Ora*Tst Reference Manual

V3.6.0.3.0

Table of Contents

1 Introduction

Ora*Tst is an internal software testing tool used by base development and product line organizations at Oracle Corporation. Testing software with Ora*Tst consists of exercising the software on well defined input. The output that the software generates must be deterministic. It is important to note that within this paradigm, the correctness of the software being tested can be ascertained simply by comparing generated output with reference output, for the same input.

Ora*Tst specifies a script language and provides a test script interpreter. As much as possible, Ora*Tst is strictly based on portable concepts. This allows for the generation of portable tests which can be executed without modification on the base development platform and on any other target platform. The only prerequisites are that the scripts be written in a portable fashion and that Ora*Tst be ported to these platforms.

Ora*Tst is geared towards automated, easily repeatable testing. Tests implemented in Ora*Tst consist of test scripts, input and reference output files, all of which are source controlled and maintained along with the respective product by the base development organization. A test script is a text file containing one command per line defining a sequence of actions to be taken to execute a test. Ora*Tst interprets the test scripts at runtime.

This manual provides detailed descriptions of each Ora*Tst command including command usage and syntax, options and restrictions, availability (i.e., which Ora*Tst version the command first appeared in), and known bugs. It is not intended to be a learning tool. It is a reference manual, which will be most useful if you are already familiar with Ora*Tst. If you are new to Ora*Tst, we recommend that you read the *Ora*Tst Installation and User's Guide* first.

2 Manual Pages

2.1 ADD

2.1.1 Synopsis

```
ADD [LOCAL|GLOBAL] <varname> <addend> [<target_var>] [log|nolog]
```

2.1.2 Description

Adds <addend> to <varname> and stores the result in <varname>. If variable <target_var> is specified it is updated with the result and <varname> is left unchanged. The <varname> must be an Ora*Tst variable and <addend> is a literal value or dereferenced variable. The <varname> value and <addend> must be integers in the range of signed 32bit values.

If <varname> does not exist it is created with an initial value of 0 and a default scope of global.

If <target_var> is specified but does not exist it is created in the same scope as <varname>.

2.1.3 Options

LOCAL| GLOBAL

LOCAL: If local is specified, the selection and/or creation of <varname> and optionally <target_var> is local scope only.

GLOBAL: If global is specified, the selection and/or creation of <varname> and optionally <target_var> is global scope only.

If neither local or global is specified, selection of <varname> follows normal scoping rules.

target_var

Optional variable name that is updated with the result and <varname> is left unchanged. If it does not exist it is created in the same scope as <varname>.

log|nolog

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.1.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.1.5 Prerequisites

The <varname> value and <addend> must be valid signed 32bit integers.

If the result exceeds the value of a signed 32bit integer a "TST*WRN-66" message is issued.

2.1.6 See Also

subtract, multiply, divide, modulo, incr and decr Ora*Tst commands.

2.1.7 Known Bugs

None.

2.2 BLOCK_BEGIN

2.2.1 Synopsis

```
block_begin
```

2.2.2 Description

Marks the beginning of a block of code in which the BLOCK_BREAK command may appear.

The Block related commands are intended to provide a mechanism to break out of code paths that may be nested within loops. The primary goal is to make it easier for Test Developers to handle error conditions.

2.2.3 Options

None.

2.2.4 Notes

Each BLOCK_BEGIN command must have a matching BLOCK_END command. Nested BLOCK_BEGIN / BLOCK_END pairs are allowed.

2.2.5 Availability

Ora*Tst built-in command since V3.5.5.4.0.

2.2.6 Prerequisites

This command cannot be coded within a multi-line ECHO command.

The matching BLOCK_END command and optional BLOCK_BREAK command(s) must be coded in the same .tsc file, whether that is a test script, macro or INCLUDEd file.

2.2.7 See Also

block_break, block_end Ora*Tst commands.

2.2.8 Known Bugs

None.

2.3 BLOCK_BREAK

2.3.1 Synopsis

```
block_break
```

2.3.2 Description

Used to break out of the current block of code defined by the BLOCK_BEGIN / BLOCK_END command pair. Normal command execution flow is interrupted and resumes at the first statement following the closest matching BLOCK_END command.

The Block related commands are intended to provide a mechanism to break out of code paths that may be nested within loops. The primary goal is to make it easier for Test Developers to handle error conditions.

2.3.3 Options

None.

2.3.4 Notes

Multiple BLOCK_BREAK commands can be coded within a block.

2.3.5 Availability

Ora*Tst built-in command since V3.5.5.4.0.

2.3.6 Prerequisites

This command cannot be coded within a multi-line ECHO command.

The BLOCK_BREAK command can only be specified within matching BLOCK_BEGIN / BLOCK_END commands and must be coded in the same .tsc file, whether that is a test script, macro or INCLUDEd file.

2.3.7 See Also

block_begin, block_end Ora*Tst commands.

2.3.8 Known Bugs

None.

2.4 BLOCK_END

2.4.1 Synopsis

```
block_end
```

2.4.2 Description

Marks the end of a block of code in which the BLOCK_BREAK command may appear. If a BLOCK_BREAK command is executed within the block, script execution continues at the first statement following this command.

The Block related commands are intended to provide a mechanism to break out of code paths that may be nested within loops. The primary goal is to make it easier for Test Developers to handle error conditions.

2.4.3 Options

None.

2.4.4 Note

Each BLOCK_END command must have a matching BLOCK_BEGIN command. Nested BLOCK_BEGIN / BLOCK_END pairs are allowed.

2.4.5 Availability

Ora*Tst built-in command since V3.5.5.4.0.

2.4.6 Prerequisites

This command cannot be coded within a multi-line ECHO command.

The matching BLOCK_BEGIN command and optional BLOCK_BREAK command(s) must be coded in the same .tsc file, whether that is a test script, macro or INCLUDEd file.

2.4.7 See Also

block_begin, block_break Ora*Tst commands.

2.4.8 Known Bugs

None.

2.5 BREAK

2.5.1 Synopsis

```
 (1)
loop
   ...
   break
   ...
endloop
 (2)
for <var> <starting value> <ending value> [<step factor>]
   ...
   break
   ...
endloop
```

2.5.2 Description

The BREAK Ora*Tst command breaks out of a loop...endloop construct or a for...endloop construct and continues the flow of control with the first statement after the ENDLOOP command.

2.5.3 Options

None.

2.5.4 Availability

Ora*Tst built-in command since V1.0.6.

2.5.5 Prerequisites

Must be used only within a loop...endloop or a for...endloop construct.

2.5.6 See Also

loop, for, endloop, continue and block_break Ora*Tst commands.

2.5.7 Known Bugs

None.

2.6 CLEANOUT

2.6.1 Synopsis

```
cleanout <area> [<file>.<type>]
```

2.6.2 Description

Removes all files in the specified area <area> except for the file <file> of type <type>, if specified. The area itself is not deleted. Note that the second argument does not allow the specification of an area, since that would be redundant with the area specification <area>, the first argument.

Note that since the first argument is already interpreted as an area name, it should not be terminated with a `:', i.e. use `T_WORK', not `T_WORK:'.

2.6.3 Options

None.

2.6.4 Availability

Ora*Tst built-in command since V1.0.6.

2.6.5 Prerequisites

The area <area> must be defined in the environment.

2.6.6 See Also

MKDIR command.

For more detailed information on Ora*Tst areas, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

For more information on the dynamic creation of areas, refer to the Ora*Tst command mkdir.

2.6.7 Known Bugs

None.

2.7 CLONEDB (macro)

2.7.1 Synopsis

```
clonedb [CLONESRC=<src_num>] CLONEDST=<dest_num>
        [SDB_UP=<UP|DOWN>] [DDB_UP=<UP|DOWN>] [MODE=TKMODE]
```

2.7.2 Description

The CLONEDB command will create a target DB in the T_DBS area. It will do this by either restoring a pre-existing copy of the DB from the T_HIST area or by “cloning” the source DB if there is not a pre-existing target DB. “Cloning” involves using the data and log files from the source DB to create the corresponding target DB files. A parameter file (init.ora) for the target DB will be generated from the source DB parameter file.

Currently, this command only supports the mode TKMODE. This mode can be used by the RDBMS tests that use tkstart.tsc and stkstart.tsc for setting up its configuration variables.

Here are the steps that CLONEDB makes:

CLONEDB will create the destination DB in the T_DBS area

for an existing destination DB in T_HIST

- copy destination DB files from T_HIST to T_DBS

- create an ‘init.ora’ file for the destination DB from the source ‘init.ora’

or for a new destination DB

- copying the source DB data and log files from either T_DBS or T_HIST to T_DBS.

- creating a new ‘init.ora’ file from the source ‘init.ora’ by doing string replacements on all of the source specific entries with the new destination entries.

- start up the new destination DB using the new ‘init.ora’ in order to create a new control file and to update any source specific entries in the DB (for example database links).

- shutdown the database and save destination DB files in T_HIST.

CLONEDB will check the SDB_UP and DDB_UP flag to see what state to leave source and destination DB.

The <src_num> and <dest_num> currently use numbers to specify the database (‘1’, ‘2’, etc...) The database number is used to construct variable names that contain the real names of the database files. See the MODE section for the expected variable names.

2.7.3 Options

CLONEDST=<dest_num> (required)

New database “number”. This number is used to look up the name of the database and the files associated with that database.

CLONESRC=<src_num>

Existing database “number”. Default is 1. This number is used to look up the name of the database and the files associated with that database.

MODE=TKMODE

Default is TKMODE (for test kernel mode which is currently the only mode). Specifies that naming of the database files is in the format used by the test kernel. So, names for everything is specified in tkstart.tsc and stkstart.tsc. Also, macros defined in the test kernel are used (database, startup, shutdown, etc...)

The mode can be set via the global variable “tst_clonedb_mode”. See modes section for more information.

SDB_UP=<UP|DOWN>

flag for specifying if source DB is left in up or down state. Default is up.

DDB_UP=<UP|DOWN>

flag for specifying if destination (new) DB is left in up or down state. Default is up.

2.7.4 Notes

A default version of this macro is provided by Ora*Tst. It is common to find this macro overridden in the RDBMS Server environment. Thus, the options and behavior of the overriding macro may differ from that described here.

2.7.5 Database Areas

T_DBS

Area where database files are stored when used for tests

T_HIST

Archive area for previously created databases

2.7.6 Modes

TKMODE

Test Kernel Mode, currently the only mode. This mode supports the RDBMS tests that uses variables defined in tkstart.tsc and stkstart.tsc to determine the names of the log, data, control, and init files for each database. Uses the following macros:

database

dbcopy

startup reuse

shutdown

restoredb

Uses the utility:

svrmgrl

Expects to be run after rdbmsini which will create:

loop.sql

dbs1.sql

dbs2.sql

dbs3.sql

dbs4.sql

Variables from tkstart.tsc and stkstart.tsc which identify the files associated with the database numbers:

dbs^dbnum^_name

dbs^dbnum^_datafile_0_1_name

dbs^dbnum^_datafile_0_2_name

dbs^dbnum^_logfile_1_1_name

dbs^dbnum^_logfile_1_2_name

t_logfsize1

t_logfsize2

maxlogfiles

maxlogmembers

maxinstances

2.7.7 Warnings

Since the destination DB is a “copy” of the source DB, certain things such as

character set

data file size

log file size

need to be the same between the databases. There may be other things that have not been listed here. Clonedb is better suited for making copies of databases with new names. For databases that are very similar, the user can create the first DB, run clonedb to get a copy, and then run additional set ups for differences between the DBs.

2.7.8 Availability

The command is an Ora*Tst macro since V3.1.0.

2.7.9 Potential changes in future versions

Currently, this command is very specific to the RDBMS testing environment.

2.7.10 Prerequisites

The macro is defined in ctminit.tsc.

2.7.11 See Also

For more information on the implementation of the portable Ora*Tst default macro, refer to the file clonedb.tsc.

2.7.12 Known Bugs

None.

2.8 COLON-IF

2.8.1 Synopsis

```
<global var>:
   ...
endif
```

2.8.2 Description

The undefined global variable <global var> will be implicitly defined and evaluate to TRUE the first time the colon-if construct is encountered with that variable <global var>.

Subsequently, the same colon-if construct will evaluate to FALSE and the body will not be executed again.

This construct was designed for the conditional execution of an initialization command block which allows test scripts to execute both as part of a test suite (i.e., the conditional command block is only executed the first time) and also stand-alone (i.e., the conditional command block is executed, since it is encountered only once).

2.8.3 Options

None.

2.8.4 Availability

Ora*Tst built-in command since V1.0.6.

2.8.5 Prerequisites

No local variable by the same name must be defined. No global variable by that name must be explicitly defined via the set or the unset Ora*Tst commands.

If a global (or local) variable is explicitly defined via the Ora*Tst set or unset commands, the colon-if construct's "once-only" behavior will be lost, and the block will degenerate to an "if <var>...endif" type construct.

2.8.6 See Also

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

if, endif Ora*Tst Commands.

2.8.7 Known Bugs

None.

2.9 COMMENT

2.9.1 Synopsis

```
# { <comment> } ...
... # { <comment> } ...
```

2.9.2 Description

If the line begins with a # sign, the entire line in the test script is interpreted as a comment and is not executed.

If a # sign is encountered on a command line, the remainder of the command line is interpreted as comment and disregarded.

To include a `#' character in, say, an ECHO command, use the Ora*Tst quote character to escape it:

```
echo '#' this is not a comment, but a phone '# 555-1212'
```

2.9.3 Options

None.

