variable of size greater than the maximum size of `01 items.	General failure.
007752	Parameter-list item ''`01' is not a member of the parameter list for this Query. Verify the query metadata.	No such input/inout parameter.
007753	Attempt to access List- row (`01) beyond the List size (`02 rows).	Bad list assignment/setup.
007754	Attempt to assign/modify a List row is not compatible with the List definition.	Bad list assignment/setup.

Error Number	*Error Message*	*Suggestion/Interpretation*
007755	Attempt to assign a row to a non-existent List variable. Define the List first, using the syntax: let %lname[size] = list(NUMBER | DATE | TEXT #var | $var [, ...])	Bad list assignment/setup.
007756	Incorrect syntax for List-variable reference. Use: let [$ | #]var = %listname[nlit | #var].colname	Bad list assignment/setup.
007757	Alter-connection statement missing 'DSN=...'.	Improper alter-conn.
007758	List-definition size specifier must be literal.	Improper alter-conn.
007759	Attempt to access a non-existent List- column ('01').	No such list column name.
007760	Must specify one of the keywords, FROM- ROWSETS or FROM_PARAMETER .	Incorrect syntax for LOAD_LOOKUP.
007761	Incorrect syntax to Load-lookup 'PARAMETERS=' keyword. Use: PARAMETERS=(slit | nlit | $var | #var | %var | &var, ...) No line wrapping is allowed for this usage.	Incorrect syntax for LOAD_LOOKUP.
007762	Too many parameters (`02) entered to Load- Lookup command. Max parameters is `01.	Incorrect syntax for LOAD_LOOKUP.
007763	Problem executing the cursor for LOAD- LOOKUP table '`01'. DETAILS: `02.	The database server returned an error while trying to execute the SQL statement needed to process the LOAD-LOOKUP command. `01 = Load lookup table name
007764	Bad return fetching row from database in LOAD-LOOKUP table '`01'. DETAILS: `02	The database server returned an error while fetching the data. `01 = Load lookup table name

Error Number	*Error Message*	*Suggestion/Interpretation*
007765	DC, DI sort options not supported with this SQR version. To sort, use SORT=SC or SORT=SI.	Database sort not supported for LOAD_LOOKUP with DDO. <obsolete>
007766	Must specify a query keyword; PROCEDURE=, COMMAND= or GETDATA=.	Incorrect syntax for Load-lookup. Specify a keyword representing the query.
007767	Unknown column variable type.	Unknown data type returned by the server.
007768	The property `01` was not found in the property sheet for the specified datasource (`02). Available property names are: `03.The datasource property sheet does not include the named property.	Verify the metadata and correct the syntax.
007769	The specified CONNECTION ('01') references a datasource whose property sheet does not show support for the Get-Data query method.The datasource property sheet does not show support for Get-Data.	Verify the metadata and property sheet and correct the syntax.
007770	Attempt to create a Selector (or MDSelector) object failed. This event should not occur. Contact your system administrator.	Failed to create the requested object. Contact your system administrator.
009000	Error reading the font information from the [`01] section in sqr.ini. Font name `02 is too long. The maximum length allowed for a font name is `03.	Use a shorter font name in sqr.ini. `01 = Printing device specific font configuration section. `02 = Font name. `03 = Internal limit for font name.
009001	Error reading the font information from the [`01] section in sqr.ini. Font file path `02 is too long. The maximum length allowed for a font file path is `03.	Use a shorter font file path in sqr.ini. Relocate font file if necessary. `01 = Font configuration section name. `02 = Font file path. `03 = Internal limit for file path.

