Table of Contents

PROCEDURE BUILDER REFERENCE ................................................................. 1
  USING INTERPRETER COMMANDS ......................................................... 1
  ALPHABETIC LIST OF COMMANDS ....................................................... 3
BIND VARIABLE COMMANDS ................................................................. 4
DATABASE COMMANDS ............................................................................... 4
DEBUG ACTION COMMANDS ........................................................................ 4
DEBUGGING COMMANDS ........................................................................... 5
LIBRARY COMMANDS ................................................................................ 5
LOAD PATH COMMANDS ............................................................................. 5
LOGGING COMMANDS ............................................................................... 6
PROGRAM UNIT COMMANDS ................................................................. 6
SESSION COMMANDS ................................................................................ 6
INTERPRETER COMMANDS ........................................................................ 6
BIND VARIABLE COMMANDS ....................................................................... 8
  CREATE (BIND VARIABLE) COMMAND (PB STANDALONE ONLY) .......... 8
  DELETE (BIND VARIABLES) COMMAND (PB STANDALONE ONLY) ..... 9
DATABASE COMMANDS ............................................................................... 10
  CONNECT COMMAND (PB STANDALONE ONLY) ..................................... 10
  DESCRIBE (TABLES AND VIEWS) COMMAND .................................... 10
DISCONNECT COMMAND (PB STANDALONE ONLY) .................................. 11
GRANT COMMAND (DATABASE COMMANDS) ........................................... 11
REVOKE COMMAND (DATABASE COMMANDS) .......................................... 11
STORE COMMAND ................................................................................... 12
DEBUG ACTION COMMANDS ...................................................................... 14
  BREAK COMMAND ................................................................................ 14
  DELETE (DEBUG ACTIONS) COMMAND ............................................... 16
  DESCRIBE (DEBUG ACTIONS) COMMAND ............................................ 16
  DISABLE (DEBUG ACTIONS) COMMAND ............................................. 17
  ENABLE (DEBUG ACTIONS) COMMAND .............................................. 17
  LIST (DEBUG ACTIONS) COMMAND .................................................. 18
  SHOW (DEBUG ACTIONS) COMMAND .................................................. 18
  TRIGGER COMMAND .......................................................................... 19
DEBUGGING COMMANDS .......................................................................... 22
  DESCRIBE (LOCALS) COMMAND ......................................................... 22
  GO COMMAND .................................................................................... 22
  RESET COMMAND .............................................................................. 23
  SET COMMAND ................................................................................... 23
  SHOW (CALL STACK) COMMAND ........................................................ 24
  STEP COMMAND .................................................................................. 24
LIBRARY COMMANDS ............................................................................... 27
  ATTACH COMMAND ............................................................................ 27
  CLOSE COMMAND ............................................................................... 28
  COMPIL E (LIBRARIES) COMMAND ..................................................... 28
  CREATE (LIBRARIES) COMMAND ...................................................... 29
  DELETE (LIBRARIES) COMMAND ....................................................... 29
  DELETE (LIBRARY PROGRAM UNITS) COMMAND .............................. 30
  DESCRIBE (LIBRARIES) COMMAND .................................................. 31
  DETACH COMMAND ............................................................................. 31
  EXPORT (LIBRARIES) COMMAND ..................................................... 31
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENERATE COMMAND</td>
<td>32</td>
</tr>
<tr>
<td>GRANT COMMAND (LIBRARY COMMANDS)</td>
<td>32</td>
</tr>
<tr>
<td>INSERT (LIBRARY PROGRAM UNITS) COMMAND</td>
<td>33</td>
</tr>
<tr>
<td>LOAD (LIBRARY PROGRAM UNITS) COMMAND</td>
<td>34</td>
</tr>
<tr>
<td>OPEN COMMAND</td>
<td>35</td>
</tr>
<tr>
<td>RENAME (LIBRARIES) COMMAND</td>
<td>35</td>
</tr>
<tr>
<td>REVERT COMMAND</td>
<td>36</td>
</tr>
<tr>
<td>REVOKE COMMAND (LIBRARY COMMANDS)</td>
<td>36</td>
</tr>
<tr>
<td>SAVE COMMAND</td>
<td>37</td>
</tr>
<tr>
<td>SHOW (LIBRARIES) COMMAND</td>
<td>37</td>
</tr>
<tr>
<td>GRANT COMMAND (LIBRARY COMMANDS)</td>
<td>32</td>
</tr>
<tr>
<td>INSERT (LIBRARY PROGRAM UNITS) COMMAND</td>
<td>33</td>
</tr>
<tr>
<td>LOAD (LIBRARY PROGRAM UNITS) COMMAND</td>
<td>34</td>
</tr>
<tr>
<td>OPEN COMMAND</td>
<td>35</td>
</tr>
<tr>
<td>RENAME (LIBRARIES) COMMAND</td>
<td>35</td>
</tr>
<tr>
<td>REVERT COMMAND</td>
<td>36</td>
</tr>
<tr>
<td>REVOKE COMMAND (LIBRARY COMMANDS)</td>
<td>36</td>
</tr>
<tr>
<td>SAVE COMMAND</td>
<td>37</td>
</tr>
<tr>
<td>SHOW (LIBRARIES) COMMAND</td>
<td>37</td>
</tr>
<tr>
<td>INTERPRETER COMMANDS</td>
<td>38</td>
</tr>
<tr>
<td>LOAD PATH COMMANDS</td>
<td>41</td>
</tr>
<tr>
<td>DELETE (LOAD PATH) COMMAND</td>
<td>41</td>
</tr>
<tr>
<td>DESCRIBE (LOAD PATH) COMMAND</td>
<td>41</td>
</tr>
<tr>
<td>INSERT (LOAD PATH) COMMAND</td>
<td>41</td>
</tr>
<tr>
<td>LOGGING COMMANDS</td>
<td>43</td>
</tr>
<tr>
<td>DISABLE (LOGGING) COMMAND</td>
<td>43</td>
</tr>
<tr>
<td>ENABLE (LOGGING) COMMAND</td>
<td>43</td>
</tr>
<tr>
<td>LOG COMMAND</td>
<td>43</td>
</tr>
<tr>
<td>PROGRAM UNIT COMMANDS</td>
<td>45</td>
</tr>
<tr>
<td>COMPIL (PROGRAM UNITS) COMMAND</td>
<td>45</td>
</tr>
<tr>
<td>DELETE (PROGRAM UNITS) COMMAND</td>
<td>45</td>
</tr>
<tr>
<td>DESCRIBE (PROGRAM UNITS) COMMAND</td>
<td>46</td>
</tr>
<tr>
<td>DISABLE (COMPILER OPTIONS) COMMAND</td>
<td>47</td>
</tr>
<tr>
<td>ENABLE (COMPILER OPTIONS) COMMAND</td>
<td>47</td>
</tr>
<tr>
<td>EXECUTE COMMAND (PB STANDALONE ONLY)</td>
<td>48</td>
</tr>
<tr>
<td>EXPORT (PROGRAM UNITS) COMMAND</td>
<td>48</td>
</tr>
<tr>
<td>EXPORT (STORED PROGRAM UNITS) COMMAND</td>
<td>49</td>
</tr>
<tr>
<td>LIST (PROGRAM UNITS) COMMAND</td>
<td>50</td>
</tr>
<tr>
<td>LOAD (PROGRAM UNITS) COMMAND</td>
<td>52</td>
</tr>
<tr>
<td>LOAD (STORED PROGRAM UNITS) COMMAND</td>
<td>52</td>
</tr>
<tr>
<td>SHOW (LOCALS) COMMAND</td>
<td>53</td>
</tr>
<tr>
<td>SHOW (PROGRAM UNITS) COMMAND</td>
<td>54</td>
</tr>
<tr>
<td>SESSION COMMANDS</td>
<td>55</td>
</tr>
<tr>
<td>DESCRIBE (VERSION) COMMAND</td>
<td>55</td>
</tr>
<tr>
<td>HELP COMMAND</td>
<td>55</td>
</tr>
<tr>
<td>INTERPRET COMMAND</td>
<td>55</td>
</tr>
<tr>
<td>QUIT COMMAND (PROCEDURE BUILDER STANDALONE ONLY)</td>
<td>56</td>
</tr>
</tbody>
</table>
Preface

Welcome to Release 6.0 of the Oracle Procedure Builder Reference.

This reference guide includes information to help you effectively work with Oracle Procedure Builder and contains detailed information about its commands.

This preface explains how this user’s guide is organized and introduces other sources of information that can help you use Oracle Procedure Builder.
Prerequisites

You should be familiar with your computer and its operating system. For example, you should know the commands for deleting and copying files and understand the concepts of search paths, subdirectories, and path names. Refer to your Microsoft Windows 95 or NT and DOS product documentation for more information.

You should also understand the fundamentals of Microsoft Windows, such as the elements of an application window. You should also be familiar with such programs as the Explorer, Taskbar or Task Manager, and Registry.

Notational Conventions

The following typographical conventions are used in this guide:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fixed-width font</td>
<td>Text in a fixed-width font indicates commands that you enter exactly as shown. Text typed on a PC is not case-sensitive unless otherwise noted. In commands, punctuation other than brackets and vertical bars must be entered exactly as shown.</td>
</tr>
<tr>
<td>lowercase</td>
<td>Lowercase characters in a command statement represent a variable. Substitute an appropriate value.</td>
</tr>
<tr>
<td>UPPERCASE</td>
<td>Uppercase characters within the text represent command names, SQL reserved words, and keywords.</td>
</tr>
<tr>
<td>boldface</td>
<td>Boldface is used to indicate user interface items such as menu choices and buttons.</td>
</tr>
<tr>
<td>C&gt;</td>
<td>C&gt; represents the DOS prompt. Your prompt may differ.</td>
</tr>
</tbody>
</table>

Related Publications

You may also wish to consult the following Oracle documentation:

<table>
<thead>
<tr>
<th>Title</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Developer: Guidelines for Building Applications</td>
<td>A58766</td>
</tr>
<tr>
<td>SQL*Plus User’s Guide and Reference Version 3.1</td>
<td>A24801</td>
</tr>
</tbody>
</table>
Using Interpreter commands

Commands adhere to the following general syntax:

```
.command-name [option...]
```

In other words, a command consists of a period (.), then the command name, followed by zero or more keywords and keyword value arguments.

Command options generally follow the form shown below:

```
keyword
```

or

```
keyword value(s)
```

Thus, an option consists of either a single keyword, or a keyword followed by one or more argument values. The command name, keywords, and argument values are separated by white space. Command names, keywords, and argument values are not case sensitive.

For example, the following DESCRIBE command invocation illustrates the basic elements of Procedure Builder command syntax:

```
.DESCRIBE PROCEDURE proc1 BODY
```

The command name DESCRIBE is followed by the PROCEDURE and BODY keywords. The PROCEDURE takes a single argument value, `proc1`, while the BODY keyword takes no argument values.

**Multi-valued Arguments**

Keyword arguments may be multi-valued, in which case the individual values are delimited by commas as shown below:

```
value, value...
```

Spaces may appear between the commas and neighboring values.