2.9.4 Availability

Ora*Tst built-in command V1.0.6.

2.9.5 Prerequisites

None.

2.9.6 See Also

For more information on commenting test scripts, refer to the Oracle Testing Standards (Test Documentation).

2.9.7 Known Bugs

None.

2.10 COMPARE (macro)

2.10.1 Synopsis

```
compare <file1>[.<type1>] [<reffile>[.<reftype>]] [NOMASK|MASK]
        [CASE|NOCASE]
```

2.10.2 Description

Performs a line-based comparison of the two files specified using the Ora*Tst LDIFF command. The area of the first file <file1> specified is already implicitly T_WORK, while the area of the second file <file2>, also called the reference file, is implicitly T_SOURCE. Therefore, you cannot supply an area name for either file parameter. Use the LDIFF command directly if you wish to compare files in other areas.

If only one file is specified, the name and type for the reference file are assumed to be identical to the first file. If a type is omitted, it defaults to LOG.

The comparison is case-sensitive by default (see the NOCASE option below.)

If the comparison succeeds, i.e. no differences are found, an empty SUC file with the name <file1> is created in T_WORK; if the comparison fails, the differences are reported in a (non-empty) DIF file with the name <file1> in T_WORK. Existing SUC and/or DIF files with name <file1> will be deleted and/or overwritten, if they exist.

After the comparison the file <file1> of type <type1> is copied to the area T_HIST (for future reference). The result of the comparison, i.e. success or failure, is recorded in the TLG file as well.

Note: The default macro acts as a wrapper that calls the Ora*Tst LDIFF command. However, since the reference file parameter is optional, the generated file parameter is the first parameter to the macro, an opposite order from the LDIFF command. The macro takes care of re-ordering the parameters for you. However, if you actually try to read the resulting DIF file, the “arrows” (`<‘ and `>’) will seem to point in the wrong direction from the parameters you supplied to the COMPARE macro.

2.10.3 Options

NOMASK (default)

If specified, Ora*Tst simply reports the differences in a DIF file.

MASK

If specified, Ora*Tst attempts to resolve the differences reported in the DIF file, if any, by interpreting the lines from reference file <reffile> of <reftype> as regular expressions and trying to match them against the corresponding lines in file <file1> of type <type1>. If all differences are resolved, the DIF file is renamed to a SUC file. See the "Regular Expressions" section in the Ora*Tst Installation and User’s Guide for details on the masking characters.

CASE (default)

If specified, files are compared in a case-sensitive manner.

NOCASE

If specified, all characters are lowercased before comparison, making the comparison case-insensitive. Note that this is a simple lowercasing; currently, Ora*Tst is not NLS-capable.

2.10.4 Notes

This reference describes the default version of the COMPARE macro provided by Ora*Tst. The default macro is a simple wrapper for the Ora*Tst LDIFF command and is not intended to support all LDIFF options.

It is very common to find a customized version of this macro in the RDBMS Server and other product environments that overrides the default version. Thus, the options and behaviour of the COMPARE macro in your environment **may differ significantly** from that described here. As with the default COMPARE, customized versions almost always use the Ora*Tst LDIFF command for the actual file comparison. See the Ora*Tst LDIFF command for more information.

2.10.5 Availability

The command is an Ora*Tst macro since V1.0.6. There is an automatic default macro definition installed since V2.0.

As mentioned in the Notes section, the COMPARE macro might be part of a product-specific test suite; in that case the custom macro definition will override the default definition. If you are getting confusing results, check the custom macro first, or try using LDIFF directly.

2.10.6 Prerequisites

Defined automatically at Oratst Startup. The default source file is ctm00.tsc.

2.10.7 See Also

For compare masking information, refer to the Ora*Tst Macros and Regular Expressions sections of the Ora*Tst Installation and User's Guide.

For more information on the implementation of the portable Ora*Tst default macro, refer to the file ctm00.tsc.

ldiff Ora*Tst command.

2.10.8 Known Bugs

Refer to the commands that implement the macro, in particular refer to the Ora*Tst LDIFF command.

2.11 COMPILE (OSD macro)

2.11.1 Synopsis

```
compile [NODBUG|DEBUG|PCA] [LINK|NOLINK] [LIB=<libname>]
        [EXESPEC=[<area>:]<name>[.exe]] {
        <srcfile>[.c|.fo|.ada|.co|.obj] } ...
```

2.11.2 Description

Compiles specified sources, possibly links them against specified objects and optionally to one library, and generates an executable image in the area T_EXE. The default filetype is .c. The important point is that upon successful completion of the COMPILE command, it will be possible to invoke the executable with the EXECUTE command using the implicitly or explicitly specified name. If no name is explicitly provided for the executable, the name of the last file provided as a parameter is used as the name for the executable.

The COMPILE macro is mainly provided for the testing of the internal CORE product and of the ORACLE Precompiler products (Pro*ADA, Pro*C, Pro*Fortran and Pro*Cobol); in these cases, the COMPILE command is necessary due to the nature of the products being tested. Tests of other products should, if possible, not compile but simply require that the respective utilities be present in the appropriate area (e.g. T_EXE, T_SYSTEM). For more information on testing standards, refer to the “Oracle Testing Standards” (OTS) document.

2.11.3 Options

It is important to note that the test results should only depend on the following two guaranteed and portable options to the COMPILE macro:

EXESPEC

If specified, names the executable to be generated; otherwise the name of the last file listed is used as name for the executable.

LIB

If specified, the compiled sources are linked against that library; the location of the library is port-specific. The specified library must have portable semantics and must be available with the test suite.

However, in order to allow for maximal flexibility and convenience, it is allowed to specify additional port-specific and even non- portable options, if the underlaying implementation recognizes them and takes appropriate action.

2.11.4 Notes

A default version of this macro is provided by Ora*Tst. It is common to find this macro overridden in the RDBMS Server environment. Thus, the options and behavior of the overriding macro may differ from that described here.

2.11.5 Availability

It was an Ora*Tst built-in command in V1.0.6. The command has been an Ora*Tst macro since V2.0.

Ora*Tst only provides and maintains the portable interface to the Ora*Tst macro. The macro must be ported and/or customized by each development group. Compilation is highly port-specific and environment specific.

2.11.6 Prerequisites

A customized version of the COMPILE macro must be available in the environment. The COMPILE macro must be defined using the defmacro command.

2.11.7 See Also

For more detailed information on macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User's Guide. Specifically regarding the COMPILE macro, see the file sctm31.tsc.

2.11.8 Known Bugs

None.

2.12 CONTINUE

2.12.1 Synopsis

```
 (1)
loop
   ...
   continue
   ...
endloop
 (2)
for <var> <starting value> <ending value> [<step factor>]
   ...
   continue
   ...
endloop
```

2.12.2 Description

The CONTINUE Ora*Tst command will restart a loop/for and start the next iteration. The flow of control continues with the first statement in the loop/for body.

2.12.3 Options

None.

2.12.4 Availability

Ora*Tst built-in command since V1.0.6.

2.12.5 Prerequisites

Must be used only within a loop...endloop construct or a for...endloop construct.

2.12.6 See Also

loop, for, endloop, break and block_break Ora*Tst commands.

2.12.7 Known Bugs

None.

2.13 CPFILE

2.13.1 Synopsis

```
cpfile [<srcarea>:]<srcfile>.<srctype>
       [<tgtarea>:][<tgtfile>.<tgttype>]
```

2.13.2 Description

Copies a file <srcfile> of type <srctype> in area <srcarea> to a file <tgtfile> of type <tgttype> in area <tgtarea>. The default area for both area <srcarea> and area <tgtarea> is the working area T_WORK.

The specification of a target area only is sufficient (i.e., `T_WORK:'.) The <tgtarea> and <tgttype> will then default to match that of the source file.

The protection mode of the target file and the source file are the same, with the exception that the target file will always be writable by the owner.

2.13.3 Options

None.

2.13.4 Notes

For portability reasons the source file should be a text file.

The target file is overwritten if it exists.

The specification of any portion of a platform specific path component (i.e. dirname/filename) for <srcfile> or <tgtfile> is not portable and is not supported.

Source or target file names longer than 128 bytes are truncated without warning.

2.13.5 Availability

Ora*Tst built-in command since V1.0.6.

2.13.6 Prerequisites

The file <srcfile> of type <srctype> in area <srcarea> must exist.

2.13.7 See Also

For more detailed information on file specifications, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

mvfile, rmfile and findfile Ora*Tst commands.

2.13.8 Known Bugs

Generic bug (#): Copying non-text files (binary) files might cause portability problems on some platforms.

2.14 DECR

2.14.1 Synopsis

```
decr [GLOBAL|LOCAL] <var> [<step>] [nolog]
```

2.14.2 Description

Decrements the variable <var> by 1, or the amount of <step> if specified. <step> is a literal numeric value that can be negative.

2.14.3 Options

LOCAL

If specified, the variable <var> is interpreted as a local variable only.

GLOBAL

If specified, the variable <var> is interpreted as a global variable only. (V3.0)

step

A numeric literal value to decrement <var>. The default is 1.

nolog

The optional nolog parameter indicates to not log the command execution in the Ora*Tst log (.tlg) file. The default is to log the command.

2.14.4 Availability

The command is an Ora*Tst built-in command in V2.1, and the GLOBAL and auto-scoping features were added in V3.0.

The nolog option is available at V3.5.5.3.0.

2.14.5 Prerequisites

The variable <var> must have a string value that can be interpreted as an integer. The range of legal values is -2^31 to 2^31 - 1 (signed 32-bit integer.) If no variable scope option is specified, the command will “automatically” determine the scope for you by looking for a variable of that name starting at the local scope, and if not found will look for a global variable. The variable will be set as the same type found. If <var> does not exist, an error message will be issued.

Remember that a local variable will hide a global variable of the same name, according to scoping rules. You can use the GETGLOBAL command to bypass the scoping rules and retrieve a hidden global.

2.14.6 See Also

subtract, incr, add, getglobal Ora*Tst commands.

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User’s Guide.

2.14.7 Known Bugs

None.

2.15 DEFMACRO

2.15.1 Synopsis

```
defmacro <name> [[<area>:]<scr name>[.TSC]] [ASIS]
```

2.15.2 Description

Defines a macro command <name> which can subsequently be invoked like any other Ora*Tst command. Macros are implemented via Ora*Tst scripts.

Unless explicitly specified otherwise in the second argument <area>:<src name>[.TSC], the macro is defined in a TSC file with the same name <name> as the macro command. The default area for macro definitions is T_SOURCE.

Macro definitions can be nested. In that case, the latest definition overwrites previous ones, such as default definitions. Upon exiting a level of nesting, the previous definition, if any, comes again into effect. Beware of recursion.

2.15.3 Options

[ASIS]

If specified, then there will be no special processing of the macro arguments. This allows characters such as `=' and `>' to be passed as arguments to the macro.

This option was added in V3.2.3.

2.15.4 Notes

The maximum macro recursion level is 18. This value includes nesting of run/runtest. Thus, the limit is the combined nesting level of run/runtest and macro recursion. In versions prior to V3.5.5.4.0 the limit was 16. This limit cannot be changed.

The maximum number of macros that can be defined is 1500. The redefinition of a macro is treated like a new macro and counts against the limit. The limit cannot be changed or reconfigured. If more than 1500 macro definitions are needed then the test is likely broken. Consider "fence" variables (similar to what is done for C header files) so macros are not redefined multiple times. For versions prior to V3.5.5.4.0 the limit was 600.

2.15.5 Availability

Ora*Tst built-in command since V1.0.6.

Asis option available at V3.2.3.

2.15.6 Prerequisites

The TSC file implicitly or explicitly referenced in the command must exist when the macro is defined (dynamic scoping).

2.15.7 See Also

For more detailed information on macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User's Guide.

For more information on default definitions for macros, refer to the installed script ctminit.tsc and the respective installed TSC files which implement the default macros.

2.15.8 Known Bugs

None.

2.16 DIVIDE

2.16.1 Synopsis

```
DIVIDE [LOCAL|GLOBAL] <varname> <divisor> [<target_var>]
        [log|nolog]
```

2.16.2 Description

Divides <varname> by <divisor> and stores the integer result in <varname>. If variable <target_var> is specified it is updated with the result and <varname> is left unchanged. The <varname> must be an Ora*Tst variable and <divisor> is a literal value or dereferenced variable. The <varname> value and <divisor> must be integers in the range of signed 32bit values. The result is an integer value truncated towards 0.

If <varname> does not exist it is created with an initial value of 0 and a default scope of global.

If <target_var> is specified but does not exist it is created in the same scope as <varname>.

2.16.3 Options

`LOCAL| GLOBAL`

LOCAL: If local is specified, the selection and/or creation of <varname> and optionally <target_var> is local scope only.

GLOBAL: If global is specified, the selection and/or creation of <varname> and optionally <target_var> is global scope only.

If neither local or global is specified, selection of <varname> follows normal scoping rules.

`target_var`

Optional variable name that is updated with the result and <varname> is left unchanged.. If it does not exist it is created in the same scope as <varname>.

`log|nolog`

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.16.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.16.5 Prerequisites

The <varname> value and <divisor> must be valid signed 32bit integers.

If <divisor> is 0 a "TST*WRN-23" message is issued.

If the result exceeds the value of a signed 32bit integer a "TST*WRN-66" message is issued.

2.16.6 See Also

add, subtract, multiply, modulo, incr and decr Ora*Tst commands.