Error Number	*Error Message*	*Suggestion/Interpretation*
009002	Font `01 is not valid for `02 output. Please correct the font configuration in the [`02 Fonts] section of the sqr.ini file.	Use a valid font. Refer to the documentation for supported fonts. If you are using disk based fonts like TrueType, also ensure that the font type specific configuration section (like [TrueType Fonts]) is correctly configured. `01 = Font name. `02 = Selected output type.
009003	File name for `01 font `02 is not specified correctly. Please correct the configuration in the [`01 Fonts] section of sqr.ini file.	The font file path is not specified correctly. Correct the sqr.ini file. `01 = Font type. `02 = Font name.
009004	`01 font file `02 cannot be opened. Please check if the file exists, or correct the font configuration in the [`01 Fonts] section of the sqr.ini file.	Either the font file does not exist in the location specified in sqr.ini file, or the path specified in sqr.ini file is not correct. Correct the error and try again. `01 = Font type. `02 = Font file path.
009005	The directory ID specified for `01 in the [TrueType Fonts] section of the sqr.ini file exceeds the actual number of fonts included in the TrueType collection font file. `02 includes `03 fonts. Please specify a valid directory ID.	Correct the configuration in sqr.ini so that the directory ID is set correctly for the font. Note that the directory ID starts from 0. `01 = Font name. `02 = Font file path. `03 = Actual number of fonts included in TrueType collection.
009006	Font `01 is not a supported type of TrueType/OpenType font.	Currently, SQR does not support CFF based OpenType font. Remove the font from the configuration and try again.
009007	Font `01 is not a supported type of TrueType/OpenType font. The ''02' table is missing from the font file.	This TrueType/OpenType font does not have the table SQR needs for processing. Remove the font from the configuration and try again. `01 = Font name. `02 = Table name.

Error Number	*Error Message*	*Suggestion/Interpretation*
009008	Font `01 is not a supported type of TrueType/OpenType font. This font does not allow embedding in a document.	SQR requires TrueType/OpenType font with embedding feature allowed to include the subset of the font in the print output. This TrueType/OpenType font does not allow embedding. Remove the font from the configuration and try again.
009009	Font `01 is not a supported type of TrueType/OpenType font. This font does not have a supported type of character to glyph mapping (CMAP) table.	SQR requires a TrueType/OpenType font with CMAP table with Platform ID 3 (Microsoft), Encoding ID 0 (Symbol), 4 (UCS-2), or 10 (UCS-4) and table format 4 or 12. Other CMAP table types and encodings are currently not supported. Remove the font from the configuration and try again. `01 = Font name
009010	Error in the [`01:Exclusion Ranges] section in theINI file. Start character code must be greater than the end character code.	In the exclusion range section, the start character code must be smaller than the end character code. Review the section in INI file and correct error. You can specify the range in a decimal or hexadecimal number. If you use a hexadecimal number, you must prefix the number with '0x' or the number is recognized as a decimal.

Appendix B

Using SQR Sample Programs

This chapter discusses SQR sample programs.

SQR Samples Library

SQR Samples is a library of SQR programs that you can use to adapt and experiment with programs. These programs are stored in the <PS_HOME>\bin\sqr\<database_platform>\SAMPLE (SAMPLEW for Microsoft Windows) directory. You can modify these programs to create configured SQRs.

SQR Sample Programs

Each program comprises a report specification and a sample of the output. This table describes all of the sample SQR programs:

Name	*Description*
_____.DAT	Data files that are used by the loadall.sqr programs.
_____.MEM	SQR startup files for running small, medium, and large SQR programs.
APPEND.SQR	Demonstrates the APPEND and FIXED-NOLF (no line feed) commands.
APTDIARY.SQR	Demonstrates columns and text wrapping.
AREA100.SQR	Demonstrates a 100 percent area chart.
BAR100.SQR	Demonstrates a 100 percent bar chart.
BARCODE.SQR	Demonstrates printing a bar code.
CALENDAR.SQR	Demonstrates nondatabase formatting.

Name	*Description*
COMP_FOR.SQR	Prints a graph of the forecasted and actual sales for a given employee.
COMP_F_G.SQR	Prints a graph of the forecasted and actual sales for a month or quarter.
COMP_PLN.SQR	Prints a graph of the planned and actual sales for a given employee.
COMP_P_G.SQR	Prints a graph of the planned and actual sales for month or quarter.
COVLET02.SQR	Uses SQR to input data from a user, enter the data in the database, and write a form letter by using a DOCUMENT paragraph.
CUST.SQR	Prints a list of all of the customers, bursted by page.
CUSTLBLS.SQR	Demonstrates printing mailing labels within columns.
CUSTOMER.SQR	Demonstrates multiple detail lines and the NEXT-LISTING command.
CUSTOMR2.SQR	Demonstrates the use of the ON-BREAK argument to the PRINT command.
CUSTOMR3.SQR	Demonstrates the use of the INPUT command to change the report output.
CUSTOMR4.SQR	Demonstrates the use of arrays.
CUSTOMR5.SQR	Demonstrates dynamic queries to enable the user to qualify a report as it runs.
CUST_SUM.SQR	Prints a group of information about each customer in the customer table.
CUSTTAPE.SQR	Demonstrates the flat file output for magnetic tape or other postprocessing.
DATAA.DAT	Needed for append.sqr.