Keyword arguments that can be multi-valued according to the syntax specified above will be described as shown below:

```
name[, name...]
```

For example, the LOAD command has the following partial syntax:

```
.LOAD FILE name[, name...]
```

Thus, the file argument can be single-valued as shown below:

```
.LOAD FILE file1
```

or multi-valued as shown below:

```
.LOAD FILE file1, file2, file3
```

**Position Independence**

Unless explicitly specified in the syntax descriptions, keywords may appear in any order. For example, the command:

```
.DESCRIBE PROCEDURE proc1 BODY
```

can also be entered as:

```
.DESCRIBE BODY PROCEDURE proc1
```
Multi-line Commands

Normally, commands are terminated by a newline character or a carriage return. However, it is often desirable to make a command span multiple lines. This can be done by including the continuation character (backslash by default) as the last character of each line to be continued. For example, the continuation character is used below to place each file name argument value to the LOAD command on a separate line:

```
.LOAD FILE long_file_name_number_one, \
  long_file_name_number_two, \
  long_file_name_number_three
```

Argument Value Quoting

Non-numeric command argument values may be optionally enclosed in double quotes. The quotes serve only as delimiters and are not considered part of the argument value. This is particularly useful in specifying argument values that contain white space, commas, or wildcard characters. For example, if supported by the native operating system, a file name containing a space could be specified in a load command as follows:

```
.LOAD FILE "my file"
```

A double quote may be included as a part of the argument value by preceding it with another double quote. For example, the command

```
.LOAD FILE ""quoted file"
```

loads a file with a name containing two double quotes—one at the beginning and one at the end.

Abbreviating Keywords

A command keyword may be abbreviated by typing only as many characters as it takes to distinguish it from all other keywords accepted by the same command.

Command names may not be abbreviated. This is to minimize conflict with the PL/SQL namespace and avoid confusion in distinguishing between commands and PL/SQL code fragments.

Entering PL/SQL Code

In addition to commands, the Interpreter accepts and evaluates PL/SQL constructs (e.g., statements, blocks, procedure definitions, etc.), and SQL statements. Procedure Builder interprets a line beginning with anything other than a valid command name as the beginning of a PL/SQL statement, block, program unit, or SQL statement.

While commands occupy a single line (unless the continuation character is used), PL/SQL or SQL statements may occupy any number of lines, and continuation characters are neither necessary nor allowed. If necessary, a PL/SQL construct can always be distinguished from a command by enclosing it in the block delimiters BEGIN and END.

Notational Conventions

The following table describes the notation and conventions for command syntax used in this section.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>uppercase</td>
<td>BREAK</td>
<td>A command or keyword name; it need not be typed in uppercase</td>
</tr>
<tr>
<td>lowercase</td>
<td>numbers</td>
<td>A keyword value; substitute an appropriate value</td>
</tr>
<tr>
<td>italics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vertical bar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>braces</td>
<td>{STACK</td>
<td>SCOP E}</td>
</tr>
</tbody>
</table>
Enter other punctuation marks (such as commas) where shown in the command syntax.

Alphabetic list of commands

ATTACH
BREAK
CLOSE
COMPILE (libraries)
COMPILE (program units)
CONNECT
CREATE (bind variables)
CREATE (libraries)
DELETE (bind variables)
DELETE (debug actions)
DELETE (libraries)
DELETE (library program units)
DELETE (load path)
DELETE (program units)
DESCRIBE (debug actions)
DESCRIBE (load path)
DESCRIBE (locals)
DESCRIBE (libraries)
DESCRIBE (program units)
DESCRIBE (tables and views)
DESCRIBE (version)
DETACH
DISABLE (compiler options)
DISABLE (debug actions)
DISABLE (logging)
DISCONNECT
ENABLE (compiler options)
ENABLE (debug actions)
ENABLE (logging)
EXECUTE
EXPORT
GENERATE
GO
GRANT
HELP
INSERT (library program unit)
INSERT (load path)
INTERPRET
LIST (debug actions)
LIST (program units)
LOAD (library program units)
LOAD (program units)
LOAD (stored program units)
LOG
OPEN
QUIT
RENAME
RESET
REVERT
REVOKE
SAVE
SET
SHOW (call stack)
SHOW (debug actions)
SHOW (libraries)
SHOW (locals)
SHOW (program units)
STEP
STORE
TRIGGER

---

**Bind variable commands**

CREATE (bind variables)
DELETE (bind variables)

---

**Database commands**

CONNECT
DESCRIBE (tables and views)
DISCONNECT
GRANT
REVOKE
STORE

---

**Debug action commands**

BREAK
DELETE (debug actions)
DESCRIBE (debug actions)
DISABLE (debug actions)
ENABLE (debug actions)
LIST (debug actions)
SHOW (debug actions)
TRIGGER

**Debugging commands**

DESCRIBE (locals)
GO
RESET
SET
SHOW (call stack)
STEP

**Library commands**

ATTACH
CLOSE
COMPILE (libraries)
CREATE (libraries)
DELETE (libraries)
DELETE (library program unit)
DESCRIBE (libraries)
DETACH
EXPORT (libraries)
GENERATE
GRANT
INSERT (library program unit)
LOAD (library program units)
OPEN
RENAME
REVERT
REVOKE
SAVE
SHOW (libraries)

**Load path commands**

DELETE (load path)
DESCRIBE (load path)
Logging commands

DISABLE (logging)
ENABLE (logging)
LOG

Program unit commands

COMPILE (program units)
DELETE (program units)
DESCRIBE (program units)
DISABLE (compiler options)
ENABLE (compiler options)
EXECUTE
EXPORT (program units)
EXPORT (stored program units)
LIST (program units)
LOAD (program units)
LOAD (stored program units)
SHOW (locals)
SHOW (program units)

Session commands

DESCRIBE (version)
HELP
INTERPRET
QUIT

Interpreter commands

Bind variable commands
Database commands
Debug action commands
Debugging commands
Library commands
Load path commands
Logging commands
Bind Variable Commands

CREATE (bind variable) command (PB standalone only)

Description
Creates a bind variable. This command is valid only when Procedure Builder is invoked as a standalone session.

Syntax
CREATE CHAR var_name [LENGTH number]
CREATE NUMBER var_name [PRECISION number] [SCALE number]
CREATE RAW var_name [LENGTH number]
CREATE DATE var_name

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>Specifies a bind variable, var_name, of the datatype CHAR.</td>
</tr>
<tr>
<td>var_name</td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td>Optionally specifies the length of a CHAR bind variable.</td>
</tr>
<tr>
<td>number</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>Specifies a bind variable, var_name, of the datatype DATE.</td>
</tr>
<tr>
<td>var_name</td>
<td></td>
</tr>
<tr>
<td>NUMBER</td>
<td>Specifies a bind variable, var_name, of the datatype NUMBER.</td>
</tr>
<tr>
<td>var_name</td>
<td></td>
</tr>
<tr>
<td>PRECISION</td>
<td>Optionally determines a maximum number of numeric digits for the variable.</td>
</tr>
<tr>
<td>number</td>
<td></td>
</tr>
<tr>
<td>SCALE</td>
<td>Optionally determines where rounding should occur.</td>
</tr>
<tr>
<td>number</td>
<td></td>
</tr>
<tr>
<td>RAW</td>
<td>Specifies a bind variable, var_name, of the datatype RAW.</td>
</tr>
<tr>
<td>var_name</td>
<td></td>
</tr>
</tbody>
</table>

Comments
The LENGTH attribute of the CHAR datatype defaults to 1 byte if you do not specify an alternate setting. The maximum value for LENGTH is 32767 bytes.

The maximum value for PRECISION is 38 characters. SCALE can be from -84 to 127. If you do not specify a value for SCALE, it defaults to zero, meaning numbers are rounded to the nearest whole number.

For more information about datatypes and their attributes, see the PL/SQL User's Guide and Reference.

CREATE (bind variable) command example

The following command creates a bind variable x of the datatype NUMBER that should round to the nearest hundredth decimal place:

```
.CREATE NUMBER x SCALE 2
```
DELETE (bind variables) command (PB standalone only)

**Description**  
Deletes one or more bind variables. This command is valid only when Procedure Builder is invoked as a standalone session.

**Syntax**
```
DELETE BINDVAR name [, name...]
DELETE CHAR name [, name...]
DELETE DATE name [, name...]
DELETE NUMBER name [, name...]
```

**Keywords and Values**
- **BINDVAR name**  
  Specifies a bind variable or set of bind variables of any datatype
- **CHAR name**  
  Specifies a bind variable or set of bind variables of the datatype CHAR
- **DATE name**  
  Specifies a bind variable or set of bind variables of the datatype DATE
- **NUMBER name**  
  Specifies a bind variable or set of bind variables of the datatype NUMBER

**DELETE (bind variables) command examples**

The following command deletes the bind variable `y` of the datatype CHAR:
```
.DELETE CHAR y
```

The following command deletes a set of bind variables (`x`, `y`, and `z`) of different datatypes:
```
.DELETE BINDVAR x,y,z
```
Database Commands

CONNECT command (PB standalone only)

**Description**
Establishes a database connection. This command is valid only when Procedure Builder is invoked as a standalone session.

**Syntax**
CONNECT DB [username/password@ | network_device: | datasource_node: | datasource_name] [SILENT]

**Keywords and Values**
- **username/password@** Indicates a valid user name and password for the datasource to which you wish to connect. The '@' symbol must precede the remaining database location specifiers.
- **network_device**: Specifies the networking device driver used to connect to the remote database.
- **datasource_node**: Specifies the network node of the remote datasource to which you wish to connect.
- **datasource_name**: Specifies the name of the remote or local datasource to which you wish to connect.
- **SILENT**: Optionally suppresses the status messages issued by the Interpreter.

**Note** If you wish to connect to an ODBC datasource, use the following syntax:

```
username/password@ODBC:datasource[:dbname]
```

If `dbname` is not specified, the current database for the ODBC connection is used.

**CONNECT command examples**
The following command would connect you to the remote "inventory" database on the "boston" network node using the TCP/IP device driver.

```
.CONNECT DB scott/tiger@t.boston:inventory
```

If the "inventory" database were a local database, the following command would connect you:

```
.CONNECT DB scott/tiger@inventory
```

**DESCRIBE (tables and views) command**

**Description**
Displays detailed information about database tables and views.

**Syntax**
DESCRIBE TABLE name
DESCRIBE VIEW name

Keywords and Values

<table>
<thead>
<tr>
<th>TABLE name</th>
<th>Specifies a table in the current database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW name</td>
<td>Specifies a view in the current database.</td>
</tr>
</tbody>
</table>

Comments

The information displayed for tables and views includes the columns and their types.

DESCRIBE (tables and views) command examples

The following command displays information about the EMP table:

`.DESCRIBE TABLE emp`

The following command displays information about the view named ASSOCIATE:

`.DESC V associate`

DISCONNECT command (PB standalone only)

Description

Disconnects you from the database to which you are currently connected. This command is valid only when Procedure Builder is invoked as a standalone session.