2.16.7 Known Bugs

None.

2.17 DUMPVAR

2.17.1 Synopsis

```
dumpvar <var>[,<var>] [<out_area>:<outfile>] [LOC_LEVELS]
        [NOFORMAT] [NOHEADING]

dumpvar _GLOBAL_|_LOCAL_|_ALL_
        [<out_area>:<outfile>][LOC_LEVELS][NOFORMAT][NOHEADING]
```

2.17.2 Description

Prints or "dumps" the value of the specified variable(s) for debugging purposes. Do not use this command as part of a regular production test script.

<var> is a single argument indicating the variable name(s) to be dumped. This can be a literal value or de-referenced Ora*Tst variable. It can be a list of variable names separated by commas. No white space is allowed between names. Leading, trailing, and multiple commas are allowed.

GLOBAL indicates to dump all global variables that are currently defined. Mutually exclusive with <var>, _LOCAL_ and _ALL_.

LOCAL indicates to dump all local variables that are currently defined. All local variables for all nested run levels are dumped. Mutually exclusive with <var>, _GLOBAL_ and _ALL_. The LOC_LEVELS parameter is implied.

ALL indicates to dump all global and local variables that are currently defined. Mutually exclusive with <var>, _GLOBAL_ and _LOCAL_. The LOC_LEVELS parameter is implied.

<outfile> is the output file to write the variable data. If specified, it must be the second parameter and be an Ora*Tst portable file specification. The default Area is T_WORK. The output is always APPENDED to this file. If not specified, the default is to write to the test log (tlg) file.

2.17.3 Options

LOC_LEVELS - indicates to dump the local value of the variable(s) for each nested level of Ora*Tst execution. The top most level is 1. The default is to print the local value according to the standard Ora*Tst scoping rules.

NOFORMAT - indicates to not format the output. Only display the variable and its' value(s).

The default is to format the output like:

V:<varname> G:<global_value> L:<local_value>

When <var> and LOC_LEVELS is specified, the format includes the local variable value for each nested level. Level 1 (L1) is the top most parent script:

V:<varname> G:<global_value> L2:<local_value> L1:<local_value>

NOHEADING - indicates to not include a heading. By default, a heading is created which indicates the test script and line number where the DUMPVAR command was issued. Only the current RUN level script name is reported. If DUMPVAR is executed from an INCLUDE'd file or macro, only the parent test run script and line number is reported.

2.17.4 Notes

The DUMPVAR command is for debugging purposes only.

DO NOT include this command in production code. Parameters and/or defaults may change at any time without notice.

Do not use the dump output as input to a separate Ora*Tst execution as an attempt to "clone" or "duplicate" a test run. Doing so is not supported.

Ora*Tst internal system variables are included in the output when using _GLOBAL_, _LOCAL_ and _ALL_. There is no way to restrict or suppress this behavior. In general, the system variables start with "TST_", but there are many exceptions to this rule.

A value of "_UNDEF_" indicates the variable is undefined. A value of "(null)" indicates the variable is defined but null.

Examples:

1. File: sample1.tsc

```
SET foobar 12345
LET foobar 67890
DUMPVAR foobar
```

The .tlg files shows:

```
> DUMPVAR         Issued from T_SOURCE:sample1.tsc:3 on Jan 15 2013
22:30:37
> DUMPVAR          V:foobar G:12345 L:67890
```

2. File: sample2.tsc

```
SET foobar 12345
LET foobar 67890
SET myvar1 AAA
LET myvar2 BBB
DUMPVAR foobar,myvar1,myvar2
```

The .tlg files shows:

```
> DUMPVAR      Issued from T_SOURCE:sample2.tsc:5 on Jan 15 2013
         22:39:49
> DUMPVAR      V:foobar G:12345 L:67890
> DUMPVAR      V:myvar1 G:AAA L:_UNDEF_
> DUMPVAR      V:myvar2 G:_UNDEF_ L:BBB
```

3. File: sample3.tsc

```
SET foobar 12345
LET foobar 67890
SET myvar1 AAA
LET myvar2 BBB
```

```
DUMPVAR _ALL_ T_WORK:dump_all.out
```

The .tlg files shows:

```
> DUMPVAR            Written to file T_WORK:dump_all.out
```

The dump_all.out shows (partial):

```
DUMPVAR Issued from T_SOURCE:sample3.tsc:5 on Jan 15 2013 22:42:52
DUMPVAR GLOBAL
DUMPVAR V:unix G:1
DUMPVAR V:TST_OS G:unix
DUMPVAR V:foobar G:12345
DUMPVAR V:myvar1 G:AAA
DUMPVAR LOCAL[1]
DUMPVAR V:TST_TSCNAME L1:sample3
DUMPVAR V:ARGC L1:0
DUMPVAR V:cnt L1:26
DUMPVAR V:foobar L1:67890
DUMPVAR V:myvar2 L1:BBB
```

2.17.5 Availability

Ora*Tst built-in command since V3.6.0.3.0.

2.17.6 Prerequisites

None.

2.17.7 See Also

getglobal, getlocal, let, set, rmfile Ora*Tst commands.

2.17.8 Known Bugs

None.

2.18 ECHO

2.18.1 Synopsis

```
 (1)
echo { [<string>] | [%t] }
   ...
 (2)
echo > [<area>:]<name>.<type> [APPEND]
   ...
   > { [<string>] | [%t] }...
   ...
endecho
```

2.18.2 Description

The Ora*Tst echo command comes in two flavors: (1) simple line-echo, and (2) multi-line echo. The two are distinguished from each other by the specification of redirection on the echo command line (see Synopsis).

The simple line-echo prints a line of text (interpreting command line arguments as text strings) to the current TLG file, merging it with the regular Ora*Tst logging information.

The multi-line echo allows for writing text into a specified file, instead of just the TLG file. The specified file is created and kept open until the next ENDECHO command is reached. While the specified file is open, all lines starting with the redirection character `>' are written to the file specified by the redirection syntax. If the APPEND option is used, the echoed lines will be appended to the end of the existing file, rather than overwriting the file. If the file does not exist, a new one will be created. Please note that you cannot use the APPEND feature to echo into another currently open TLG file (for example, a FORKed script echoing into its parent's TLG); this behavior is not supported.

If not specified, <area> defaults to T_WORK.

If the string %t is specified in the text to be echoed, it is expanded to the current date/time. If you want the actual string %t to be echoed, specify %%t.

Extra spaces or tabs are compressed into one white space. Note that most other Ora*Tst commands, in particular control structures (excepting control flow commands like INCLUDE), can be embedded in the body of the multi-line echo command. See Prerequisites below.

It is strongly recommended to NOT use the ECHO command to directly echo raw filenames into a file, for logging or parsing (by the shell, SQLDBA, etc.) These filenames are not portable, as they do not go through the Ora*Tst portable file specification mechanism. As of V2.1, you can use the new OSDFILENAME command to use this mechanism and store the translated OSD filename in a variable, and then echo that stored string into the file. This is the recommended, portable solution. If you do not have V2.1, and must echo filenames, please

make them as generic as possible: i.e., only the basename, no pathnames, no extensions if possible (if the product will append the expected extension), etc., and put the reference in a OSD script or macro, of course. Remember that product-line groups could be forced to port hundreds of scripts, if they are not written portably by the base developers.

2.18.3 Options

APPEND (Multi-line echo only)

If specified, echoed lines will be appended to the end of an existing file, rather than overwriting the file. If the file does not exist, a new one will be created. (V3.0) Availability

The command is an Ora*Tst built-in command in V1.0.6. APPEND option added in V3.0.

2.18.4 Notes

Output is limited to approximately 2000 bytes on single line. Truncation is indicated by ellipses (...) in the output. This limit applies when output is directed to the log (.tlg) or an external file.

The following characters should always be quoted in an argument list to prevent Ora*Tst interpretation:

(cross hatch / number sign) - must be quoted or it comments out any remaining characters.

< (less than sign / left angle bracket) - must be quoted or it is treated as a file redirection.

> (greater than sign / right angle bracket) - must be quoted or it is treated as a file redirection (this includes >>, >>>, >>>>).

^ (circumflex / caret) - must be quoted unless dereferencing a variable.

See the Ora*Tst ECHO command:Known Bugs for details and examples on quoting.

2.18.5 Prerequisites

Within the body of a multi-line echo the control flow should not be transferred to another Ora*Tst script; i.e. the INCLUDE command is not allowed inside an echo body.

Nested invocations of echo will yield unpredictable results. Do not nest them.

2.18.6 See Also

osdfilename Ora*Tst command.

2.18.7 Known Bugs

Ora*Tst bug #143129: There is one extraneous white space after each line echoed or written to a file (simple and multi-line echo.) Fixed in V2.1.1; for compatibility, there is an Ora*Tst command-line option -o or customization variable TST_NOFIXECHO which will disable the fix, if your existing scripts and comparisons relied upon this. In other words, set TST_NOFIXECHO to a TRUE value to enable the old behavior of adding a space to the end of each line. See the section on Invoking Ora*Tst in the Ora*Tst Installation and User's Guide for more information.

Ora*Tst bugs #3975903 and #2621432: The Ora*Tst parsing routine adds spaces around specific characters which can affect the output of the ECHO command (simple and multi-line). The solution is to enclose these characters in single quotes. Ora*Tst adds spaces around the following characters when passed unquoted in an argument to a command:

~ (tilde)

‘ (apostrophe/back quote)

! (exclamation mark/bang)

& (ampersand)

***** (asterisk)

+ (plus sign)

{ (left curly brace)

} (right curly brace)

| (vertical bar/pipe)

; (semicolon)

The solution is to add single quotes around the string containing one of the above characters or around the character itself. In the example below, if the asterisks that make up the comment block are not quoted, spaces will be added creating an invalid comment, which ultimately results in a SQL error.

Example:

```
ECHO select * /* sql comment */ from t;
```

The resulting output is:

```
select * / * sql comment * / from t ;
```

A space is added before and efter each asterisk (*) and the semicolon (;). Additional spaces are not added if the character is already separated by a space. Thus, the asterisk after the select is unchanged.

By adding single quotes in various ways, the problem can be avoided:

```
ECHO 'select * /* sql comment */ from t;'
 -OR-
ECHO select * '/* sql comment */' from 't;'
 -OR-
ECHO select * /'*' sql comment '*'/ from t';'
```

The use of single quotes can get messy if you need to include single quotes in the generated output. In general, use two single quotes to get one in the output. If the string is already quoted and you need to quote a substring, use four single quotes. Here are some examples:

If you want to get this (note space before semicolon):

```
select * from emp where ename = 'KING' ;
```

Then code this:

```
ECHO > tmp.sql
> select * from emp where ename = ''KING'';
ENDECHO
```

If you want to get this (no space before semicolon):

```
select * from emp /* comment */ where ename = 'KING';
```

Then code this:

```
ECHO > tmp.sql
> 'select * from emp /* comment */ where ename = ''''KING'''';'
ENDECHO
```

In this example, the four quotes around KING results in single quotes in the output. The single quotes around the entire string prevents spaces from being added around the asterisks (thus preserving the comment) and semicolon.

2.19 ELSE

2.19.1 Synopsis

```
if {conditional statement}
   {if-body}
else
   {else-body}
endif
```

2.19.2 Description

If the condition specified evaluates to TRUE, the {if-body} is executed. Otherwise, the {else-body} is executed.

2.19.3 Options

None.

2.19.4 Availability

New Ora*Tst built-in command in V3.0.

2.19.5 Prerequisites

Must be used after a valid IF statement

2.19.6 See Also

colon-if, if, elseif and endif Ora*Tst commands.

2.19.7 Known Bugs

None.

2.20 ELSEIF

2.20.1 Synopsis

```
if {1st conditional statement}
   {if-body}
elseif {2nd conditional statement}
   {elseif-body}
else
   {else-body}
endif
```

2.20.2 Description

If the 1st condition specified evaluates to TRUE, the {if-body} is executed. Otherwise, if the 2nd condition specified evaluates to TRUE, the {elseif-body} is executed.

2.20.3 Options

None.

2.20.4 Notes

There is a limit of 63 tokens in the "{conditional statement}" expression, including parenthesis. Thus, the practical limit is about 15 expressions (i.e ELSEIF (var == 1 or var == 2 or ...)). A TST*WRN-23 message with "IF expression too complex" is issued when the limit is exceeded.

2.20.5 Availability

New Ora*Tst built-in command in V3.0.

2.20.6 Prerequisites

Must be used after a valid IF statement.

2.20.7 See Also

colon-if, if, elseif and endif Ora*Tst commands.

2.20.8 Known Bugs

Ora*Tst (#2437818) bug: The else clause was incorrectly executed regardless of previous elseif results in Ora*Tst versions before 3.5.3.0.0.

2.21 ENDECHO

2.21.1 Synopsis

```
echo > [<area>:]<name>.<type>
   ...
> { [<string>] | [%t] } ...
   ...
endecho
```

2.21.2 Description

Ends a multi-line ECHO block.

2.21.3 Options

None.

2.21.4 Availability

Ora*Tst built-in command since V1.0.6.

2.21.5 Prerequisites

Must be used only to end a multi-line echo construct.

2.21.6 See Also

echo Ora*Tst command.

2.21.7 Known Bugs

None.

2.22 ENDIF

2.22.1 Synopsis

```
 (1)
if {conditional statement}
   ...
endif
 (2)
<global var>:
   ...
endif
 (3)
if {conditional statement}
   ...
else
   ...
endif
 (4)
if {conditional statement}
...
elseif {conditional statement}
   ...
endif
```

2.22.2 Description

Ends a colon-if or if construct.