Name	*Description*
DATAB.DAT	Needed for append.sqr.
DROPALL.SQR	Drops all of the SQR sample tables that are created by the LOADALL program.
DROPPROC.SQR	(Sybase) Deletes leftover, temporary stored procedures belonging to the user.
DYNAMCOL.SQR	Demonstrates the use of dynamic columns, dynamic tables, and variables that are passed to the ON-ERROR procedure.
EMP.SQR	Prints a list of all of the employees, bursted by page.
EMP_COMM.SQR	Calculates each employee's commission, based on sales.
EMP_M_Q.SQR	Lists all employee quotas for a given month or quarter.
ENVELOPE.SQR	Demonstrates the use of printing envelope with proper bar codes.
EXPORT.SQR	Creates two SQR reports: one to export a database table and one to import that table. Data from the table is stored in an external operating system file in compressed format, with trailing blanks removed.
FLATFILE.SQR	Creates an SQR report to extract a database table and place it in a flat file.
FLOATBAR.SQR	Demonstrates a floating bar chart.
FOR_CUST.SQR	Creates a sales forecast for a given customer, grouped by month or quarter.
FOR_EMP.SQR	Creates a sales forecast for a given employee, grouped by month or quarter.
FOR_PROD.SQR	Creates a sales forecast for a given product, grouped by month or quarter.
FOR_REG.SQR	Creates a sales forecast for a given region, grouped by month or quarter.

Name	*Description*
FOR_SUM.SQR	Creates a table of projected product sales with links to more information.
FORMLETR.SQR	Demonstrates form letters by using a document paragraph.
HILO.SQR	Demonstrates a high-low-close chart.
HISTGRAM.SQR	Demonstrates a histogram chart.
INQUIRY.SQR	Creates an SQR program to display rows at your terminal that are selected from a database table that you specify. The resulting SQR program prompts you to qualify the rows to be selected, display those rows, then repeat.
INVOICE.SQR	Demonstrates creating multiple reports, printing invoices, and printing envelopes.
LOADALL.SQR	Creates and loads the sample tables that are used in the preceding SQR programs.
MAKEDATA.SQR	Creates a data file with a fixed length and NOLF attributes.
MAKEREPT.SQR	Helps you create SQR reports more quickly.
MULTIPLE.SQR	Demonstrates creating multiple reports.
NESTREPT.SQR	Demonstrates nesting procedures.
ORDERS.SQR	Lists all of the orders and the order lines that are associated with them.
ORD_MONG.SQR	Lists all orders for a given month and groups them by employee number.
ORD_M_Q.SQR	Lists all orders for a given month or quarter.
ORD_PROD.SQR	Lists all orders for a given product.
ORD_REGG.SQR	Creates a report of all orders from a given region, grouped by month or quarter.

Name	*Description*
ORD_SUM.SQR	Displays an summary of orders, grouped by month.
ORD_S_Q.SQR	Prints a graph of the percent of orders for each region (in a year) and four graphs of the percent of orders for each region (one for each quarter of that year).
OVERBAR.SQR	Demonstrates an overlapped bar chart.
PHONELST.SQR	Demonstrates printing within columns, page headings, and page footings.
PLN_EMP.SQR	Creates a sales plan for a given employee, grouped by month or quarter.
PLN_GEN.SQR	Creates a sales plan, grouped by month or quarter.
PLN_REG.SQR	Creates a sales plan for given region, grouped by month or quarter.
PRODUCT.SQR	Lists products and their prices and graphs orders of products.
SALELEAD.SQR	Demonstrates DOCUMENT paragraphs.
SALES.SQR	Demonstrates charting from stored data and printing several charts on one page.
SCATTER.SQR	Demonstrates a scatter chart.
SHOWPROC.SQR	(Sybase) Displays leftover, temporary stored procedures belonging to the user.
STCKAREA.SQR	Demonstrates a stacked area chart.
SQR3DBAR.SQR	Demonstrates a three-dimensional bar chart.
SQRLASER.SQR	Demonstrates graphic and file input/output commands.
SQRLINE.SQR	Demonstrates a line chart.

Name	*Description*
SQRLOGO.SQR	Demonstrates printing images.
SQRPIE.SQR	Demonstrates a pie chart.
TABREP.SQR	Creates a tabular SQR report for a table that you select.
UPDATE.SQR	Generates an SQR program that enables you to query and update database tables. The created program uses the SHOW command to simulate a menu interface.

Index