Syntax

`DISCONNECT`

GRANT command (Database commands)

Description

Grants a user access to a library stored in the database.

Syntax

`GRANT LIBRARY name USER name`

Keywords and Values

<table>
<thead>
<tr>
<th>LIBRARY name</th>
<th>Specifies the library.</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER name</td>
<td>Specifies a user name.</td>
</tr>
</tbody>
</table>

Comments

You can specify any single valid user name, or PUBLIC (all users).

GRANT command example (Database commands)

The following command grants user SCOTT access to database library `lib1`:

`.GRANT LIB lib1 USER scott`

REVOKE command (Database commands)

Description

Revokes a user's access to a library stored in the database.

Syntax

`REVOKE LIBRARY name USER name`

Keywords and Values
LIBRARY name  Specifies a library.
USER name     Specifies a user.

Comments   You can specify any single valid user name, or PUBLIC (all users).

REVOKE command example (Database commands)
The following command revokes user SCOTT’s access to database library \textit{lib1}:
\texttt{.REVOKE LIB lib1 USER scott}

\section*{STORE command}
\textbf{Description}   Stores one or more program units in the database.
\textbf{Syntax}
\begin{verbatim}
STORE PROGRAMUNIT name [\[name...]  
FILE \{directory\}name[extension]  
\{SPECIFICATION | BODY\}  
\{NOWARN\}
STORE PACKAGE name[, name...]  
\{OWNER name\}  
\{SPECIFICATION | BODY\}
STORE SUBPROGRAM name[, name...]  
\{OWNER name\}  
\{SPECIFICATION | BODY\}
STORE PROCEDURE name[, name...]  
\{OWNER name\}  
\{SPECIFICATION | BODY\}
STORE FUNCTION name[, name...]  
\{OWNER name\}  
\{SPECIFICATION | BODY\}
\end{verbatim}

\textbf{Keywords and Values}
\begin{itemize}
\item PROGRAMUNIT name   Specifies one or more program units.
\item PACKAGE name        Specifies one or more packages.
\item SUBPROGRAM name     Specifies one or more subprograms.
\item OWNER name          Specifies the owner of the stored program unit(s).
\item SPECIFICATION or BODY Dictates whether the specification or body of a package is stored in the database, respectively.
\end{itemize}

Comments   If OWNER is not specified, Procedure Builder assigns the currently connected user as the owner of the stored program units.
If neither SPECIFICATION nor BODY is supplied, both the body and the specification (if available) of the designated package(s) are stored in the database.
STORE command examples

The following command stores procedure \textit{proc1} and function \textit{func2} in the current database:
\small
\texttt{.STORE PROGRAMUNIT proc1, func2}
\normalsize

The following command stores the specification and body of package \textit{pack1} and specifies the owner to be SCOTT:
\small
\texttt{.STORE PACK pack1 OWNER scott}
\normalsize
Debug Action Commands

BREAK command

Description  Establishes a breakpoint at the specified source line within a program unit.

Syntax

\[
\text{BREAK \{[USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.]name} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK \{USER schema PACKAGE name | PACKAGE schema.name} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK \{USER schema SUBPROGRAM name | SUBPROGRAM schema.name} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK \{USER schema PROCEDURE name | PROCEDURE schema.name} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK \{USER schema FUNCTION name | FUNCTION schema.name} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK ACTION number} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK BREAKPOINT number} \\
\text{[LINE number]} \text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK .} \\
\text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK PC} \\
\text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

\[
\text{BREAK SCOPE} \\
\text{[ENABLED | DISABLED]} \text{[TRIGGER plsql-block]}
\]

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER schema</td>
<td>Specifies a schema name in the database where the stored program unit is located.</td>
</tr>
<tr>
<td>PROGRAMUNIT</td>
<td>Specifies a program unit body.</td>
</tr>
<tr>
<td>T name</td>
<td></td>
</tr>
<tr>
<td>PACKAGE name</td>
<td>Specifies a package body.</td>
</tr>
<tr>
<td>SUBPROGRAM</td>
<td>Specifies a subprogram body.</td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
</tbody>
</table>
PROCEDURE name  Specifies a procedure body.

FUNCTION name  Specifies a function body.

ACTION number  Specifies a debug action (breakpoint or trigger).

BREAKPOINT number  Specifies a breakpoint.

.  Specifies the current source location. This is the default.

PC  Specifies the current execution location.

SCOPE  Specifies the current scope location.

LINE number  Specifies the line in a program unit at which to establish the breakpoint.

ENABLED or DISABLED  Specifies whether or not the breakpoint is initially enabled or disabled. The default is ENABLED.

TRIGGER pl/sql-block  Defines a PL/SQL trigger for the breakpoint. The trigger fires each time the breakpoint is reached.

Note  If supplied, the TRIGGER keyword must appear as the last command option.

Comments  BREAK may operate only on executable source lines.

Trigger blocks may span multiple input lines. As is the case when entering PL/SQL constructs elsewhere in the Interpreter, no line continuation characters are required when entering the trigger body (nor are they allowed).

If you wish to interrupt your program conditionally, you should use the TRIGGER command in conjunction with the DEBUG.BREAK exception.

If the statement is reached while running PL/SQL, Procedure Builder suspends execution just before the statement is executed, and passes control to the Interpreter. At this point, you can inspect and even modify program state using a variety of Procedure Builder functions.

Once satisfied, you can resume execution with the GO or STEP commands. Alternatively, you can abort execution using the RESET command.

BREAK command examples

The following command sets a breakpoint at the current source location:

.BREAK .

The following command sets a breakpoint at the second line of the procedure named my_proc:

.BREAK PROCEDURE my_proc LINE 2

The following command sets a breakpoint at the tenth line of my_proc that shows all of the local variables and their values whenever the breakpoint is entered:

.BREAK PROC my_proc LINE 10 TRIGGER debug.interpret('.SHOW LOCALS')

The following command sets a breakpoint at line twelve of the program unit that contains debug action number four:

.BREAK ACTION 4 LINE 12

The following command sets a breakpoint at the current source location in a server-side program unit my_proc from the schema owned by user scott:

.BREAK USER scott PROC my_proc
DELETE (debug actions) command

Description Deletes one or more debug actions.

Syntax
DELETE ACTION number [, number...]
DELETE BREAKPOINT number [, number...]
DELETE TRIGGER number [, number...]

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>Specifies one or more debug actions (breakpoint or trigger), by number.</td>
</tr>
<tr>
<td>BREAKPOINT</td>
<td>Specifies one or more breakpoints, by number.</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Specifies one or more debug triggers, by number.</td>
</tr>
</tbody>
</table>

Comments This command permanently removes one or more debug actions. If you wish to temporarily remove a debug action, use the DISABLE command instead.

DELETE (debug actions) command example

The following command deletes debug actions two and three:
.DELETE ACTION 2, 3

DESCRIBE (debug actions) command

Description Displays detailed information about the specified debug action.

Syntax
DESCRIBE ACTION number
DESCRIBE BREAKPOINT number
DESCRIBE TRIGGER number

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>Specifies a debug action (a breakpoint or a trigger).</td>
</tr>
<tr>
<td>BREAKPOINT</td>
<td>Specifies a breakpoint.</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>Specifies a trigger.</td>
</tr>
</tbody>
</table>

Comments The information displayed for a debug action includes its ID, the source location with which it is associated, and whether or not it is enabled.
DESCRIBE (debug actions) command examples

The following command displays information about breakpoint number two:

```
.DESCRIBE BREAK 2
```

The following command displays information about debug action number three:

```
.DESCRIBE ACTION 3
```

DISABLE (debug actions) command

Description

Removes one or more debug actions temporarily.

Syntax

```
DISABLE ACTION number [, number...]
DISABLE BREAKPOINT number [, number...]
DISABLE TRIGGER number [, number...]
```

Keywords and Values

- **ACTION**
  - `number`: Specifies one or more debug actions (breakpoints and triggers).
- **BREAKPOINT**
  - `number`: Specifies one or more breakpoints.
- **TRIGGER**
  - `number`: Specifies one or more triggers.

Comments

DISABLE has no effect on debug actions that are already disabled. You can restore disabled debug actions using the ENABLE command.

DISABLE (debug actions) command examples

The following command disables breakpoint number two:

```
.DISABLE BREAK 2
```

The following command disables debug action number three:

```
.DISABLE ACTION 3
```

ENABLE (debug actions) command

Description

Reactivates disabled debug actions.

Syntax

```
ENABLE ACTION number [, number...]
ENABLE BREAKPOINT number [, number...]
ENABLE TRIGGER number [, number...]
```

Keywords and Values

- **ACTION**
  - `number`: Specifies a debug action.
- **BREAKPOINT**
  - `number`: Specifies a breakpoint.
- **TRIGGER**
  - `number`: Specifies a trigger.
Comments

ENABLE has no effect on debug actions that are already enabled. To temporarily disable a debug action, use the DISABLE command.

ENABLE (debug actions) command examples

The following command enables breakpoint number two, which was previously disabled:

```
.ENABLE BREAK 2
```

The following command enables debug action number one:

```
.ENABLE ACTION 1
```

---

LIST (debug actions) command

Description

Displays the program unit source text to which the specified debug action is attached.

Syntax

```
LIST ACTION number
LIST BREAKPOINT number
LIST TRIGGER number
```

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION number</td>
<td>Specifies a debug action (breakpoint or trigger).</td>
</tr>
<tr>
<td>BREAKPOINT number</td>
<td>Specifies a breakpoint.</td>
</tr>
<tr>
<td>TRIGGER number</td>
<td>Specifies a trigger.</td>
</tr>
</tbody>
</table>

Comments

LIST displays the text associated with the specified debug action in the Source pane of the Interpreter. The line on which the specified debug action appears becomes the current source location.

LIST (debug actions) command examples

The following command displays breakpoint number one and sets the source location:

```
.LIST BREAK 1
```

The following command displays debug action number three and sets the current source location:

```
.LIST ACTION 3
```

---

SHOW (debug actions) command

Description

Enumerates the debug actions that are currently defined in the development session.

Syntax

```
SHOW ACTION
SHOW BREAKPOINTS
SHOW TRIGGERS
```

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>Specifies all debug actions.</td>
</tr>
<tr>
<td>BREAKPOINTS</td>
<td>Specifies all breakpoints.</td>
</tr>
<tr>
<td>TRIGGERS</td>
<td>Specifies all triggers.</td>
</tr>
</tbody>
</table>
SHOW (debug actions) command example

The following command lists all of the breakpoints that are currently set:

`.SHOW BREAKPOINTS`

---

**TRIGGER command**

**Description**  Creates a debug trigger, which is a PL/SQL block associated with the specified source location.

**Syntax**

```
TRIGGER {[USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.]name}
   [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER {USER schema PACKAGE name | PACKAGE schema.name}
   [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER {USER schema SUBPROGRAM name | SUBPROGRAM schema.name}
   [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER {USER schema PROCEDURE name | PROCEDURE schema.name}
   [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER {USER schema FUNCTION name | FUNCTION schema.name}
   [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER ACTION number [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER BREAKPOINT number [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER TRIGGER number [LINE number]
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER .
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER PC
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER SCOPE
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER DEBUG
   [ENABLED | DISABLED]
   [IS plsql-block]

TRIGGER *
   [ENABLED | DISABLED]
   [IS plsql-block]
```

**Keywords and Values**

- **USER schema**  Specifies a schema name in the database where the stored program unit is located.
PROGRAMUNI  Specifies a program unit.
T  name

PACKAGE  name  Specifies a package.