2.22.3 Options

None.

2.22.4 Availability

Ora*Tst built-in command since V1.0.6.

2.22.5 Prerequisites

Must be used only to end a colon-if or if construct.

2.22.6 See Also

colon-if, if, else and elseif Ora*Tst commands.

2.22.7 Known Bugs

None.

2.23 ENDLOOP

2.23.1 Synopsis

```
 (1)
loop
   ...
   { [break | continue] }
   ...
endloop
 (2)
for <var> <starting value> <ending value> [<step factor>]
   ...
   { [break | continue] }
   ...
endloop
```

2.23.2 Description

Used to end a LOOP construct or a FOR construct.

2.23.3 Options

None.

2.23.4 Availability

Ora*Tst built-in command since V1.0.6.

2.23.5 Prerequisites

Must be used to end a LOOP construct or a for construct.

2.23.6 See Also

loop, for, break, continue and block_end Ora*Tst commands.

2.23.7 Known Bugs

None.

2.24 EXECUTE

2.24.1 Synopsis

```
execute [ASIS] [SETRESULT] [<exe_area>:]<exe_name>[.EXE|.COM]
        { [ <exe cmdline arg> ] } ...
        [< <inarea>:<inname>.<intype>]
        [>[>>] <outarea>:<outname>.<outtype>]
        [>>[>>] <errarea>:<errname>.<errtype>]
```

2.24.2 Description

Invokes an external executable with the specified command line arguments. The executable may either be a compiled and, if applicable, linked program (portable filetype EXE) or an executable OSD script (portable filetype COM). If the ASIS option is used, the command line arguments and filenames (including redirection) are left as-is, i.e. are not lowercased. (Note that this is not portable; it is meant as a work-around only. See Options below.) For more information on command line arguments and how they are passed to the executable, refer to the Command Line Arguments section of the Ora*Tst Installation and User's Guide.

This command is the same as RUNUTL except:

- <exe_area> defaults to T_WORK
- A TST*WRN-40 message is issued for non-zero return codes

If the <exe area> is not explicitly specified, it defaults to T_WORK.

The current working directory for the external command is T_WORK.

<inarea>, <outarea> and <errarea> default to T_WORK if not specified.

By default, standard output will be redirected to the file T_WORK:<exe_name>.log. Redirected output can overwrite (>) or append (>>>) (note the difference between common UNIX redirection) to the specified file.

By default, standard error is likely to print to the terminal (there is no default redirection to a file for standard error). Redirected error can overwrite (>>) or append (>>>>) (note the difference between common UNIX redirection) to the specified file. The default behavior for stderr output can be changed by specifying the -e parameter on Ora*Tst which causes stderr to be written to <testname>.ter. See section 2.7 of the Ora*Tst Installation and User's Guide.

Note that I/O redirection (as well as the concept of standard output and input) are not always portable and may not be available on all ports of Ora*Tst. I/O redirection during the execution of an OSD script is supported in the sense that, if an executable is invoked from the script, I/O will be redirected for that executable in the way described above.

The purpose of the execute Ora*Tst command is the invocation of an executable which either has been previously compiled via the compile Ora*Tst command or pre-exists in the OS environment. Since it is meant to run an executable, it will only accept the COM and EXE filetypes.

The return code of the executable is checked and a "TST*WRN-40: Extern executable error:" message is reported in the TLG file for non-zero return codes. If a non-zero return code is expected or not significant, consider using the RUNUTL command. This aovids the need to suppress the TST*WRN-40 message.

The OSD return code is stored in the variable TST_EXE_STATUS. This variable is changed by each invocation of EXECUTE and RUNUTL. Do not compare the return code with absolute numbers since different operating systems have different meanings for success and failure.

On UNIX, the OSD return code of 0 is returned, if and only if the executable exited successfully.

On VMS, the OSD return code of 1 is returned, if and only if the executable exited successfully. As of Ora*Tst Version 3.5.4.0.0, the value of TST_EXE_STATUS is set to 0 on success. See bug 3843511 for details.

If you need to pass a filename to the external command, use the OSDFILENAME command (V2.1) to convert the file specification to an OSD filename first, otherwise your script may not be portable, and you may not be able to specify areas etc. as the external OSD environment does not understand Ora*Tst portable file specifications.

Starting from V3.1.0, users can specify arguments using multi-byte characters.

Starting from V3.5.4.0.0, changes have been implemented to allow the EXECUTE command to return a string value that is stored in a predefined Oratst internal variable, TST_EXE_RESULT. This variable is changed by each invocation of EXECUTE or RUNUTL. The resulting string is obtained by reading the standard output file of the command that is run by EXECUTE. This functionality is enabled by specifying the SETRESULT flag (available 3.5.4.1.0) or by setting the internal Oratst variable TST_GET_EXE_RESULT to a true value (the default value is false).

2.24.3 Options

ASIS

If specified, Ora*Tst will leave all command line options as-is, i.e. it will not lowercase them. This includes filenames (redirection too.) To retain the case of the filenames, but lowercase the arguments as normal, use the -i or -u Ora*Tst command line options. See Invoking Ora*Tst in the Ora*Tst Installation and User's Guide.

Note that using this option, since it is case-sensitive, is not portable and will likely break on other platforms. It is meant as a workaround solution feature only. Please do not use this feature unless absolutely necessary in base development test scripts, and as with any non-portable feature, put it in an OSD macro. You should be putting external executable invocations in an OSD macro in any case (see below.)

SETRESULT

If specified, Ora*Tst stores the output of the external command into the Ora*Tst internal variable TST_EXE_RESULT. This provides functionality similar to the UNIX shell command substitution with back quotes (i.e. result=`cmd`).

Starting with Version 3.5.4.0.0, changes were implemented to allow the EXECUTE command to return a string value that is stored in a predefined Oratst internal global variable, TST_EXE_RESULT. This variable is changed by each invocation of EXECUTE or RUNUTL Thus, the value should be saved into a separate Ora*Tst variable immediately after the command. The resulting string is obtained by reading the Ora*Tst standard output file of the EXECUTE command. This functionality was originally enabled by setting the internal Oratst variable TST_GET_EXE_RESULT to a true value (the default value is false). The same action can now be specified with the SETRESULT flag. The SETRESULT flag is the preferred method for returning results into TST_EXE_RESULT.

The Ora*Tst standard output file is the default file T_WORK:<exename>.log or the the output file specified by <outname>.<outtype>.

Additional TST_EXE_RESULT details:

When SETRESULT is specified, the standard output of the EXECUTE command is processed and the TST_EXE_RESULT variable set accordingly. Each execution of this command starts by setting TST_EXE_RESULT to null. In most cases, TST_EXE_RESULT is set to null when errors are encountered. However, there is the possibility that some error conditions may leave garbage in the TST_EXE_RESULT variable. If this possibility presents a problem for your script consider adding logic to validate the TST_EXE_RESULT value.

When SETRESULT is specified (or TST_GET_EXE_RESULT is set to a true value), additional processing is required to read the Ora*Tst standard output file and set the TST_EXE_RESULT variable. It is suggested to use this flag only for those invocations of EXECUTE that require string result processing.

A maximum of 2048 bytes are read from the Ora*Tst standard output file and assigned to the TST_EXE_RESULT variable. Before the assignment to TST_EXE_RESULT:

* Newline characters (CR/LF) are converted to spaces.
* All non-printable characters are converted to spaces.
* Leading and trailing spaces are removed.
* Multiple spaces are trimmed to a single space.

By default, the EXECUTE command creates a standard output file containing the stdout of the external command (T_WORK:<exename>.log). Thus, it is not necessary to explicitly specify a standard output file for TST_EXE_RESULT to be set. The standard output is processed after the external command specified by EXECUTE completes. Thus, the TST_EXE_RESULT variable is assigned what exists in the standard output file after any appends or redirection from stderr.

2.24.4 Notes

The following characters should always be quoted in an argument list to prevent Ora*Tst interpretation:

(cross hatch/number sign) - must be quoted or it comments out any remaining characters.

< (less than sign/left angle bracket) - must be quoted or it is treated as a file redirection.

> (greater than sign/right angle bracket) - must be quoted or it is treated as a file redirection (this includes >>, >>>, >>>>).

^ (circumflex/caret) - must be quoted unless dereferencing a variable.

See the Ora*Tst ECHO command:Known Bugs for details and examples on quoting.

On Windows a command argument ending with a backslash (\) needs to be quoted or an additional backslash needs to be added. This is a common requirement when invoking Java and -Dsep is specified with a backslash. Failure to do so can result in a truncated command line. For example, specify -Dsep=\ \ instead of -Dsep=\

On Windows each command line argument is enclosed in double quotes before it is passed to the operating system for execution. It is possible to turn off this behavior by setting the TST_NOQUOTE_WIN_CMD variable to a TRUE value. The default is FALSE. Only use this if absolutely necessary and to turn the switch off after execution. This switch is available starting at V3.5.5.3.0. See bug 7686482 for more details.

Keep in mind that when executing scripts such as Perl or Java (i.e execute T_SYSTEM:perl mycmd.pl) the default output file will match the <exe_name> (i.e. perl or java) and not the script name (which is an argument). Thus, it common to end up with an output file of perl.log, which is then overwritten on subsequent Perl commands using the same format.

2.24.5 Availability

The command is an Ora*Tst built-in command in V1.0.6. The ability to redirect standard error was added in V2.0; the ability to execute OSD scripts, if applicable, was added in V2.0.

ASIS option added in V2.1.

The return of a string value into TST_EXE_RESULT was added in V3.5.4.0.0.

SETRESULT option added v3.5.4.1.0.

2.24.6 Prerequisites

It is strongly recommended that the invocation of any executable be encapsulated in an OSD Ora*Tst macro, as external executable names, options etc. may vary according to platform.

The specified executable must exist.

2.24.7 See Also

compile, runutl, and osdfilename Ora*Tst commands.

2.24.8 Example usages

EXECUTE T_WORK:myprog abc 123 -recur < T_SOURCE:input

EXECUTE ASIS PATH:ls -alF > T_WORK:listing.dat

EXECUTE ASIS PATH:grep '^xyz' ^osdfname^

2.24.9 Known Bugs

Ora*Tst bugs #3975903 and #2621432: The Ora*Tst parsing routine adds spaces around specific characters which can affect the output of the EXECUTE command. The solution is to enclose these characters in single quotes. See the Ora*Tst ECHO command:Known Bugs for additional details and the list of special characters.

See bug 7686482 regarding double quotes added to arguments for Windows.

2.25 EXIT

2.25.1 Synopsis

```
exit [FATAL] <code>
```

2.25.2 Description

Causes the current test script to abort and return to parent script, if any, copying the exit code to the Ora*Tst TST_STATUS system variable.

2.25.3 Options

FATAL

If specified, EXIT will abort the current script and all parent test scripts, producing a domino exiting effect, and exit Ora*Tst itself. The exit is clean, i.e. the normal end of script logging will proceed; Ora*Tst treats the exit command as if it reached a normal end-of-file condition. Useful for severe cases where checking for an exit code throughout all nested levels is impossible or too laborious.

Note: EXIT FATAL called from a FORKed script does *not* abort the parent test script.

2.25.4 Notes

Use EXIT to terminate the execution flow of a test script during an exception. Avoid using it at the end of a nested test or macro to mark the return of normal execution flow. This command should only be used for serious errors where test recovery / continuation is not expected. Consider using the RETURN command for non-fatal returns. The EXIT code is stored in the global variable TST_STATUS which is updated on every non-control Ora*Tst command. Thus, you need to save TST_STATUS in a separate variable if you want to reference the original exit value in subsequent commands.

Specifying an exit code of 1 causes Ora*Tst to return a failure code (also 1 on most platforms) to the operating system upon termination. Specifying any value other than 1 causes Ora*Tst to return a success code (a 0 on most platforms) to the operating system upon termination.

When EXIT is called from a FORKed script the exit code is *not* available to the parent or forking script.

When EXIT is called from an INCLUDEd script only the included script is terminated, not the parent calling INCLUDE. The same applies to macros.

2.25.5 Availability

Ora*Tst built-in command since V2.1.

2.25.6 Prerequisites

The <code> must be an integer ranging between -255 and 255.

2.25.7 See Also

break, run / runtest, return Ora*Tst commands.

2.25.8 Known Bugs

Generic bug (#3305161): The EXIT command behaves as if FATAL is specified when the exit code is negative. This is fixed AFTER Version 3.5.3.2.0.

Bug 15948899 - TST_STATUS VARIABLE IS NOT CORRECT AFTER CALLING EXIT WITHIN A MACRO. This is fixed at Version 3.6.0.3.0.

2.26 EXPORT

2.26.1 Synopsis

```
export <var>
```

2.26.2 Description

The Ora*Tst variable <var> is exported to the OS environment with its current value. The variable can subsequently be imported with its respective value from the OS environment via the import Ora*Tst command. Please note that there may be a system specific limit on the length of the value stored in an environment variables. The Solaris limit is 1024.

The explicit export of certain variables is strongly discouraged, since it might interfere with the internal implementation of Ora*Tst built- in commands (e.g., ORA_SID and instance command).

Note that although the Ora*Tst variable <var> can be either local or global, the IMPORT command will only set a global Ora*Tst variable. If you export a local, change it in the outside environment, and re-import it, you will not see the change since the original local variable will hide the new global variable, according to normal scoping rules.