SUBPROGRAM  name  Specifies a subprogram.

PROCEDURE  name  Specifies a procedure.

FUNCTION  name  Specifies a function.

ACTION  number  Specifies a debug action (breakpoint or trigger).

BREAKPOINT  number  Specifies a breakpoint.

TRIGGER  number  Specifies a trigger.

LINE  number  Specifies the line of the program unit where the trigger should be located.
.
PC  Specifies the current source location.

SCOPE  Specifies the current execution location.

DEBUG  Specifies entry into the debugger (i.e., when program execution is suspended due to a breakpoint, program stepping, etc.).

*  Specifies every PL/SQL source line. Thus, placing a trigger on * will cause the specified block to be evaluated just prior to executing every PL/SQL source line.

ENABLED  or  DISABLED  Dictates whether the trigger is initially enabled or disabled. The default is ENABLED.

IS  pl/sql-block  Defines the body of the trigger.

Note: IS must appear as the last command option.

Comments  Procedure Builder executes the trigger just before the program reaches the specified location. The trigger block may span multiple input lines. As is the case when entering PL/SQL constructs elsewhere in the Interpreter, no line continuation characters are required when entering the trigger body (nor are they allowed).

TRIGGER is especially handy for creating conditional breakpoints. This is done by raising the exception DEBUG.BREAK from within the arbitrarily complex control logic of the trigger body. The exception is trapped by the debugger, which interrupts program execution and passes control to the Interpreter as if a breakpoint had been entered at the trigger location.

TRIGGER command examples

The following trigger establishes a conditional breakpoint on line ten of my_proc that is only reached if the local NUMBER variable 'i' exceeds 100:

```
.TRIGGER PROC my_proc LINE 10 IS
  IF DEBUG.GETN('i') > 100 THEN
    RAISE DEBUG.BREAK;
  END IF;
```
Triggers can also be used to trace program execution. The following trigger lists every source statement as it is executed:

```
.TRIGGER * IS debug.interpret('LIST PC');
```

The following command sets a trigger at line 8 in a server-side program unit `my_proc` from the schema owned by user `scott`:

```
.TRIGGER USER scott PROC my_proc LINE 8
```

or

```
.TRIGGER PROC scott.my_proc LINE 8
```
Debugging Commands

DESCRIBE (locals) command

Description  Displays the name, type, and value of a variable or parameter that is local to the current scope location.

Syntax

- DESCRIBE LOCAL name
- DESCRIBE PARAMETER name
- DESCRIBE VARIABLE name

Keywords and Values

- LOCAL name  Specifies a parameter or variable local to the current scope location.
- PARAMETER name  Specifies a parameter local to the current scope location.
- VARIABLE name  Specifies a variable local to the current scope location.

DESCRIBE (locals) command examples

The following command displays information about the parameter p1:

```
.DESCRIPTION PARAM p1
```

The following command displays information about the local variable sal:

```
.DESCRIPTION LOCAL sal
```

GO command

Description  Resumes program execution indefinitely, after a breakpoint or debug trigger.

Syntax

- GO

Comments  GO resumes program execution until the currently executing thread of execution either terminates or is interrupted by a debug action.

GO command example

The following command resumes program execution:

```
.GO
```
RESET command

**Description**  Returns control to an outer debug level without continuing execution in the current debug level.

**Syntax**

```
RESET LEVEL number
```

**Keywords and Values**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL number</td>
<td>Specifies an outer debug level.</td>
</tr>
</tbody>
</table>

**Comments**  RESET effectively aborts execution at the current and possibly higher debug levels. You can perform a relative reset by supplying a negative value for LEVEL number. Invoking RESET with no options always returns to top level.

**RESET command examples**

The following command resets to the previous debug level:

```
.RESET LEVEL -1
```

The following command resets to the top level:

```
.RESET
```

SET command

**Description**  Changes the current scope location to a specified frame of the call stack. The current scope location affects how local variable references are treated in the Interpreter.

**Syntax**

```
SET SCOPE FRAME number
SET SCOPE UP [COUNT number]
SET SCOPE DOWN [COUNT number]
SET SCOPE TOP
SET SCOPE BOTTOM
SET SCOPE PROGRAMUNIT name
SET SCOPE PACKAGE name
SET SCOPE SUBPROGRAM name
SET SCOPE PROCEDURE name
SET SCOPE FUNCTION name
```

**Keywords and Values**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRAME number</td>
<td>Specifies a frame by number.</td>
</tr>
<tr>
<td>UP</td>
<td>Specifies relative motion toward the top of the stack.</td>
</tr>
<tr>
<td>DOWN</td>
<td>Specifies relative motion toward the bottom of the stack.</td>
</tr>
<tr>
<td>COUNT number</td>
<td>Specifies a repeat count in the specified direction (UP or DOWN). The default is one.</td>
</tr>
<tr>
<td>TOP</td>
<td>Specifies the top frame in the call stack.</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>Specifies the bottom frame in the call stack.</td>
</tr>
<tr>
<td>PROGRAMUNIT</td>
<td>Specifies a program unit.</td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
</tbody>
</table>
PACKAGE name  Specifies a package.
SUBPROGRAM name  Specifies a subprogram.
PROCEDURE name  Specifies a procedure.
FUNCTION name  Specifies a function.

Comments  Frames are numbered from 0 (top frame) to \( n \) (bottom frame).

SET command examples
The following command moves up one stack frame:
\[ .SET \text{SCOPE UP} \]
The following command moves down two frames:
\[ .SET \text{SCOPE DOWN COUNT 2} \]
The following command moves to the frame associated with the function \( \text{func1} \):
\[ .SET \text{SCOPE FUNCTION func1} \]
The following command moves to the top of the stack:
\[ .SET \text{SCOPE TOP} \]
The following command moves to the fifth frame:
\[ .SET \text{SCOPE FRAME 5} \]

SHOW (call stack) command
Description  Lists the frames on the current call stack.
Syntax
\[ \text{SHOW STACK} \]
\[ \text{SHOW SCOPE} \]

Keywords and Values

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STACK</td>
<td>Lists the program unit name and line number for every frame on the call stack.</td>
</tr>
<tr>
<td>SCOPE</td>
<td>Lists the frames from the top of the call stack down to the frame containing the current scope location.</td>
</tr>
</tbody>
</table>

SHOW (call stack) command example
The following command lists the current call stack:
\[ .SHOW \text{STACK} \]

STEP command
Description  Advances execution of an interrupted program unit.
Syntax
STEP INTO
STEP OVER
STEP OUT
STEP TO PROGRAMUNIT name [LINE number]
STEP TO PACKAGE name [LINE number]
STEP TO SUBPROGRAM name [LINE number]
STEP TO PROCEDURE name [LINE number]
STEP TO FUNCTION name [LINE number]
STEP TO ACTION number
STEP TO BREAKPOINT number
STEP TO TRIGGER number
STEP TO . [LINE number]
STEP COUNT number

Keywords and Values

INTO Enables stepping into subprogram calls. This is the default if no keywords are specified.
OVER Prevents stepping into a called subprogram body.
OUT Resumes execution until the current subprogram has returned.
TO ... Continues execution until the specified source location is reached. Using the TO option is analogous to setting a temporary breakpoint at the specified location.

PROGRAMUNIT name Specifies a program unit.
PACKAGE name Specifies a package.
SUBPROGRAM name Specifies a subprogram.
PROCEDURE name Specifies a procedure.
FUNCTION name Specifies a function.
ACTION number Specifies a debug action (breakpoint or trigger).
BREAKPOINT number Specifies a breakpoint.
TRIGGER number Specifies a trigger.
.
LINE number Specifies the line of the program unit.
COUNT number Dictates how many times the STEP command (as qualified by the other options) is repeated. The default is 1.

Comments Control returns to the current debug level after the specified set of statements have been executed.
STEP command examples

The following command resumes execution until the first breakpoint is reached:

```
.STEP TO BREAK 1
```

The following command resumes execution for five lines:

```
.STEP COUNT 5
```
Library Commands

**ATTACH command**

**Description**  
Attaches a PL/SQL library to the current session.

**Syntax**

```plaintext
ATTACH LIBRARY [directory] name[extension]  
[FILESYSTEM | DB]  
[BEFORE library]  
[AFTER library]  
[START | END]
```

**Keywords and Values**

- **LIBRARY name**: Specifies the name of the library, including the optional directory path and extension if stored in the file system.
- **FILESYSTEM or DB**: Specifies whether the specified library is stored in the file system or in the currently connected database. If neither keyword is specified, tries to access the specified library first in the file system and then, if unsuccessful, in the current database.
- **BEFORE library**: Specifies to attach the library before the named attached library.
- **AFTER library**: Specifies to attach the library after the named attached library.
- **START or END**: Specifies whether the attached library is placed at the beginning or the end of the attach list, respectively. The default is START.

**Comments**  
If you attempt to attach a library in the file system, the load path and the extension .pll are used when resolving `lib-name`, unless `directory` and `extension` are specified explicitly. Note that the syntax of `directory` is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

Libraries are attached read-only. If you want to modify the contents of a library, use the OPEN command to open the library for editing.

**ATTACH command example**

The following command attaches to the library residing in the file `lib1`:

```
ATTACH LIB lib1 FILE
```
CLOSE command

Description        Removes one or more libraries from the current set of open libraries.
Syntax
CLOSE LIBRARY name[, name...] [DISCARD]

Keywords and Values
LIBRARY name        Specifies one or more currently open libraries that you wish to close.
DISCARD             Discards changes made to the libraries since the last save.

Comments           Closing a library removes all program units loaded from that library into the environment. The namespace used to represent the library is also removed.
Closing a library automatically saves any changes made to the library since it was opened. Specifying DISCARD discards any changes made to the library since the last save operation.

CLOSE command example
The following command closes the libraries `lib1` and `lib2`:
```
CLOSE LIB lib1, lib2
```

COMPILE (libraries) command

Description        Compiles/recompiles all of the program units in one or more open libraries.
Syntax
COMPILE LIBRARY name[, name...] [INCREMENTAL]

Keywords and Values
LIBRARY name        Specifies one or more open libraries whose program units you wish to compile.
INCREMENTAL         Compiles only those program units within the library that need to be compiled.