2.26.3 Options

None.

2.26.4 Notes

The <var> name is case insensitive. Ora*Tst automatically changes the <var> string to uppercase during processing. Thus, regardless of how <var> is specified, the variable name exported to the environment is uppercase. Lower or mixed case environment variable names may not be portable and are not supported.

It is not necessary to export a variable that is not to be used outside of Ora*Tst. Due to potential port specific issues, it is not recommended that you export variables unless absolutely necessary. An export is only required if a command external to Ora*Tst (perhaps initiated by execute or runutl) requires the environment variable. Instead, define the environment variable before calling Ora*Tst.

It is not currently possible to unset or undefine an environment variable using Ora*Tst.

2.26.5 Availability

The command is an Ora*Tst built-in command in V1.0.6.

2.26.6 Potential changes in future versions

The implementation of the import/export mechanism is currently being reviewed and subject to change. Please do not rely on port-specific implementation details.

2.26.7 Prerequisites

The Ora*Tst variable <var> must be defined either globally or locally.

2.26.8 See Also

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

2.26.9 Known Bugs

None.

2.27 FILTER

2.27.1 Synopsis

```
filter [REVERSE] [<filter_area>:]<filterfile>.<filtertype>
         [<orig_area>:]<origfile>.<origtype>
         [<new_area>:]<newfile.<newtype>
```

2.27.2 Description

This command filters out (removes) lines from <origfile> based on regular expressions defined in the <filterfile> and writes the result to file <newfile>. The resulting <newfile> contains the lines from <origfile> that **do not match** the expression(s) in <filterfile>.

Null lines in the <origfile> are always removed regardless of the expression(s) in <filterfile>. The default area for all three files is T_WORK.

The filter file must contain Ora*Tst regular expression(s), one per line and have non-blank delimiters. The expression must match the entire line. No trailing blanks are allowed. Blank lines in the <filter> file, if any, are skipped.

For example:

/^ora-1234.*/

@^ora-789.*@

The reason for the delimiters is to eliminate confusion as to whether blank spaces should be matched or not. If the delimiter is in the expression, then it needs to be escaped with a backslash.

2.27.3 Options

REVERSE

If specified, the filter logic is reversed. Lines in <origfile> matching the expression(s) in the <filterfile> are written to the <newfile>.

This option is available starting with Ora*Tst Version 3.5.5.2.0.

2.27.4 Notes

Be sure the <filterfile> contains no trailing blanks. These can be introduced inadvertently if the filter file is created using the multi-line ECHO command and the TST_NOFIXECHO variable is true (default is false). Symptoms of trailing blanks include:

TST_WRN-55: read failed: FILTER: <filterfile>.flt: mismatch delimiter: ignore ...

2.27.5 Availability

Ora*Tst built-in command since V3.2.2.

REVERSE option available since V3.5.5.2.0.

2.27.6 Prerequisites

The maximum line length of the <filterfile> and <origfile> is 57342 bytes (56K - 2) as of V3.5.5.3.0. Prior to this version, the line length limit was 2047 bytes.

A maximum of 1023 lines can be specified in the <filterfile>.

2.27.7 See Also

For more detailed information on regular expressions, refer to the Regular Expressions section of the Ora*Tst Installation and User's Guide.

2.27.8 Known Bugs

Bug 8476650: The wrong file may be reported on TST_WRN-55. Fixed at V3.5.5.3.0.

2.28 FINDDIR

2.28.1 Synopsis

```
 (1)
finddir [<area>:] <local var>
 (2)
finddir [<area>:]<name>[.<type>] <local var>
```

2.28.2 Description

Determines whether the specified <area> or the specified <name> exists and is a directory. If it is, a local variable <local var> is set to a value evaluating to TRUE, otherwise it is set to a value evaluating to FALSE. If <area> is not specified, it defaults to the working area T_WORK.

If the directory is found, the full OSD pathname for the found directory is printed in the TLG file to help determine in which path element of the area the directory was found.

2.28.3 Options

None.

2.28.4 Notes

The specification of any portion of a platform specific path component (i.e. dir1/dir2) for <name> is not portable and is not supported.

2.28.5 Availability

Ora*Tst built-in command since V3.1.0.

2.28.6 Prerequisites

None.

2.28.7 See Also

findfile, osddirname, cleanout, mkdir, rmdir Ora*Tst commands.

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

2.28.8 Known Bugs

None.

2.29 FINDFILE

2.29.1 Synopsis

```
findfile [<area>:]<name>.<type> <local var>
```

2.29.2 Description

Determines whether the specified file <name> of type <type> exists in the area <area>. If it does, a local variable <local var> is set to a value evaluating to TRUE, otherwise to a value evaluating to FALSE. If <area> is not specified, it defaults to the working area T_WORK.

In V3.0, If the file is found, the full OSD pathname for the found file is printed in the TLG file (to help determine in which path element of the area the file was found.)

2.29.3 Options

None.

2.29.4 Notes

This command is often used to look for a DIF or SUC file after an LDIFF compare to determine if the files compared as expected. However, a quicker (better performing) and more accurate method is to check the TST_LDIFF Ora*Tst variable (V2.1.1) instead, as that involves no filesystem access. See the ldiff command for additional details on TST_LDIFF.

The specification of any portion of a platform specific path component (i.e. dirname / filename) for <name> is not portable and is not supported.

A file name longer than 128 bytes is truncated without warning.

2.29.5 Availability

Ora*Tst built-in command since V1.0.6. OSD pathname echo to tlg in V3.0.

2.29.6 Prerequisites

None.

2.29.7 See Also

finddir, osdfilename, ldiff Ora*Tst command.

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

2.29.8 Known Bugs

None.

2.30 FINDRSTR

2.30.1 Synopsis

```
findrstr <srcstr_var> <substr_var> <index_var> [log|nolog]
```

2.30.2 Description

Finds in variable <srcstr_var> the first occurrence of the sequence of characters in variable <substr_var> starting from the **end** of <srcstr_var> and returns in local variable <index_var> the position in <srcstr_var> of the first match. If <substr_var> is not found in <srcstr_var>, -1 is returned in <index_var>.The first character of <srcstr_var> is considered the 0th position. This is similar to the FINDSTR command except that the search starts at the end of <srcstr_var>.

If <srcstr_var> is null then -1 is returned. If <substr_var> is null then 0 is returned. If both <srcstr_var> and <substr_var> are null then 0 is returned. The default is log.

2.30.3 Options

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log.

2.30.4 Availability

Ora*Tst built-in command since Version 3.5.5.3.0.

2.30.5 Prerequisites

None.

2.30.6 See Also

findstr, strcat, substr, varlen, subword, word, words Ora*Tst commands.

2.30.7 Known Bugs

None.

2.31 FINDSTR

2.31.1 Synopsis

```
findstr <srcstr_var> <substr_var> <index_var> [log|nolog]
```

2.31.2 Description

Finds in variable <srcstr_var> the first occurrence of the sequence of characters in variable <substr_var> and returns in local variable <index_var> the position in <srcstr_var> of the first match. If <substr_var> is not found in <srcstr_var>, -1 is returned in <index_var>.The first character of <srcstr_var> is considered the 0th position.

If <srcstr_var> is null then -1 is returned. If <substr_var> is null then 0 is returned. If both <srcstr_var> and <substr_var> are null then 0 is returned.

2.31.3 Options

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log. New option starting at V3.5.5.3.0.

2.31.4 Availability

Ora*Tst built-in command since Version 3.2.5.

The log / nolog parameter is available at Version 3.5.5.3.0.

2.31.5 Prerequisites

None.

2.31.6 See Also

findrstr, strcat, substr, subword, varlen, word, words Ora*Tst commands.

2.31.7 Known Bugs

None.

2.32 FOR

2.32.1 Synopsis

```
for <var> <starting value> <ending value> [<step factor>]
  ...
  { [ break | continue] }
  ...
endloop
```

2.32.2 Description

The {for body} is executed repeatedly until the break Ora*Tst command is encountered or when the value of “var” is greater than the ending value. The flow of control will then continue with the first statement following the matching endloop Ora*Tst command. If a CONTINUE Ora*Tst command is encountered, the flow of control continues with the first statement in the {for body} (next iteration).

At the beginning of a for..endloop construct, variable “var” will be set to the starting value. If variable “var” does not exist, it will be created. After each iteration, the value of “var” will be incremented by the “step factor”. Note that step factor can be either positive or negative. If no step factor is provided, the default value is+1 if <starting value> is less than or equal to <ending value> and -1 if <starting value> is greater than <ending value>.

2.32.3 Options

None.

2.32.4 Availability

New Ora*Tst built-in command in V3.0.

2.32.5 Prerequisites

In Ora*Tst V3.0, the {for body} will be stored in memory to minimize I/O. Huge for...endloop can be a potential problem on desktop platforms.

The maximum loop nesting level is 10.

The <starting value>, <ending value> and <step factor> values must be in the range of a signed 32-bit integer.

2.32.6 See Also

loop, endloop, break, continue, block_begin Ora*Tst commands.

2.32.7 Known Bugs

None.

2.33 FORK

2.33.1 Synopsis

```
fork [<area>:]<scr_name>[.TSC] [WAIT|NOWAIT|KILL] { [ <local_var1>
        ] | [ [<local_var2>[=<value2]] ] } ...
```

2.33.2 Description

Creates a new Ora*Tst process (child) to run the specified test script <scr_name>, possibly simultaneously with the current (parent) test script and / or additional child processes.

If the area <area> is not explicitly specified, it defaults to T_SOURCE.

The filetype of the script <scr_name> is forced to be TSC. The command line arguments are interpreted as keywords, such as <local_var1>, or keyword / value pairs, such as <local_var2>=<value2>.

The command line arguments are handled exactly as command line arguments for RUN / RUNTEST scripts.

Explicit redirection on the command line with `>' or `<' is not allowed.

In the parent test script the global Ora*Tst variable ENTRY is set to a process handle, e.g. an OSD process id. This process handle has an OSD value which uniquely identifies the newly created process in the context of the current test script execution. The sole purpose of this process handle is its usage in conjunction with the Ora*Tst STOP command. Note that subsequent invocations of the FORK command overwrites the global ENTRY variable, so if you want to use the value later to stop the child, store it in your own variable.

2.33.3 Options

`WAIT (default)`

If specified or allowed to default, the new child process is created in a synchronized fashion with the parent test script. The child process waits to start executing the specified test script until the parent script executes the next Ora*Tst WAIT command. When the WAIT command is encountered in the parent script the child process begins execution The parent script then waits until the forked script is finished. Multiple FORK commands can be issued in order to fork several scripts simultaneously.

If multiple scripts have been FORKed with the WAIT option, all children will start at the parent's WAIT command and the parent will wait for all children to finish before resuming the next command.

Note: If no later Ora*Tst WAIT command is encountered the newly created child process(es) will never start.

`NOWAIT`

If specified, the new child process is created and starts executing the specified test script immediately. The parent script then continues execution. Subsequent Ora*Tst WAIT commands in the parent script have no effect on children forked with the NOWAIT option, nor will the parent wait for such child processes to finish. Since the child runs asynchronously, in certain cases the child process may not complete execution until after the parent process has finished.

KILL

If specified, the new child process is created and starts executing the specified test script immediately (similar to the NOWAIT option). When the parent script encounters the next Ora*Tst WAIT command, the newly created child process is killed.

The kill is performed after waiting on any children previously FORKed with the WAIT option. That is, if there are any child processes FORKed with WAIT in addition to FORKed with KILL, the parent waits until the "wait" child processes finish before performing the kill. If no children have previously been FORKed with WAIT, then the kill is performed immediately upon execution of next the Ora*Tst WAIT command.

If the parent script does not encounter a WAIT Ora*Tst command, the newly created child process will continue to run until its script completes execution (which may be after the parent process has finished.)

The specification of the kill signal is configurable for UNIX platforms. Bug 3108294 allows for a UNIX SIGTERM signal to be used on the kill (the default is SIGKILL). The Ora*Tst variable TST_FORK_KILLTYPE can be set to KILL or TERM (not case sensitive, default is KILL) to specify the UNIX signal to use. This setting applies to all processes forked with the KILL option. See the Bug text for additional details.

The kill command may not be effective at stopping child processes of the forked process on all platforms. This is known to be an issue on Windows.

2.33.4 Notes

The implementation of the fork / wait mechanism is very system specific.

The internal limit for the total number of forked processes is platform specific. The coded limit is 128 for most UNIX platforms and 64 for Windows platforms. However, the practical limit may be less depending on the capacity of the machine. This total applies individually to each class of FORK with WAIT and FORK with KILL. FORK with NOWAIT currently has no coded maximum and is limited only by the capacity of the machine.

Nested FORK commands (a FORKed script that performs a FORK) are known to cause problems in versions prior to V3.5.5.4.0. The nesting of FORKs is not recommended and should be avoided, but is permitted because of existing usage. Starting at 3.5.5.4.0 a warning and error is issued when nesting is greater than 2. The warning / error limit is configurable. See the TST_FORK_NEST_WRN and TST_FORK_NEST_ERR variable descriptions in the Ora*Tst Installation and User's Guide for more information. Consider setting TST_FORK_NEST_WRN to a value of 1 (default is 2) to avoid inadvertently creating a test with nested FORKs.

2.33.5 Availability

Ora*Tst built-in command since V1.0.6. Startup and cleanup features added in V3.0. The configurable UNIX kill signal was available in V3.5.3.2.0.

2.33.6 Prerequisites

The specified test script <scr name> must be a TSC file which exists in the area specified.

The Ora*Tst FORK command must **not** be executed by an already forked test script (no nested forks). Unpredictable results may occur and no warning is issued.

2.33.7 See Also.

isactive, wait, stop, sleep Ora*Tst commands.

2.33.8 Known Bugs

Bug 8301705: Fork may not wait for WAIT command on Windows. Fixed at V3.5.5.3.0.

Bug 8841149: Looping forever in parent script that contains a FORK. Fixed at V3.5.5.3.0.

Bug 8669368: Macro file name is incorrect - TST*WRN-4 File Not Found. Fixed at V3.5.5.4.0.

2.34 GET (macro)

2.34.1 Synopsis

```
get <srcfile>[.<srctype>] . . .
```

2.34.2 Description

Copies a file <srcfile> of type <srctype> from the source area T_SOURCE to a file by the same name in the working area T_WORK.

The default macro accepts multiple arguments, i.e. multiple file specifications will all be copied.

The defaults are the same as for the cpfile Ora*Tst command. The protection mode of the target file and the source file are the same, with the exception that the target file will in any case be writable by the owner.

The default macro is essentially a multi-argument cpfile command, except that if a file by the same name already exists in the target area, the file is not copied. To overwrite an existing file in the target area, use the cpfile Ora*Tst command.

2.34.3 Options

None.

2.34.4 Notes

A default version of this macro is provided by Ora*Tst. It is common to find this macro overridden in the RDBMS Server environment. Thus, the options and behavior of the overriding macro may differ from that described here.

2.34.5 Availability

Ora*Tst macro since V1.0.6. There is an automatic default macro definition.

2.34.6 Prerequisites

None.

2.34.7 See Also

For more detailed information on macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User's Guide.

For more information on the implementation of the portable Ora*Tst default macro, refer to the file ctm01.tsc.

cpfile Ora*Tst command.

2.34.8 Known Bugs

None.

2.35 GETAREA

2.35.1 Synopsis

```
getarea [<area>:][<name>][.<type>] <local_var>
```

2.35.2 Description

String manipulation: Extracts the area name from the Ora*Tst portable file specification and places the result (minus the `:' separator) in the local variable <local_var>.

All parts of the file specification (i.e., area, name, type) are optional; however, obviously, at least ONE part should be present to evaluate. If an area is not specified, the local variable <local var> is set to the empty string ““.

2.35.3 Options

None.

2.35.4 Availability

New Ora*Tst built-in command in V3.0.

2.35.5 Prerequisites

After the evaluation of Ora*Tst variables, if any, the first argument must not be the empty string; it should be a well-formed Ora*Tst file specification, but Ora*Tst does not enforce the validity of the first argument.

2.35.6 See Also

For more detailed information on Ora*Tst file specifications, refer to the File Specifications and Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

getbasename Ora*Tst command.

2.35.7 Known Bugs

None.

2.36 GETBASENAME

2.36.1 Synopsis

```
getbasename [<area>:][<name>][.<type>] <local_var>
```

2.36.2 Description

String manipulation: Extracts the basename (<name>) from the Ora*Tst portable file specification and places the result in the local variable <local_var>.

All parts of the file specification (i.e., area, name, type) are optional; however, obviously, at least ONE part should be present to evaluate. If a <name> is not specified, the local variable <local var> is set to the empty string ““.

2.36.3 Options

None.

2.36.4 Availability

Ora*Tst built-in command since V2.0.

2.36.5 Prerequisites

After the evaluation of Ora*Tst variables, if any, the first argument must not be the empty string; it should be a well-formed Ora*Tst file specification, but Ora*Tst does not enforce the validity of the first argument.

2.36.6 See Also

getarea, getfiletype Ora*Tst command.

2.36.7 Known Bugs

None.

2.37 GETFILE

2.37.1 Synopsis

```
getfile <h_remotevar> [<remote_srcarea>:]<srcfile>.<srctype>
        [<local_destarea>:]<destfile>[.<desttype>]
        [legacy|text|binary]
```

2.37.2 Description

Copies a file <srcfile> of type <srctype> in area <remote_srcarea> from the remote host <h_remotevar> to file <destfile> of type <desttype> in area <local_destarea> on the local host. The default area for both <remote_srcarea> and <local_destarea> is T_WORK. The <h_remotevar> must already be defined and specify a remote machine and port number that is running Ora*Tst in server mode.

2.37.3 Options

LEGACY (default)

If specified (or allowed to default), legacy mode supports the copy of text files. No data translation of any kind is performed between hosts. This implies that the source and target platforms should be the same OS type. Null bytes (binary zeros) are NOT allowed in the data. This is the mode of operation available in Ora*Tst versions prior to 3.6.0.0.0.

TEXT

Text mode transfers text files, correcting the end of line (EOL) character(s) depending on the platform. It is expected that UNIX, Linux and MAC OS X based platforms use an EOL character of LF (line feed, hex 0A). It is expected that Windows platforms use EOL characters CRLF (carriage return, hex 0D, line feed, hex 0A). When transferring text files between hosts of different types (UNIX-like and Windows) the EOL is updated as appropriate for the platform.

Text mode is only supported on ASCII platforms. Transferring files between ASCII/EBCDIC platforms may cause unpredicatable results and is not currently supported.

No other translation of any kind is supported in text mode. No character set translation is made or supported. Null bytes (binary zeros) are NOT allowed in the data.

BINARY

Binary mode transfers files without data translation of any kind. The data can contain Null bytes (binary zeros).

2.37.4 Notes

The legacy, text & binary options are new for Ora*Tst version 3.6.0.0.0. Prior versions only support the equivalent of legacy mode.

Compatibility Note: As of Oratst Version 3.5.3.3.0, this command is not downward compatible with previous Oratst versions. Both the client (host running getfile) and server (oratst -server) versions must be at Version 3.5.3.3.0 and above -OR- Version 3.5.3.2.0 and below to function properly. Mixing the client and server versions between 3.5.3.3.0 (and newer) and 3.5.3.2.0 (and older) will cause unpredicable results.When using a 3.6.0.0.0 or later version client against a pre 3.6.0.0.0 server, only the legacy mode flag is supported.

2.37.5 Availability

Ora*Tst built-in command since V3.3.0

The legacy, text & binary modes are available starting with V3.6.0.0.0.

2.37.6 Prerequisites

The remote host must have Ora*Tst running in server mode (oratst -server <portnumber>). The <h_remotevar> must be a previously defined Ora*Tst variable specifying the remote host in the format <hostname:portnumber>. The remote host area <srcarea> must exist and be defined to the remotely running Ora*Tst server.

2.37.7 See Also

The Ora*Tst Installation and User's guide for details on the Ora*Tst -server command.

putfile, remotefindfile, runremote Ora*Tst commands.

2.37.8 Known Bugs

Ora*Tst bug 6070919 - Intermittent failures with Getfile/Putfile. This bug affects version 3.5.3.3.0 thru 3.5.4.1.0 and is fixed starting at 3.5.4.1.2.

2.38 GETFILETYPE

2.38.1 Synopsis

```
getfiletype [<area>:][<name>][.<type>] <local_var>
```

2.38.2 Description

String manipulation: Extracts the file type <type> from the Ora*Tst portable file specification and places the result in the local variable <local_var>.

extract substring; i.e. interprets the first argument (string) as an Ora*Tst file specification, extracts the portable filetype <type> from that specification and deposits that filetype in the local variable <local var>.

All parts of the file specification (i.e., area, name, type) are optional; however, obviously, at least ONE part should be present to evaluate. If a filetype is not specified, the local variable <local var> is set to the empty string ““.

2.38.3 Options

None.

2.38.4 Availability

Ora*Tst built-in command since V1.0.6.

2.38.5 Prerequisites

After the evaluation of Ora*Tst variables, if any, the first argument must not be the empty string; it should be a well-formed Ora*Tst file specification, but Ora*Tst does not enforce the validity of the first argument.

2.38.6 See Also

getarea, getbasename Ora*Tst command.

2.38.7 Known Bugs

None.

2.39 GETGLOBAL

2.39.1 Synopsis

```
getglobal <globvar> <newvar> [nowarn]
```

2.39.2 Description

Bypasses normal scoping rules and retrieves the value of the global Ora*Tst variable <globvar>, if it exists, and sets local variable <newvar> to the global's value. This command can be used to work around a hiding local variable.

If no such global variable exists, a warning is issued and <newvar> is not set.

2.39.3 Options

`nowarn`

This optional parameter indicates to not issue "TST*WRN-13: referenced variable not found" when the <global_var> does not exist. Even though the warning is not issued, the TST_STATUS variable will be be set to a non-0 value. It is better to use the nowarn option on this command instead of globally suppressing the "TST*WRN-13" message.

2.39.4 Availability

New Ora*Tst built-in command in V3.0.

Nowarn option available starting at V3.5.5.4.0.

2.39.5 Prerequisites

Global of that name exists.

2.39.6 See Also

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

let, unlet, set, unset, and varlen Ora*Tst commands.

2.39.7 Known Bugs

None.

2.40 GETLOCAL

2.40.1 Synopsis

```
getlocal <localvar> <newvar> [nowarn]
```

2.40.2 Description

Retrieves the value of the local Ora*Tst variable <localvar>, if it exists, and sets local variable <newvar> to the local's value and sets internal variable TST_STATUS to 0. If the <localvar> does not exist as a local variable then <newvar> is NOT updated and TST_STATUS is set to a non-0 value.

When NOWARN is specified, if the <localvar> does not exist, then no "TST*WRN-13" message is issued. TST_STATUS is still set to a non-0 value.

This command is primarily available for debugging purposes. It allows you to confirm that a local variable of a specific name exists and may be "hiding" a global variable of the same name.

2.40.3 Options

nowarn

This optional parameter indicates to not issue "TST*WRN-13: referenced variable not found" when the <local_var> does not exist. Even though the warning is not issued, the TST_STATUS variable will be be set to a non-0 value. It is better to use the nowarn option on this command instead of globally suppressing the "TST*WRN-13" message.

2.40.4 Availability

New Ora*Tst built-in command atV3.6.0.3.0.

2.40.5 Prerequisites

None.

2.40.6 See Also

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

getglobal, let, unlet, set and unset Ora*Tst commands.

2.40.7 Known Bugs

None.

2.41 IF

2.41.1 Synopsis

```
 (1)
if {conditional statement}
   ...
endif
 (2)
if {conditional statement}
   {if-body}
else
   {else-body}
endif
 (3)
if {conditional statement}
   {if-body}
elseif {conditional statement}
   {elseif-body}
endif
```

2.41.2 Options

None.

2.41.3 Description

Simple IF statement (1): If the condition specified evaluates to TRUE, the {if-body} is executed, otherwise it is not.

IF...ELSE...ENDIF (2): If the condition specified evaluates to TRUE, the {if-body} is executed. Otherwise, the {else-body} is executed.

IF...ELSEIF...ENDIF (3): If the 1st condition evaluates to TRUE, the {if-body} is executed. Otherwise, if the 2nd condition evaluates to TRUE, the {elseif-body} is executed; and so on.

Note that IF...ELSEIF...ENDIF can be nested. One ENDIF should match a single IF command (ELSEIF does not need to be “ended”.)

The conditional statement, and its evaluation, can take one of the forms listed below. Note that <var> will be dereferenced implicitly, and any comparisons/evaluations listed below are to the resulting, dereferenced value. <var> may itself be a string expression, such as `arg^i^’; `i’ will first be dereferenced, as normal for any command line argument, and then IF will implicitly dereference the resulting string as if it were a variable name.

<value> is the string expression (which may also of course contain explicitly dereferenced variables etc.) which will be compared to the contents of the implicitly dereferenced variable <var>.

If the comparison is a numeric one, and either value is not an integer, an error will be returned. The values for numeric comparisons must be in the range of a signed 32-bit integer for versions prior to V3.5.5.3.0. Later versions support signed 64-bit integer compares.

Simple expressions:

<var> Truth value of variable <var>

!<var> Negated truth value of variable <var>

<var> == <value> TRUE if variable <var> matches the string <value>; note that case is not significant.

<var> != <value> TRUE if the string contained by variable <var> does not match the string <value>; note that case is not significant.

<var> > <num value> TRUE if the numeric value of variable <var> is greater than the numeric value of the string <value>.

<var> < <num value> TRUE if the numeric value of variable <var> is less than the numeric value of the string <value>.

<var> >= <num value> TRUE if the numeric value of variable <var> is greater than or equal to the numeric value of the string <value>.

<var> <= <num value> TRUE if the numeric value of variable <var> is less than or equal to the numeric value of the string <value>.

Complex expressions:

<expr> represents one of the above simple expressions, or another complex expression.

<expr1> AND <expr2> TRUE if both <expr1> and <expr2> both evaluate to TRUE values.

<expr1> OR <expr2> TRUE if either <expr1> or <expr2> (or both) evaluate to TRUE values. If <expr1> is TRUE, the evaluations falls through immediately without evaluating the second expression.

Additional Keywords:

NOT <expr> TRUE if <expr> evaluates to FALSE (boolean negation.)

DEFINED <var> TRUE if <var> exists, regardless of value (may be null).

IS_GLOBAL <var> TRUE if a global variable by the name of <var> exists.

IS_LOCAL <var> TRUE if a local variable by the name of <var> exists.