Comments           When invoked, COMPILE first checks for any currently loaded program unit(s) that match the name and type of the program unit(s) in the library to be compiled. If there is at least one match, you are asked if you wish to continue compilation. Answering Yes removes all of the matching program units from the environment, compiles them, and saves them in the specified open library. Answering No aborts the operation.
Note: The compiled program unit(s) are saved in the open library, but are not reloaded into the environment. You can invoke the LOAD command (via the Interpreter command line or File→Load) to reload them into the environment.
If INCREMENTAL is not specified, all program units in the library are force-compiled.

COMPILE (libraries) command example
The following command compiles all of the program units in the open library named `lib1`:
```
.COMPILE LIB lib1
```
CREATE (libraries) command

Description
Creates a new library that can be stored in either the file system or the current database.

Syntax
CREATE LIBRARY [directory] lib-name[extension]
[SOURCE pld-file]
[NOCOMPILE]
[FILESYSTEM | DB]
[BEFORE | AFTER]
[NOCONFIRM]

Keywords and Values
- **LIBRARY name**: Specifies the name of the library, including the optional directory path and extension if created in the file system.
- **SOURCE pld-file**: Specifies a file containing the source of one or more program units.
- **NOCOMPILE**: Prevents the contents of the newly created library from being compiled.
- **FILESYSTEM or DB**: Indicates whether the specified library should be created in the file system or in the currently connected database. The default is FILESYSTEM.
- **BEFORE or AFTER**: Dictates whether the attached library is placed at the beginning or the end of the attach list, respectively. The default is BEFORE.
- **NOCONFIRM**: Specifies to overwrite an existing library without prompting you for confirmation.

Comments
For libraries created in the file system, the name of the library is designated by the basename of the file (the full filename minus any leading directory and any trailing extension). For example, in UNIX, the file /private/libs/emplib.pll holds the library named *emplib*.

The syntax of directory is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

The newly created library is automatically opened. Once a library has been opened, you can modify its contents using the INSERT and DELETE commands.

Using the SOURCE keyword, you can immediately populate the newly created library with the source of one or more program units contained in the specified pld-file. The library is then compiled (using the COMPIL E LIBRARY command) unless you specify the NOCOMPILE keyword.

If you try to create a library with the same name as an existing library, a message box displays, asking if you want to overwrite the existing library. Specifying NOCONFIRM in the command string suppresses the alert.

CREATE (libraries) command example

In UNIX, the following command creates a new library named *lib1* residing in the file /private/libs/lib1.pll:

```
.CREATE LIBRARY /private/libs/lib1.pll FILE
```

DELETE (libraries) command

Description
Deletes one or more libraries that reside in the current database.
Syntax
DELETE LIBRARY name[, name...]

Keywords and Values
LIBRARY name Specifies a library.

Comments You cannot delete a library that is currently attached. Use the DETACH command to detach a library before you delete it.

DELETE (libraries) command example
The following command deletes library lib1 from the database:
.DELETE LIB lib1

DELETE (library program units) command
Description Deletes one or more program units from an open library.

Syntax
DELETE PROGRAMUNIT name [, name...]
   LIBRARY name
   [SPECIFICATION | BODY]
DELETE PACKAGE name [, name...]
   LIBRARY name
   [SPECIFICATION | BODY]
DELETE SUBPROGRAM name [, name...]
   LIBRARY name
   [SPECIFICATION | BODY]
DELETE PROCEDURE name [, name...]
   LIBRARY name
   [SPECIFICATION | BODY]
DELETE FUNCTION name [, name...]
   LIBRARY name
   [SPECIFICATION | BODY]

Keywords and Values

   PROGRAMUNIT name Specifies one or more program units.
   PACKAGE name Specifies one or more packages.
   SUBPROGRAM name Specifies one or more subprograms.
   PROCEDURE name Specifies one or more procedures.
   FUNCTION name Specifies one or more functions.
   LIBRARY name Specifies the library from which the program unit(s) should be deleted.
   SPECIFICATION or BODY Specifies that either the specification or the body of the program unit be deleted.

DELETE (library program units) command example
The following command deletes the package named p2 from the library named lib1:
DESCRIBE (libraries) command

Description  Displays detailed information about an attached library.

Syntax
DESCRIBE LIBRARY name

Keywords and Values
LIBRARY name  Specifies the name of an attached library.

Comments  The information displayed for a library includes the mode in which it was attached, its external location, and its contents.

DESCRIBE (libraries) command example

The following command displays information about the library named lib1:
.DESCRIBE LIBRARY lib1

DETACH command

Description  Removes one or more libraries from the current set of attached libraries.

Syntax
DETACH LIBRARY name[, name...]

Keywords and Values
LIBRARY name  Specifies one or more attached libraries.

Comments  Detaching a library removes all unmodified program units loaded from that library into the environment. Note that once a program unit loaded from a library has been modified within the environment (e.g., by compilation), it will not be removed when the library is detached.

DETACH command example

The following command detaches the libraries lib1 and lib2:
.DEDETACH LIBRARY lib1, lib2

EXPORT (libraries) command

Description  Writes the source of a library to a text file.

Syntax
EXPORT {LIBRARY name} FILE [directory]name[extension] [NOWARN]

Keywords and Values
LIBRARY name  Specifies an attached library.
FILE name     Specifies the name of the file, including the
optional directory path and extension.

NOWARN Suppresses the warning that a built-in program unit was not added to the attached library.

Comments If unspecified, the file extension defaults to .pld. The syntax of directory is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

EXPORT (libraries) command example
The following command writes the source of procedure \( p1 \) and function \( f3 \) to the file pl1.pld:

```
.EXPORT LIB my_lib FILE lib1
```

GENERATE command
Description Creates a runtime version of a currently open library.

Syntax
```
GENERATE LIBRARY name
FILE [directory]name
[INCREMENTAL]
```

Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY name</td>
<td>Specifies the name of the open library.</td>
</tr>
<tr>
<td>FILE name</td>
<td>Specifies the file name of the runtime library, including the optional directory path. The default file extension .plx is automatically assigned.</td>
</tr>
<tr>
<td>INCREMENTAL</td>
<td>Specifies that only the uncompiled program units in the open library are to be compiled.</td>
</tr>
</tbody>
</table>

Comments This command creates a temporary library, copies all program units from the open library and inserts them in the temporary library, compiles the program units, then executes the SAVE command on the temporary library using the NOSOURCE and NODIANA keywords. The resulting library is identified as a runtime only library with the .plx file extension. If you specify a different file extension, that extension will be used instead of .plx.

GENERATE command example
The following command creates a runtime library named runlib1.plx based on the open library mylib.pll:

```
.GENERATE LIB mylib FILE runlib1
```

Since the INCREMENTAL keyword was not specified, all program units in the library mylib will be force compiled.

GRANT command (Library commands)
Description Grants a user access to a library stored in the database.

Syntax
```
GRANT LIBRARY name USER name
```
Keywords and Values

LIBRARY name  Specifies the library.
USER name     Specifies a user name.

Comments    You can specify any single valid user name, or PUBLIC (all users).

GRANT command example (Library commands)

The following command grants user SCOTT access to database library lib1:

```
.GRANT LIB lib1 USER scott
```

INSERT (library program units) command

Description    Inserts one or more program units into an open library.

Syntax

```
INSERT PACKAGE name[, name...]  
  [SPECIFICATION | BODY]  
  LIBRARY name  
  [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]

INSERT SUBPROGRAM name[, name...]  
  [SPECIFICATION | BODY]  
  LIBRARY name  
  [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]

INSERT PROCEDURE name[, name...]  
  [SPECIFICATION | BODY]  
  LIBRARY name  
  [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]

INSERT FUNCTION name[, name...]  
  [SPECIFICATION | BODY]  
  LIBRARY name  
  [NOPCODE] [NODIANA] [NOSOURCE] [NOWARN]
```

Keywords and Values

PACKAGE name    Specifies one or more packages.
SUBPROGRAM name  Specifies one or more subprograms.
PROCEDURE name   Specifies one or more procedures.
FUNCTION name    Specifies one or more functions.
SPECIFICATION or BODY  Specifies the specification or the body of the designated program unit(s), respectively. If neither keyword is supplied, both are inserted into the library.
LIBRARY name     Specifies the target library.
NOPCODE          Adds the program unit(s) to the library without the PCODE.
NODIANA          Adds the program unit(s) to the library without the DIANA.
NOSOURCE         Adds the program unit(s) to the library without the source code.
NOWARN           Suppresses the warning that a built-in program
unit was not inserted into the open library.

Comments You can use NODIANA and NOSOURCE to dramatically reduce the size of a PL/SQL library.

Note: Program units inserted into libraries with NODIANA and NOSOURCE can be used only in a runtime environment, because it is impossible to compile references to program units that do not include DIANA.

Attempting to insert a built-in program unit into an open library (e.g., .INSERT PROG * LIB lib3) displays a warning in the Interpreter pane (Warning: Program unit <progunit name> has no source to insert...). Specifying NOWARN in the command string suppresses the warning.

INSERT (library program units) command example

The following command inserts the packages p1 and p2 into the library named lib1:

```
.INSERT PACKAGE p1,p2 LIBRARY lib1
```

LOAD (library program units) command

Description Loads one or more program units from an attached library.

Syntax

```
LOAD PROGRAMUNIT name[, name...]  
  LIBRARY name  
  [SPECIFICATION | BODY]  
  [NOCONFIRM]
LOAD PACKAGE name[, name...]  
  LIBRARY name  
  [SPECIFICATION | BODY]  
  [NOCONFIRM]
LOAD SUBPROGRAM name[, name...]  
  LIBRARY name  
  [SPECIFICATION | BODY]  
  [NOCONFIRM]
LOAD PROCEDURE name[, name...]  
  LIBRARY name  
  [SPECIFICATION | BODY]  
  [NOCONFIRM]
LOAD FUNCTION name[, name...]  
  LIBRARY name  
  [SPECIFICATION | BODY]  
  [NOCONFIRM]
```

Keywords and Values

- `PROGRAMUNIT` Specifies one or more program units.
- `PACKAGE` Specifies one or more packages.
- `SUBPROGRAM` Specifies one or more subprograms.
- `PROCEDURE` Specifies one or more procedures.
- `FUNCTION` Specifies one or more functions.
- `LIBRARY` Specifies an attached library.
- `SPECIFICATION` Specifies that either the specification or the body of the program unit be loaded.
N or BODY respectively. If neither is specified, both are loaded.

NOCONFIRM Specifies to redefine an existing program unit without prompting you for confirmation.

Comments If you try to load a library program unit with the same name and type as an existing program unit, a message box displays, asking if you want to redefine the existing program unit. Specifying NOCONFIRM in the command string suppresses the alert.

LOAD (library program units) command example

The following command loads the function \textit{f1} and the procedure \textit{p3}, both of which are stored in the attached library \textit{lib1}:

\texttt{.LOAD PROGRAMUNIT f1, p3 LIB lib1}

OPEN command

Description Opens a library for modification.