Additional Notes:

Parentheses should be used in compound boolean expressions; there is no order of preference, expressions are evaluated left-to-right. For example:

((var1 == 1 AND var2 != foobar) OR var3 >= 4)

For the truth value of the specified Ora*Tst variable <var>, the variable is implicitly dereferenced and evaluated. The string `0' (zero), the empty string (unknown, UNSET/UNLET) and the string `FALSE' (any case) evaluates to FALSE. All other values evaluate to TRUE. Note that when using variables in the value part (RHS) of the expression (<value>), variables will need to be explicitly dereferenced by the user.

The negation operator `!' or NOT toggles the boolean value of the Ora*Tst variable <var>, or the following boolean expression.

2.41.4 Notes

The DEFINED test can be used to determine if a variable has ever been created by LET, SET or as the result of some other command. A variable that is DEFINED may have a null or empty value. Even an UNLET or UNSET variable is considered DEFINED.

This test may be useful for assigning default values to variables. For example:

```
if not defined tbl_size
   set tbl_size 5000
endif
```

However, if there is the possibility that the variable might have been created previously but has a null value, then the following check is more appropriate as it handles the case where the variable is already defined but has a null value:

```
unset null_var # side-effect: creates var with a null value.
if tbl_size == ^null_var^
   # tbl_size is not defined or has a null value
   set tbl_size 5000
endif
```

There is a limit of 63 tokens in the "{conditional statement}" expression, including parenthesis. Thus, the practical limit is about 15 expressions (i.e IF (var == 1 or var == 2 or ...)). A TST*WRN-23 message with "IF expression too complex" is issued when the limit is exceeded.

The ELSEIF construct can be used to extend the limit in a manner such as:

```
if (var == 1 or var == 2 or var == 3 or ...)
   ...
elseif (var == 16 or var == 17 or ...)
   ...
else
   ...
endif
```

2.41.5 Availability

Ora*Tst built-in command since V1.0.6. Expanded in V3.0

DEFINED test available at V3.5.5.3.0.

2.41.6 Prerequisites

The values for numeric comparisons must be in the range of a signed 32-bit integer.

2.41.7 See Also

colon-if, else, elseif, and endif Ora*Tst commands.

2.41.8 Known Bugs

Ora*Tst (#2680168) bug: An if statement of “if (!A and !B)” fails to execute correctly. The work-around is to code “if (!A) and (!B)”. Fixed at Version 3.5.4.1.0.

Ora*Tst (#4893014) bug: This bug effects any compound IF statement where an operand is negated (using ! or not) and there are subsequent operands in the same IF. Fixed at Version 3.5.4.1.0.

There is a limit to the number of expressions in a single IF statement. See the Notes section of this command for a work-around.

Numeric comparisons are limited to signed 32-bit integers for versions prior to V3.5.5.3.0.

2.42 IMPORT

2.42.1 Synopsis

```
import <var> [nowarn]
```

2.42.2 Description

The variable <var> is imported from the OS environment with its current value. A global Ora*Tst variable <var> is set to the respective value.

Note that although IMPORT sets a global Ora*Tst variable, the EXPORT command can use either a local or a global variable. If you export a local, change it in the outside environment, and re-import it, you will not see the change since the original local variable will hide the new global variable, according to normal scoping rules.

2.42.3 Options

nowarn

The nowarn option indicates to not issue OraTst warning "TST*WRN-41: Unrecognized environmental variable" when the environment variable does not exist. The default is to issue a warning. Consider using NOWARN instead of globally suppressing "TST*WRN-41".

If nowarn is specified and the environment variable does not exist, the builtin TST_STATUS variable is set to a non-0 value. This can be tested to see if the variable was imported.

2.42.4 Notes

The <var> name is case insensitive. Ora*Tst automatically changes the <var> string to uppercase during processing. Thus, regardless of how <var> is specified, the variable defined in the environment must be uppercase. Lower or mixed case environment variable names may not be portable and are not supported.

While using import, avoid an export of the same variable unless necessary. That is, programmers often code an export after an import simply because that's what was done in the test script that they used as a model. Only export variables that are absolutely required. See the EXPORT command for more details.

If the environment variable to be imported does not exist, no Ora*Tst variable <var> is created. In additon, the builtin TST_STATUS variable is set to a non-0 value.

2.42.5 Availability

Ora*Tst built-in command since V1.0.6.

The nowarn option is available at V3.5.5.3.0.

2.42.6 Prerequisites

The OS environment variable <var> should be defined and be upper case.

2.42.7 See Also

export Ora*Tst command.

2.42.8 Known Bugs

None.

2.43 INCLUDE

2.43.1 Synopsis

```
include [<area>:]<scr_name>[.TSC]
```

2.43.2 Description

Transfers control-flow in a nested fashion. The contents of the specified test script <scr_name> is executed. The filetype of script <scr_name> is forced to be TSC. The default area for <area> is T_SOURCE.

2.43.3 Options

None.

2.43.4 Notes

If an EXIT command is executed from within an INCLUDEd script, the logic flow stops and control returns to the parent script that included the file. Since an INCLUDEd files works in a similar manner as "inline code" one might be expected that the EXIT command would terminate the parent script. It does not.

2.43.5 Availability

Ora*Tst built-in command since V1.0.6.

2.43.6 Prerequisites

The maximum level of include nesting is 5.

2.43.7 See Also

run, runtest Ora*Tst commands.

2.43.8 Known Bugs

None.

2.44 INCR

2.44.1 Synopsis

```
incr [GLOBAL|LOCAL] <var> [<step>] [nolog]
```

2.44.2 Description

Increments the variable <var> by 1, or the amount of <step> if specified. <step> is a literal numeric value that can be negative.

2.44.3 Options

LOCAL

If specified, the variable <var> is interpreted as a local variable only.

GLOBAL

If specified, the variable <var> is interpreted as a global variable only. (V3.0)

step

A numeric literal value to increment <var>. The default is 1

nolog

The optional nolog parameter indicates to not log the command execution in the OraTst log (.tlg) file. The default is to log the command.

2.44.4 Availability

Ora*Tst built-in command since V1.0.6. The LOCAL option was added in V2.0, and the GLOBAL and auto-scoping features were added in V3.0.

The nolog option is available at V3.5.5.3.0.

2.44.5 Prerequisites

The variable <var> must have a string value that can be interpreted as an integer. The range of legal values is -2^31 to 2^31 - 1 (signed 32-bit integer). If no variable scope option is specified, the command will “automatically” determine the type for you by looking for a variable of that name starting at the local scope, and if not found will look for a global variable. The variable will be set as the same scope found. If <var> does not exist, an error message will be issued.

Remember that a local variable will hide a global variable of the same name, according to scoping rules. You can use the GETGLOBAL command to bypass the scoping rules and retrieve a hidden global.

2.44.6 See Also

add, decr, subtract, getglobal Ora*Tst commands.

For more detailed information on variables, refer to the Ora*Tst Variables section in the Ora*Tst Installation and User’s Guide.

2.44.7 Known Bugs

None.

2.45 INIT (macro)

2.45.1 Synopsis

```
init
```

2.45.2 Description

Removes all files in the working area T_WORK except for the current test log file. The area itself is not deleted.

2.45.3 Options

None.

2.45.4 Notes

The default version of this macro does nothing on Windows platforms. That is, no files are removed under T_WORK. If you need to remove files under T_WORK on Windows, override the default version of this macro (ctm04.tsc) or use the CLEANOUT command directly.

2.45.5 Availability

Ora*Tst built-in command since V1.0.6. This is an automatic default macro definition. An init macro might be part of the product-specific test suite; in that case this macro definition is to overwrite the default definition.

2.45.6 Prerequisites

The macro must be (pre)defined.

2.45.7 See Also

For more detailed information on macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User's Guide.

For more information on the implementation of the portable Ora*Tst default macro, refer to the file ctm04.tsc.

cleanout Ora*Tst command.

2.45.8 Known Bugs

None.

2.46 INSTANCE

2.46.1 Synopsis

```
instance [<sid_suffix>] [LOCAL|GLOBAL]
```

2.46.2 Description

Appends the suffix <sid_suffix> to the existing Oracle system identifier (SID) and exports it to the OS environment. If <sid_suffix> is not specified, the original Oracle SID is unchanged. Also sets the Ora*Tst variable TST_ORASID to the instance name and the Ora*Tst variable TST_SIDSFX to <sid_suffix> if specified.

On most platforms the Oracle SID described here is defined by the ORACLE_SID environment variable and is automatically imported at Ora*Tst startup. The variables TST_ORASID and TST_SIDSFX (if defined) are "read-only" type variables that represent the current values of ORACLE_SID and <sid_suffix>, respectively. To clarify, TST_ORASID and TST_SIDSFX are set as the result of the INSTANCE command and are not intended to be modified by users using LET/SET. They normally cannot be used to change the value of ORACLE_SID. See the Notes section for more information.

2.46.3 Options

LOCAL

Sets the local Ora*Tst variables TST_ORASID and TST_SIDSFX to the instance name and suffix (if specified) respectively. Previous values are restored upon exit of the current Ora*Tst run level. This is the default.

GLOBAL

Sets the global Ora*Tst variables TST_ORASID and TST_SIDSFX to the instance name and suffix (if specified) respectively.

2.46.4 Notes

Changing the value of TST_ORASID and/or TST_SIDSFX before calling INSTANCE has absolutely no effect on the value of ORACLE_SID. The INSTANCE command uses the value of the ORACLE_SID environment variable seen at Ora*Tst startup.

The TST_ORASID and TST_SIDSFX variables are NOT intended to be modified by users. These variables are set as the result of the INSTANCE command. However, an existing side-effect of the current RUN/RUNTEST logic is that the ORACLE_SID environment variable is updated at the end of nested test execution based on the value of TST_ORASID. Thus, if TST_ORASID has been changed by the user after calling INSTANCE and a subtest is run using RUN/RUNTEST, the ORACLE_SID environment variable is set to the value of TST_ORASID after execution of the subtest has completed and control has returned to the parent script. Do Not exploit this unintended behavior intentionally by changing TST_ORASID. This behavior is likely to be removed in the future. If you encounter such behavior, you can call the INSTANCE command again to reset the ORACLE_SID environment variable back to the original value. See bug 14009334 for more details.

Starting at Ora*Tst version 3.6.0.3.0, a new variable setting is available to override the side-effect described in the previous paragraph. By setting the TST_ORASID_READ_ONLY variable to a true value, the value of TST_ORASID is ignored on return from a subtest. Thus, the value of TST_ORASID is not used to set/reset the ORACLE_SID environment variable. The default value for TST_ORASID_READ_ONLY is false. A false value leaves the sie-effect behavior in place. The T_ORASID_READ_ONLY environment variable can be set before starting Ora*Tst as a replacement to using the TST_ORASID_READ_ONLY variable from within a test script.

2.46.5 Availability

Ora*Tst built-in command since V3.2.5.

2.46.6 Prerequisites

None.

2.46.7 Known Bugs

See bug 14009334 for details on how ORACLE_SID is affected by TST_ORASID.

2.47 ISACTIVE

2.47.1 Synopsis

```
isactive ^<process_id_var>^ <flag_var>
```

2.47.2 Description

Determines if a (child) Ora*Tst process previously created by the FORK Ora*Tst command is still active. If the process is active, <flag_var> is set to a TRUE value. If the process is not active, <flag_var> is set to a FALSE value.

2.47.3 Options

None.

2.47.4 Availability

Ora*Tst built-in command since V3.1.0.

Implementation of isactive is very system specific.

2.47.5 Prerequisites

The variable <process_id_var> must be set to a valid, port-specific process handle, i.e. a value of the global Ora*Tst variable ENTRY upon successful completion of a FORK Ora*Tst command.

2.47.6 See Also

fork, wait, sleep, and stop Ora*Tst commands.

2.47.7 Known Bugs

Bug 14198767: ISACTIVE not detecting terminated processes. Fixed at V3.6.0.2.0.

2.48 LDIFF

2.48.1 Synopsis

```
ldiff [<ref_area>:]<reffile>.<reftype>
        [<tst_area>:]<tstfile>.<tsttype> [NOMASK|MASK]
        [CASE|NOCASE] [BEGIN <string> [END <string>]] [BINARY]
        [CONTEXT <#>] [EQUBLANK] [EQUSPACE] [IGNABL] [IGNPBS]
        [IGNABS] [IGNTBS] [SORT]
```

2.48.2 Description

Performs a line-based comparison of the two files specified with optional regular expression matching.

The first file specified is interpreted as the reference file <reffile> of type <reftype>. The second file is the generated or test file <tstfile> of type <tsttype>. The default area for both the reference area <ref_area> and the test area <tst_area> is T_WORK. If the types <reftype> and/or <tsttype> are omitted, they default to LOG.

If the comparison succeeds, i.e., no differences are found, an empty SUC file with the name <tstfile> is created in T_WORK; if the comparison fails, the differences are reported in a DIF file with the name <tstfile> created in T_WORK.

In addition to the SUC and DIF files, after each LDIFF (and thus COMPARE) command, the Ora*Tst global variable TST_LDIFF will be set to the following return codes: 0=success, no differences; 1=differences; 2=error, not sure if differences exist (usually from a missing file, file error, etc.). It is significantly faster to check this variable rather than using FINDFILE to check for the resulting SUC or DIF file to determine whether a comparison has failed. If an error code is returned (i.e., TST_LDIFF=2), the Ora*Tst status variable TST_STATUS will contain the Ora*Tst error number.

Both existing SUC and DIF files with a basename <genfile> will be deleted and/or overwritten so that only the results from the most recent LDIFF will remain.