Syntax

\texttt{OPEN LIBRARY \{directory\}lib-name\{extension\}}

\texttt{[FILESYSTEM | DB]}

Keywords and Values

\begin{itemize}
  \item \texttt{LIBRARY name} Specifies the name of the library, including the optional directory path and extension if stored in the file system.
  \item \texttt{FILESYSTEM} or \texttt{DB} Specifies whether the specified library is stored in the file system or in the currently connected database. The default is FILESYSTEM.
\end{itemize}

Comments If neither FILESYSTEM nor DB is specified, Procedure Builder tries to access the specified library first in the file system and then, if unsuccessful, in the current database.

If you attempt to open a library in the file system, the load path and the extension \texttt{.pll} are used when resolving \texttt{lib-name}, unless \texttt{directory} and \texttt{extension} are specified explicitly. Note that the syntax of \texttt{directory} is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

OPEN command examples

The following command opens the library named \textit{lib1}, which is stored in the file system.

\texttt{.OPEN LIB /private/libs/lib1}

The following command opens the library named \textit{libdb}, which is stored in the current database:

\texttt{.OPEN LIB libdb DB}

RENAME (libraries) command

Description Renames a library that resides in the current database.
Syntax
```
RENAME LIBRARY oldname TO newname
```

Keywords and Values
- **LIBRARY** `oldname` Specifies the current library name.
- **TO** `newname` Specifies the new library name.

Comments
You cannot rename a library to the name of a library that is currently attached. Use the DETACH command to detach the library specified by the new name, or use a different new name.
You cannot use this command to rename libraries stored in files. You must use operating system commands.

**RENAME (libraries) command example**
The following command renames database library `lib1` to `lib4`:
```
.RENAME LIB lib1 TO lib4
```

---

**REVERT command**

Description  Reverts one or more libraries to their previously saved state.

Syntax
```
REVERT LIBRARY name[, name...]
```

Keywords and Values
- **LIBRARY** `name` Specifies one or more open libraries.

**REVERT command example**
The following command reverted the library `lib1`:
```
.REVERT LIB lib1
```

---

**REVOKE command (Library commands)**

Description  Revokes a user's access to a library stored in the database.

Syntax
```
REVOKE LIBRARY name USER name
```

Keywords and Values
- **LIBRARY** `name` Specifies a library.
- **USER** `name` Specifies a user.

Comments  You can specify any single valid user name, or PUBLIC (all users).

**REVOKE command example (Library commands)**
The following command revokes user SCOTT's access to database library `lib1`:
```
.REVOKE LIB lib1 USER scott
```
SAVE command

**Description**  Saves changes made to one or more open libraries.

**Syntax**

```
SAVE LIBRARY name[, name...]
  [AS name]
  [NOPCODE]
  [NODIANA]
  [NOSOURCE]
```

**Keywords and Values**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIBRARY</td>
<td>name</td>
<td>Specifies one or more open libraries.</td>
</tr>
<tr>
<td>AS</td>
<td>name</td>
<td>Specifies a new name for the saved library.</td>
</tr>
<tr>
<td>NOPCODE</td>
<td></td>
<td>Saves the library without the PCODE.</td>
</tr>
<tr>
<td>NODIANA</td>
<td></td>
<td>Saves the library without the DIANA.</td>
</tr>
<tr>
<td>NOSOURCE</td>
<td></td>
<td>Saves the library without the source code.</td>
</tr>
</tbody>
</table>

**Comments**  Saving a library issues an implicit COMMIT. This operation commits any changes made to any database object, not just library objects. For more information on COMMIT, refer to the *Oracle7 SQL Language Reference Manual* or the *Oracle8 SQL Reference Manual*.

You can use NODIANA and NOSOURCE to dramatically reduce the size of a PL/SQL library.

**Note:** Libraries saved with NODIANA and NOSOURCE can be used only in a runtime environment, because it is impossible to compile references to library program units that do not include DIANA.

**SAVE command examples**

- The following command saves the libraries lib1 and lib2:
  ```
  .SAVE LIB lib1, lib2
  ```
- The following command saves library lib1 as library lib1a:
  ```
  .SAVE LIB lib1 AS lib1a
  ```
- The following command saves the libraries lib1 and lib2 without DIANA or source:
  ```
  .SAVE LIB lib1, lib2 NODIANA NOSOURCE
  ```

SHOW (libraries) command

**Description**  Enumerates the libraries that are currently attached.

**Syntax**

```
SHOW LIBRARIES
```

**SHOW (libraries) command example**

- The following command displays the currently attached libraries:
  ```
  .SHOW LIB
  ```
Interpreter Commands

Interpreter commands examples
Bind variable commands examples
Database commands examples
Debug action commands examples
Debugging commands examples
Library commands examples
Load path commands examples
Logging commands examples
Program unit commands examples
Session commands examples

Bind variable commands examples
CREATE (bind variable) command
DELETE (bind variable) command

Database commands examples
CONNECT
DESCRIBE (tables and views)
GRANT
REVOKE
STORE

Debug action commands examples
BREAK
DELETE (debug actions)
DESCRIBE (debug actions)
DISABLE (debug actions)
ENABLE (debug actions)
LIST (debug actions)
SHOW (debug actions)
TRIGGER

Debugging commands examples
DESCRIBE (locals)
GO
RESET
SET
SHOW (call stack)
STEP

Library commands examples
ATTACH
CLOSE
COMPILE (libraries)
CREATE (libraries)
DELETE (libraries)
DELETE (library program unit)
DESCRIBE (libraries)
DETACH
EXPORT (libraries)
GENERATE
GRANT
INSERT (library program unit)
LOAD (library program units)
OPEN
RENAME
REVERT
REVOKE
SAVE
SHOW (libraries)

Load path commands examples
DELETE (load path)
DESCRIBE (load path)
INSERT (load path)

Logging commands examples
DISABLE (logging)
ENABLE (logging)
LOG

Program unit commands examples
COMPILE (program units)
DELETE (program units)
DESCRIBE (program units)
DISABLE (compiler options)
ENABLE (compiler options)
EXECUTE
EXPORT (program units)
EXPORT (stored program units)
LIST (program units)
LOAD (program units)
LOAD (stored program units)
SHOW (locals)
SHOW (program units)

Session commands examples
DESCRIBE (version)
HELP
INTERPRET
Load Path Commands

DELETE (load path) command

Description  Resets the load path to contain no elements.
Syntax

DELETE LOADPATH

DELETE (load path) command example

The following command clears the load path of all entries:

.DELETE LOADPATH

DESCRIBE (load path) command

Description  Displays the current load path.
Syntax

DESCRIBE LOADPATH

DESCRIBE (load path) command example

The following command displays all entries in the current load path:

.DESCRIBE LOADPATH

INSERT (load path) command

Description  Inserts directories into the load path.
Syntax

INSERT LOADPATH
{DIRECTORY path[, path...] | CURRENTDIR}
[BEFORE | AFTER]

Keywords and Values

LOADPATH  Specifies the current load path.
DIRECTORY  Specifies one or more directory paths.
path
CURRENTDIR  Specifies the current directory path.
BEFORE or  Specifies whether the directories should be
AFTER    inserted before or after the existing directories
in the load path, respectively. The default is
AFTER.
Comments   The syntax of path is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

INSERT (load path) command example

In UNIX, the following command appends the directory /usr/plsql to the load path:

```
INSERT LOADPATH DIRECTORY /usr/plsql
```
Logging Commands

DISABLE (logging) command

**Description**  
Suspends logging to the current log file.

**Syntax**  
DISABLE LOGGING

**Comments**  
DISABLE has no effect on logging if no log file has been specified (via the LOG command) or if logging is already disabled.

You can enable disabled logging by using the ENABLE command.

DISABLE (logging) command example

The following command temporarily suspends logging:

```
.DISABLE LOG
```

ENABLE (logging) command

**Description**  
Resumes logging to the current log file.

**Syntax**  
ENABLE LOGGING

**Comments**  
ENABLE has no effect if no log file has been specified via the LOG command, or if logging is already enabled.

You can temporarily disable logging with the DISABLE command.

ENABLE (logging) command example

The following command resumes logging after it has been suspended:

```
.ENABLE LOG
```

LOG command

**Description**  
Saves a transcript of Interpreter input and output to the specified log file.

**Syntax**  
LOG FILE directory\name[extension] [APPEND] [ENABLED | DISABLED] [OFF]

**Keywords and Values**  

| FILE name  | Specifies the log file name and, optionally, the |
directory path and file extension.

**APPEND** Indicates that log output should be appended to the specified file; otherwise, the file is overwritten.

**ENABLED** or **DISABLED** Specify whether logging is initially enabled or disabled, respectively. The default is **ENABLED**.

**OFF** Terminates logging and saves the log file.

**Comments** If unspecified, the log file extension defaults to .log. The syntax of directory is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

If the specified log file does not exist, a new file is created in the specified directory. If no directory path is specified, the log file is created in the current directory.

You can temporarily disable logging and then enable it again using the **DISABLE** and **ENABLE** commands, respectively.

**LOG command examples**

The following command begins logging Interpreter input and output in the file debug.log in the current directory:

```
.LOG FILE debug
```

The following command terminates logging and saves the log file:

```
.LOG OFF
```
Program Unit Commands

**COMPILE (program units) command**

**Description**
Compiles/recompiles the specified program units.

**Syntax**
```
COMPILE PROGRAMUNIT name[, name...]
  [SPECIFICATION | BODY]
COMPILE PACKAGE name[, name...]
  [SPECIFICATION | BODY]
COMPILE SUBPROGRAM name[, name...]
  [SPECIFICATION | BODY]
COMPILE PROCEDURE name[, name...]
  [SPECIFICATION | BODY]
COMPILE FUNCTION name[, name...]
  [SPECIFICATION | BODY]
```

**Keywords and Values**

- **PROGRAMUNIT name**: Specifies one or more program units.
- **PACKAGE name**: Specifies one or more packages.
- **SUBPROGRAM name**: Specifies one or more subprograms.
- **PROCEDURE name**: Specifies one or more procedures.
- **FUNCTION name**: Specifies one or more functions.
- **SPECIFICATION or BODY**: Specifies that either the specification or the body of the program unit be compiled. By default, both are compiled.

**COMPILE (program units) command example**

The following command compiles procedure `proc1` and package `pkg1`:
```
.COMPILE PROG proc1, pkg1
```

**DELETE (program units) command**

**Description**
Deletes one or more program units from the current development session.

**Syntax**
```
DELETE PROGRAMUNIT name[, name...]
  [SPECIFICATION | BODY]
DELETE PACKAGE name[, name...]
```

Oracle Procedure Builder Reference 45
DELETE SUBPROGRAM name [, name...]
(SPECIFICATION | BODY)

DELETE PROCEDURE name [, name...]
(SPECIFICATION | BODY)

DELETE FUNCTION name [, name...]
(SPECIFICATION | BODY)

Keywords and Values

PROGRAMUNI name Specifies one or more program units.

PACKAGE name Specifies one or more packages.

SUBPROGRAM name Specifies one or more subprograms.

PROCEDURE name Specifies one or more procedures.

FUNCTION name Specifies one or more functions.

SPECIFICATION or BODY Specifies that either the specification or the body of the program unit be removed. By default, both are removed.

Comments Once deleted from the current development session, a program unit and any of its subobjects (types, variables, subprograms, etc.) are no longer defined within the development session. Deleting a program unit from the current development session can cause the invalidation of other program units that depend upon it. This is analogous to what happens when the specification of a program unit is changed.

DELETE (program units) command example

The following command deletes the package named p2 from the current development session:

.DELETE PACKAGE p2

DESCRIBE (program units) command

Description Displays detailed information about a specific program unit instance.

Syntax

DESCRIBE PROGRAMUNI name [SPECIFICATION | BODY]
DESCRIBE PACKAGE name [SPECIFICATION | BODY]
DESCRIBE SUBPROGRAM name [SPECIFICATION | BODY]
DESCRIBE PROCEDURE name [SPECIFICATION | BODY]
DESCRIBE FUNCTION name [SPECIFICATION | BODY]

Keywords and Values

PROGRAMUNI name Specifies a program unit.

PACKAGE name Specifies a package.

SUBPROGRAM name Specifies a subprogram.

PROCEDURE name Specifies a procedure.
FUNCTION  name  Specifies a function.

SPECIFICATION  N or BODY  Specifies that either the specification or the body of the program unit be added to the library. The default is SPECIFICATION.

Comments  The information displayed includes the program unit name, its type, its parameters (if any), its external location, whether it is compiled, whether it is open for editing, and cross reference information. In addition, describing a package also indicates whether the package is a built-in program unit, and whether it is an extension to the STANDARD package.

Describing a package specification lists all of the subprograms defined within the specification.

DESCRIBE (program units) command example

The following command displays information about the body of package pkg1:

.DESCRIBE BODY PACKAGE pkg1

DISABLE (compiler options) command

Description  Removes one or more compiler options temporarily.

Syntax

DISABLE COMPILER SIZECHECK

Comments  The SIZECHECK compiler option is automatically disabled for batch compilations. You can reactivate a compiler option using the ENABLE command.

DISABLE (compiler options) command example

The following command temporarily disables the SIZECHECK compiler option:

.DISABLE COMPILER SIZECHECK

ENABLE (compiler options) command

Description  Activates or reactivates one or more compiler options.

Syntax

ENABLE COMPILER SIZECHECK

Comments  You can activate the SIZECHECK compiler option for interactive compilations only. Once enabled, the compiler option remains active until disabled. This option is automatically disabled for batch compilations.

Enable the SIZECHECK option prior to compiling a program unit to raise an alert if the size of a program unit source is approaching an operating system limit for memory allocation. If the size of the source is close to an operating system limit, the compiled state of that source will probably be larger and may exceed the operating system limit.

The SIZECHECK option also raises an alert if the compiled state of the program unit is approaching or exceeds an operating system specific limit for memory allocation.

If the source of the program unit, or the compiled program unit exceeds an operating system-specific memory allocation limit, you may wish to break the program unit into smaller program units.

Check your operating system documentation for the memory allocation limit on your platform.
You can temporarily remove a compiler option using the DISABLE command.

**ENABLE (compiler options) command example**

The following command enables the SIZECHECK compiler option:

```
.DISABLE COMPILER SIZECHECK
```

---

**EXECUTE command (PB standalone only)**

**Description**  
Executes a named anonymous block or a parameterless procedure. This command is valid only when Procedure Builder is invoked as a standalone session.

**Syntax**

```
EXECUTE PROGRAMUNIT name
EXECUTE PROCEDURE name
```

**Keywords and Values**

- **PROGRAMUNIT**  
  - `name`: Specifies a named anonymous block.
- **PROCEDURE**  
  - `name`: Specifies a parameterless procedure.

**Comments**  
Use the EXECUTE command to execute program units and procedures that have no source or that have not been compiled.

**EXECUTE command example**

You create a library named `lib1.pll`. You insert the procedure `x1` into that library using the NOSOURCE and NODIANA keywords. Although the `x1` procedure is not compiled, you can run it with the EXECUTE command as follows:

```
.EXECUTE PROC x1
```

---

**EXPORT (program units) command**

**Description**  
Writes the source of one or more program units to a text file.

**Syntax**

```
EXPORT PROGRAMUNIT name [, name...]
    FILE [directory]name[extension]
    [SPECIFICATION | BODY]
    [NOWARN]
EXPORT PACKAGE name [, name...]
    FILE [directory]name[extension]
    [SPECIFICATION | BODY]
    [NOWARN]
EXPORT SUBPROGRAM name [, name...]
    FILE [directory]name[extension]
    [SPECIFICATION | BODY]
    [NOWARN]
EXPORT PROCEDURE name [, name...]
    FILE [directory]name[extension]
    [SPECIFICATION | BODY]
    [NOWARN]
```
**EXPORT FUNCTION**  
\texttt{name[, name...]}  
\texttt{FILE [directory]name[extension]}  
\texttt{[SPECIFICATION | BODY]}  
\texttt{[NOWARN]}

**Keywords and Values**

- **PROGRAMUNI**  Specifies one or more program units.  \texttt{name}
- **PACKAGE**  Specifies one or more packages.  \texttt{name}
- **SUBPROGRAM**  Specifies one or more subprograms.  \texttt{name}
- **PROCEDURE**  Specifies one or more procedures.  \texttt{name}
- **FUNCTION**  Specifies one or more functions.  \texttt{name}
- **FILE**  Specifies the name of the file, including the optional directory path and extension.  \texttt{name}
- **SPECIFICATION** or **BODY**  Specifies either the specification or the body of the designated program unit(s). By default, both are written to the file.
- **NOWARN**  Suppresses the warning that a built-in program unit has no source for export.

**Comments**  
If you export more than one program unit, Procedure Builder sorts the program units to avoid forward references—that is, each program unit appears after the program unit(s) it references. This enables you to reload exported program units into Procedure Builder using INTERPRET.  

If unspecified, the file extension defaults to .PLD. The syntax of \texttt{directory} is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

Attempting to export a built-in program unit to a text file (e.g., \texttt{.EXPORT PROG * FILE allprogs.txt}) displays a warning in the Interpreter pane (Warning: Program unit <progunit name> has no source to export...). Specifying \texttt{NOWARN} in the command string suppresses the warning.

**EXPORT (program units) command example**

The following command writes the source of procedure \texttt{p1} and function \texttt{f3} to the file \texttt{pl1.pld}:
\texttt{.EXPORT PROG p1,f3 FILE pl1}

**EXPORT (stored program units) command**

**Description**  Writes the source of one or more stored program units to a text file.  

**Syntax**

- \texttt{EXPORT [STORED] PROGRAMUNIT \texttt{schema.name [, schema.name...]} FILE [directory]name[extension]}  
  \texttt{[SPECIFICATION | BODY]}
- \texttt{EXPORT [STORED] PACKAGE \texttt{schema.name [, schema.name...]} FILE [directory]name[extension]}  
  \texttt{[SPECIFICATION | BODY]}
- \texttt{EXPORT [STORED] SUBPROGRAM \texttt{schema.name [, schema.name...]} FILE [directory]name[extension]}  
  \texttt{[SPECIFICATION | BODY]}
- \texttt{EXPORT [STORED] PROCEDURE \texttt{schema.name [, schema.name...]} FILE [directory]name[extension]}  
  \texttt{[SPECIFICATION | BODY]}
EXPORT {STORED} FUNCTION schema.name [, schema.name...] 
FILE [directory]name[extension] 
[SPECIFICATION | BODY]

Keywords and Values

PROGRAMUNI  Specifies one or more program units.
T name

PACKAGE name  Specifies one or more packages.

SUBPROGRAM name  Specifies one or more subprograms.

PROCEDURE name  Specifies one or more procedures.

FUNCTION name  Specifies one or more functions.

FILE name  Specifies the name of the file, including the optional directory path and extension.

SPECIFICATION or BODY  Specifies either the specification or the body of the designated stored program unit(s). By default, both are written to the file.

Comments  If you export more than one stored program unit, Procedure Builder sorts the stored program units to avoid forward references—that is, each stored program unit appears after the stored program unit(s) it references. This enables you to reload exported stored program units into Procedure Builder using IMPORT.

If unspecified, the file extension defaults to .PLD. The syntax of directory is operating system-specific.

For more information about syntax, see the Oracle product documentation for your operating system.

EXPORT (stored program units) command example

The following command writes the source of stored procedure p1 and stored function f3 to the file pl1.pld:

.EXPORT STORED PROG SCOTT.p1,SCOTT.f3 FILE pl1

LIST (program units) command

Description  Displays the specified program unit text and sets the current source location.

Syntax

LIST { [USER schema] PROGRAMUNIT name | PROGRAMUNIT [schema.]name} 
{ _ | PC | SCOPE} 
{ [LINE number] | BODY} 

LIST { [USER schema] PACKAGE name | PACKAGE schema.name} 
{ _ | PC | SCOPE} 
{ [LINE number] | BODY} 

LIST { [USER schema] SUBPROGRAM name | SUBPROGRAM schema.name} 
{ _ | PC | SCOPE} 
{ [LINE number] | BODY} 

LIST { [USER schema] PROCEDURE name | PROCEDURE schema.name} 
{ _ | PC | SCOPE} 
{ [LINE number] | BODY} 

LIST { [USER schema] FUNCTION name | FUNCTION schema.name} 
{ _ | PC | SCOPE}
Keywords and Values

USER *schema*  Specifies a schema name in the database where the stored program unit is located.

PROGRAMUNI T *name*  Specifies a program unit.

PACKAGE *name*  Specifies a package.

SUBPROGRAM *name*  Specifies a subprogram.

PROCEDURE *name*  Specifies a procedure.

FUNCTION *name*  Specifies a function.

.  Specifies the current source location. This is the default.

PC  Specifies the current execution location.

SCOPE  Specifies the current scope location.

LINE *number*  Specifies the line of the program unit that should become the current source location.

SPECIFICATION or BODY  Specifies that either the specification or the body of the program unit be displayed, respectively. The default is SPECIFICATION.

Comments  LIST displays the text of a program unit in the Source pane of the Interpreter. If no line is specified using LINE, the first line of the program unit becomes the current source location. This rule does not apply if you specify ., PC, or SCOPE. Specifying . or PC sets the source location to the current execution location. Specifying SCOPE sets the source location to the current scope location.

Note  PC and SCOPE are useful only when program execution has been interrupted.

LIST (program units) command examples

The following command displays the source text of procedure *p1* and sets the source location to line one:

```plaintext
.LIST PROC p1
```

The following command displays the source text of *p1* and sets the source location to line eighteen:

```plaintext
.LIST PROGRAMUNIT p1 LINE 18
```

The following command sets the source location to the current execution location and displays the source text:

```plaintext
.LIST PC
```

The following command displays the source text in a server-side program unit *my_proc* from the schema owned by user *scott* and retains the current source location:

```plaintext
.LIST USER scott PROC my_proc
```

or

```plaintext
.LIST PROC scott.my_proc
```
LOAD (program units) command

Description Loads one or more program units from the file system.

Syntax

LOAD FILE [directory] name[extension]
[ NOCONFIRM]

Keywords and Values

FILE name Specifies one or more files containing the program unit text.

NOCONFIRM Specifies to redefine an existing program unit without prompting you for confirmation.

Comments Each file you specify must contain the source text of a single program unit. If unspecified, the directory defaults to the current directory, and the file extension defaults to .pls.

The syntax of directory is operating system-specific. For more information about syntax, see the Oracle product documentation for your operating system.

The source text is compiled as it is loaded. If the resulting program unit is a named entity (i.e., a subprogram or package), processing is complete. If the program unit is an anonymous block, it is executed and automatically discarded upon completion.

If you try to load a program unit with the same name and type as an existing program unit, a message box displays, asking if you want to redefine the existing program unit. Specifying NOCONFIRM in the command string suppresses the alert.

LOAD (program units) command example

The following command loads the program units whose source is contained in the files proc1.pls and func2.pls in the current directory:

.LOAD FILE proc1, func2

LOAD (stored program units) command

Description Loads one or more program units stored in the database.

Syntax

LOAD STORED PROGRAMUNIT [owner] name[, [owner] name...]
[ SPECIFICATION | BODY]
[ NOCONFIRM]

LOAD STORED PACKAGE [owner] name[, [owner] name...]
[ SPECIFICATION | BODY]
[ NOCONFIRM]

LOAD STORED SUBPROGRAM [owner] name[, [owner] name...]
[ SPECIFICATION | BODY]
[ NOCONFIRM]

LOAD STORED PROCEDURE [owner] name[, [owner] name...]
[ SPECIFICATION | BODY]
[ NOCONFIRM]

LOAD STORED FUNCTION [owner] name[, [owner] name...]
[ SPECIFICATION | BODY]
[ NOCONFIRM]

Keywords and Values

PROGRAMUNIT Specifies one or more program units, including the optional owner.
PACKAGE name  Specifies one or more packages, including the optional owner.

SUBPROGRAM name  Specifies one or more subprograms, including the optional owner.

PROCEDURE name  Specifies one or more procedures, including the optional owner.

FUNCTION name  Specifies one or more functions, including the optional owner.

SPECIFICATIO N or BODY  Specifies that either the specification or the body of the stored program unit be loaded, respectively. If neither is specified, both are loaded.

NOCONFIRM  Specifies to redefine an existing program unit without prompting you for confirmation.

Comments  The source text is compiled as it is loaded. If the resulting program unit is a named entity (i.e., a subprogram or package), processing is complete. If the program unit is an anonymous block, it is executed and automatically discarded upon completion.

If you try to load a program unit with the same name and type as an existing program unit, a message box displays, asking if you want to redefine the existing program unit. Specifying NOCONFIRM in the command string suppresses the alert.

LOAD (stored program units) command example

The following command loads the program units whose source is contained in the files proc1.pls and func2.pls in the current directory:

LOAD STORED PROG scott.proc1, scott.func2

SHOW (locals) command

Description  Lists the current local variables and parameters that are defined at the current scope location.

Syntax

SHOW LOCALS
SHOW PARAMETERS
SHOW VARIABLES

Keywords and Values

LOCALS  Specifies all parameters and variables.
PARAMETERS  Specifies all parameters.
VARIABLES  Specifies all variables.

SHOW (locals) command example

The following command displays information about all of the current parameters:

SHOW PARAMETERS
SHOW (program units) command

Description  Enumerates the program units that are currently defined in the development session.

Syntax

SHOW PROGRAMUNITS
  [USER | BUILTIN]
  [SPECIFICATION | BODY]
SHOW PACKAGES
  [USER | BUILTIN]
  [SPECIFICATION | BODY]
SHOW SUBPROGRAMS
  [USER | BUILTIN]
  [SPECIFICATION | BODY]
SHOW PROCEDURES
  [USER | BUILTIN]
  [SPECIFICATION | BODY]
SHOW FUNCTIONS
  [USER | BUILTIN]
  [SPECIFICATION | BODY]

Keywords and Values

PROGRAMUNITS  Specifies all program units.
PACKAGES     Specifies all packages.
SUBPROGRAMS  Specifies all subprograms.
PROCEDURES   Specifies all procedures.
FUNCTIONS    Specifies all functions.
USER or      Specifies whether to show user-defined or
BUILTIN      built-in program units, respectively. The
default is USER.
SPECIFICATION Dictate whether specifications or bodies are
  listed, respectively. By default, both are
  listed.

SHOW (program units) command examples

The following command lists the names and types of all current user-defined program units:

.SHOW PROGRAMUNITS

The following command lists all of the built-in package specifications:

.SHOW PACK SPEC BUILT
Session Commands

DESCRIBE (version) command

Description Displays detailed information about the current version of Procedure Builder and the PL/SQL compiler.

Syntax

DESCRIBE VERSION

DESCRIBE (version) command example

The following command displays information about Procedure Builder:

.DESCRIPTION VER

HELP command

Description Provides descriptions and syntax summaries for commands.

Syntax

HELP [COMMAND name] [SYNTAX]

Keywords and Values

COMMAND Specifies the Procedure Builder command.
name

SYNTAX Displays the syntax of the specified command.

Comments If no command name is supplied, a list of all Procedure Builder commands is displayed.

HELP command examples

The following command displays a brief description and the syntax of the BREAK command:

.HELP COM BREAK SYNTAX

The following command lists all of Procedure Builder commands:

.HELP

INTERPRET command

Description Executes one or more Procedure Builder scripts.

Syntax

INTERPRET FILE name[, name...] [ECHO] [SILENT]
Keywords and Values

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE name</td>
<td>Specifies a file containing a Procedure Builder script.</td>
</tr>
<tr>
<td>ECHO</td>
<td>Displays each line of the script as it is processed.</td>
</tr>
<tr>
<td>SILENT</td>
<td>Suppresses status messages issued by the Interpreter.</td>
</tr>
<tr>
<td>NOCONFIRM</td>
<td>Specifies to redefine an existing program unit without prompting you for confirmation.</td>
</tr>
</tbody>
</table>

Comments

A Procedure Builder script consists of a sequence of constructs that can be any combination of program units, Procedure Builder commands, and SQL statements. The script is processed as if its contents had been typed directly into the Interpreter. Each PL/SQL construct found in the script is processed as if it had been loaded individually by the LOAD command.

If unspecified, the file extension(s) default to .pld.

If you try to load a program unit with the same name and type as an existing program unit, a message box displays, asking if you want to redefine the existing program unit. Specifying NOCONFIRM suppresses the alert.

You can include SQL statements in your script, but SQL*Plus statements and syntax are not supported.

INTERPRET provides a mechanism for loading multiple program units from a single file. However, INTERPRET lacks the performance of LOAD because Procedure Builder must preparse the source text and send program units to the PL/SQL compiler one at a time.

**INTERPRET command example**

The following command interprets (with ECHO enabled) the script in the file named `script1`:

`.INTERPRET FILE script1 ECHO`

**QUIT command (Procedure Builder standalone only)**

**Description**

Exits the current Procedure Builder session. This command is valid only when Procedure Builder is invoked as a standalone session.

**Syntax**

`QUIT [NOCONFIRM]`
Index

A
ATTACH command 27

B
BREAK command: 14

C
CLOSE command: 28
COMPILE command
  COMPILE (libraries) command 28
  COMPILE (program units) command 45
COMPILE command: 28, 45
CONNECT command: 10
CREATE command
  CREATE (bind variable) command 8
  CREATE (libraries) command 29
CREATE command: 8, 29
database connection
  CONNECT command 10
DELETE command
  DELETE (bind variables) command 9
  DELETE (debug actions) command 16
  DELETE (libraries) command 29
  DELETE (library program units) command 30
  DELETE (load path) command 41
  DELETE (program units) command 45
DELETE command: 9, 16, 29, 30, 41, 45
DESCRIBE command
  DESCRIBE (debug actions) command 16
  DESCRIBE (libraries) command 31
  DESCRIBE (load path) command 41
  DESCRIBE (locals) command 22
  DESCRIBE (program units) command 46
  DESCRIBE (tables and views) command 10
  DESCRIBE (version) command 55
DESCRIBE command: 10, 16, 22, 31, 41, 46, 55
DETACH command: 31
DISABLE command
  DISABLE (compiler options) command 47
  DISABLE (debug actions) command 17
  DISABLE (logging) command 43
DISABLE command: 17, 43, 47
DISCONNECT command: 11

E
ENABLE command
  ENABLE (compiler options) command 47
  ENABLE (debug actions) command 17
  ENABLE (logging) command 43
ENABLE command: 17, 43, 47
text
  Interpreter commands 38
  examples: 38
EXECUTE command: 48
EXPORT command
  EXPORT (libraries) command 31
  EXPORT (program units) command 48
  EXPORT (stored program units) command 49
EXPORT command: 31, 48, 49
GE
GENERATE command: 32
GO command: 22
GRANT command: 11, 32

H
HELP command: 55

I
INSERT command
  INSERT (library program units) command 33
  INSERT (load path) command 41
INSERT command: 33, 41
INTERPRET command: 55
Interpreter commands
  all6
  examples: 38
  for bind variables: 4
  for debug actions: 4
  for debugging: 5
  for libraries: 5
  for logging: 6
  for program units: 6
  for sessions: 6
  for the database: 4
  for the load path: 5
  using: 1
Interpreter commands: 1, 4, 5, 6, 38

L
LIST command
  LIST (debug actions) command 18
  LIST (program units) command 50
LIST command: 18, 50
LOAD command
LOAD (library program units) command 34
LOAD (program units) command 52
LOAD (stored program units) command 52
LOAD command: 34, 52
LOG command
   LOG command 43
   LOG command: 43

OPEN command: 35

PL/SQL Interpreter commands
   see Interpreter commands 1
   PL/SQL Interpreter commands: 1

QUIT command: 56

RENAME (libraries) command 35
RESET command: 23
REVERT command: 36
REVOKE command: 11, 36

SAVE command: 37
SET command: 23
SHOW command
   SHOW (call stack) command 24
   SHOW (debug actions) command 18
   SHOW (libraries) command 37
   SHOW (locals) command 53
   SHOW (program units) command 54
SHOW command: 18, 24, 37, 53, 54
STEP command
   STEP commands 24
STEP command: 24
STORE command: 12
We Appreciate Your Comments

Reader’s Comment Form - A66242-02

Oracle Corporation welcomes your comments about this manual’s quality and usefulness. Your feedback is an important part of our revision process.

• Did you find any errors?
• Is the information presented clearly?
• Are the examples correct? Do you need more examples?
• What features did you like?

If you found any errors or have any other suggestions for improvement, please write the topic, chapter, and page number below:

___________________________________________________________________________________
___________________________________________________________________________________
___________________________________________________________________________________
___________________________________________________________________________________
___________________________________________________________________________________
___________________________________________________________________________________

Please send your comments to:

Oracle Developer Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Or send comments by e-mail to: oddoc@us.oracle.com

Please include your name, address, and telephone number for a reply:

___________________________________________________________________________________
___________________________________________________________________________________
___________________________________________________________________________________

Thank you for your help.