2.48.3 Options

NOMASK (default)

If specified (or allowed to default), LDIFF attempts an exact match of the corresponding lines from the reference file <reffile> and the generated/test file <tstfile>. If any differences are found they are reported in the DIF file.

MASK

If specified, assumes the reference file <reffile> contains masking regular expressions. The LDIFF logic first attempts an exact match of each line from the reference file <reffile> and the generated/test file <tstfile> as if NOMASK was specified. If that succeeds it moves on to the next line in both files. If that fails, then LDIFF re-interprets the line from the reference file <reffile> as a regular expression (mask) and tries the comparison again. If no match is found then the difference is reported in the DIF file. See the "Regular Expressions" section in the Ora*Tst Installation and User's Guide for details on the masking characters. Also see the **Notes** section for additional information.

CASE (default)

If specified, files are compared in a case-sensitive manner.

NOCASE

If specified, all characters are lowercased before comparison, making the comparison (MASK and NOMASK both) case-insensitive. Note that this is a simple ASCII lowercasing.

BEGIN <string>

If specified, all lines that begin with <string> will be ignored. When this is used together with END, a whole section can be filtered out before comparison occurs. Note: If specified, the options NOCASE, EQUSPACE, IGNPBS, IGNABS and IGNTBS are **not** applied to <string>.

LDIFF does not limit the number of BEGIN statements (except that the entire LDIFF command cannot exceed the maximum command line length of the platform). However, if LDIFF is invoked through a customized COMPARE macro, that macro may limit the number of BEGIN statements. If you are specifying more than five (5) BEGIN statements consider using the Ora*Tst FILTER command to filter out unwanted lines before the comparison.

END <string>

This option needs to be specified with BEGIN. When specified in the form of BEGIN <str1> END <str2>, the section marked by the first occurrence of any line that starts with str1 and the first occurrence of any line that starts with str2 will be ignored. Note that str2 must exist in this case or a warning message will be issued. Multiple BEGIN statements or BEGIN/END pairs are allowed.

LDIFF does not limit the number of BEGIN/END pairs (except that the entire LDIFF command cannot exceed the maximum command line length of the platform). However, if LDIFF is invoked through a customized COMPARE macro, that macro may limit the number of BEGIN/END statements. If you are specifying more than five (5) BEGIN/END pairs consider using the Ora*Tst FILTER command to filter out unwanted lines before the comparison.

BINARY

If specified, will compare the files in BINARY mode. Streams of bytes are read from each file and will be compared byte by byte. Masking is not supported in binary mode. **Note:** when a difference is detected in binary mode an **empty DIF file** is created. You have to manually examine the files being compared to determine the differences.

CONTEXT <#>

If specified, the number of lines <#> to be displayed before and after a difference in order to provide more information about the context where the difference occurs.

EQUBLANK

If specified, treat strings of blank lines as equivalent.

EQUSPACE

If specified, one or more blank spaces will be treated as equivalent.

IGNABL

If specified, LDIFF will ignore all blank lines.

IGNPBS

If specified, LDIFF will ignore all preceding blank space.

IGNABS

If specified, LDIFF will ignore all blank space.

IGNTBS

If specified, LDIFF will ignore all trailing blank space.

SORT

If specified, the files are sorted before comparison. NOCASE is ignored when using the sort option.

2.48.4 Notes

The two specified files should exist. In the case when either of the specified files cannot be found, no SUC or DIF file is created. Thus, consider testing the TST_LDIFF internal variable to verify success/failure instead of or in combination with searching for SUC or DIF files.

With masking enabled the maximum length of a single line in the reference file cannot exceed 128K bytes.

With masking enabled the reference file is expected to contain regular expressions as described in the Ora*Tst Installation and User's Guide. If the reference file contains regular expression meta characters that are not intended to be part of a matching expression, you must escape these characters using '\' . Failing to do so may result in a diff or other **unpredictable behavior**, including failure of the LDIFF command itself.

Avoid specifying MASK when there is no intention to actually mask lines in the generated file. Doing so in this situation simply wastes CPU time.

Even with masking enabled, LDIFF first attempts an exact match for each line of the files being compared (as if nomask was specified). If a difference is found, it then re-interprets the line in the reference file <reffile> as a regular expression and performs the comparison again.

The options NOCASE, EQUSPACE, IGNPBS, IGNABS and IGNTBS, if specified, are **not** applied to the <string> of BEGIN/END. Furthermore, the matching for BEGIN/END takes place before NOCASE and after EQUSPACE, IGNPBS, IGNABS and IGNTBS have been applied to the files being compared.

Starting with Ora*Tst Version 3.5.5.2.0, all the LDIFF parameters are written to the test log (tlg). Previously only the CASE and MASK status was indicated in the log. For some parameters such as BEGIN/END the values displayed in the log may not match the command line exactly depending on the use of quotes and spaces.

The maximum file size is currently limited to 200,000 lines where a line may be 1 - 128K bytes in length. With very large line lengths the memory on the machine may be exhausted before the line limit is reached. With masking enabled the size is further limited by the mask expression and the contents of the file being compared. Thus, the actual limit may vary.

A reference or test file name longer than 128 bytes is truncated without warning.

At version 3.6.0.3.0, a diff "map" file is created each time there is a diff. The diff map file (file extension .dif_map) contains useful information to assist with diff triaging and resolution. It is a plain text file containing the test script name and line number of the comparison that triggered the diff. The nesting of test scripts is also shown to make it easier to trace the logic to the causing script. In addition, the reference and generated file names are included, along with the .tlg file and line number. The diff map file creation can be disabled using the TST_DISABLE_DIFF_MAP variable. See Section 4 Advanced Concepts and Features in the Installation and User's Guide for more information.

2.48.5 Availability

Ora*Tst built-in command since V1.0.6.

CASE / NOCASE options added in V2.1.

TST_LDIFF variable now set for all return cases in V2.1.1 (to 2 if an error occurs); previously, errors left the variable untouched.

A diff "map" file (.dif_map) is created for each diff starting at V3.6.0.3.0.

2.48.6 Prerequisites

The two specified files should exist. No SUC or DIF file with the name <tstfile> should exist in T_WORK or a warning will be issued. In the case when either of the specified files cannot be found, no SUC or DIF file will be created.

2.48.7 See Also

For more information on regular expressions, refer to the Regular Expressions section of the Ora*Tst Installation and User's Guide.

compare, filter Ora*Tst commands.

2.48.8 Known Bugs

Bug 6963878: BEGIN masking doesn't work with whitespace if IGNABS is set.

Bug 7499670: Count expressions do not work with groups. Fixed at V3.5.3.3.0.

Bug 8241366: Reporting of dif file name in .tlg may be incorrect. Fixed at V3.5.5.3.0.

Bug 8512439: Possible change in behavior with NOCASE and older RDBMS releases.

Bug 11847380: .SUC created even when both files are too large to compare. Fixed at V3.6.0.0.0.

Bug 14324478: Counted expression of {n,1} incorrectly matches more than the max of 1. Fixed at V3.6.0.3.0.

2.49 LET

2.49.1 Synopsis

```
let <var> [<value>]
```

2.49.2 Description

Creates (if necessary) and sets the local Ora*Tst variable <var> to the value <value>. If <value> is not specified, the variable <var> defaults to TRUE.

2.49.3 Options

None.

2.49.4 Notes

It is possible to have a Global (set) variable of the same name. In this situation, the local (let) variable takes precedence. Use the getglobal command to obtain the value of the global variable of the same name. It is best to avoid name conflicts between global and local variables when possible.

Loal (let) variables are visible in subtests or macro calls but not in parent scripts.

2.49.5 Availability

Ora*Tst built-in command since V1.0.6.

Changes from previous version:

`LET var' without a value defaults to a true value. If a value is passed that evaluates to an empty string, such as `LET var ^nullvar^', the variable will be set to an empty string.

When specifying a numeric value many Ora*Tst commands such as INCR, DECR and IF limit the range to a signed 32-bit integer.

2.49.6 Prerequisites

None.

2.49.7 See Also

For more detailed information on variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

unlet, set, unset, getlocal, getglobal Ora*Tst commands.

2.49.8 Known Bugs

None.

2.50 LOOKFOR

2.50.1 Synopsis

```
lookfor [<area>:]<name>.<type> [<timeout>]
```

2.50.2 Description

Primitive synchronization command. Keeps looking for the specified file <name> of type <type> in the area <area> for <timeout> seconds. If no <area> is specified it defaults to T_WORK. The default timeout is 3600 seconds (60 min).

As with the UNTIL command, once the file is found, it is removed to allow for future synchronization events.

2.50.3 Options

None.

2.50.4 Availability

Ora*Tst built-in command since V1.0.6. It has to be determined if this primitive sync mechanism is portable.

2.50.5 Potential changes in future versions

The lookfor Ora*Tst command will be reconsidered with regard to portability and efficiency. It might be discontinued in a future version.

2.50.6 Prerequisites

If specified, the area must exist.

If <timeout> is specified the value must be greater than 0 and less than 2,147,483,648.

2.50.7 See Also

post, until Ora*Tst commands.

2.50.8 Known Bugs

None.

2.51 LOOP

2.51.1 Synopsis

```
loop
   ...
   { [ break | continue] }
   ...
endloop
```

2.51.2 Description

The loop body is executed repeatedly until the break Ora*Tst command is encountered. As soon as a BREAK Ora*Tst command is encountered, the flow of control continues with the first statement following the matching endloop Ora*Tst command. If a CONTINUE Ora*Tst command is encountered, the flow of control continues with the first statement in the loop body (next iteration).

2.51.3 Options

None.

2.51.4 Availability

Ora*Tst built-in command since V1.0.6.

2.51.5 Prerequisites

The maximum loop nesting level is 10.

2.51.6 See Also

for, endloop, break, continue and block_begin Ora*Tst commands.

2.51.7 Known Bugs

None.

2.52 MKDIR

2.52.1 Synopsis

```
mkdir <new_area> [<existing_area>:][<name>]
```

2.52.2 Description

Defines a new area <new_area>. and if necessary, creates the physical directory defined by <existing_area>:<name>. After successful execution the newly created area <new_area> is a valid Ora*Tst area specification for the remainder of the test run (i.e., this area can subsequently be referred to by the new name <new_area>).

An <existing_area> and directory <name> may be specified to indicate where the <new_area> should be created. The default <existing_area> is T_WORK. The default <name> is <new_area>.

If the physical directory of the newly defined area already exists, no new directory is created and the content of such directory is not deleted or modified in any way. Neither the directory nor any files contained in it are deleted upon completion of the test run.

2.52.3 Options

None.

2.52.4 Notes

The specification of any portion of a platform specific path component (i.e. dir1/dir2) for <name> is not portable and is not supported.

This command is currently the only way of defining a new Ora*Tst AREA specification from within a test script. AREA specifications are often defined in the environment or initialization file before invoking Ora*Tst.

Examples:

1. The following creates a new area named "work_log". By using all the defaults the <existing_area> is T_WORK, the new area name is WORK_LOG and the new physical directory name is "work_log":

   ```
   MKDIR work_log

   CPFILE T_SOURCE:testlog.log WORK_LOG:testloga.log
   ```

2. A multiple directory "path" needs to be created to gather results for a particular portion of a test. The physical directory hierarchy starts from the existing T_WORK directory and looks like "work/testgrp1/grp1abc" on UNIX. Each directory "component" in the physical "path" must be created/referenced with a separate MKDIR command.

   ```
   MKDIR T_GRP1 T_WORK:testgrp1

   MKDIR T_GRP1ABC T_GRP1:grp1abc

   CPFILE T_SOURCE:foo.log T_GRP1ABC:foog1abc.log
   ```

3. For commands that require an existing directory name, the directory referred to in the MKDIR command should be referred to using the existing area and directory name. In this example OSDDIRNAME should not refer to area T_ARC1 because OSDDIRNAME requires a directory name to be given:

```
MKDIR T_ARC1 T_DBS:tklm_arc1
OSDDIRNAME T_DBS:tklm_arc1 name_arc1
```

2.52.5 Availability

Ora*Tst built-in command since V1.0.6. In V3.1.0, <existing_area> and <name> may be specified to control where the new area is created.

2.52.6 Potential changes in future versions

The mkdir Ora*Tst command will be reconsidered with regard to portability. It might be discontinued in a future version.

2.52.7 Prerequisites

The <new_area> name must not already be defined to Ora*Tst.

2.52.8 See Also

For more detailed information on the Ora*Tst areas concept, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

cleanout, finddir, rmdir, osddirname, osdfilename Ora*Tst commands.

2.52.9 Known Bugs

None.

2.53 MODULO

2.53.1 Synopsis

```
MODULO [LOCAL|GLOBAL] <varname> <divisor> [<target_var>]
       [log|nolog]
```

2.53.2 Description

Determines the remainder of the integer division of <varname> by <divisor> and stores the result in <varname>. If variable <target_var> is specified it is updated with the result and <varname> is left unchanged. The <varname> must be an Ora*Tst variable and <divisor> is a literal value or dereferenced variable. The <varname> value and <divisor> must be integers in the range of signed 32bit values. The result of the division is truncated towards 0 when calculating the remainder.

If <varname> does not exist it is created with an initial value of 0 and a default scope of global.

If <target_var> is specified but does not exist it is created in the same scope as <varname>.

The sign of the result matches the sign of <varname>.

2.53.3 Options

LOCAL| GLOBAL

LOCAL: If local is specified, the selection and/or creation of <varname> and optionally <target_var> is local scope only.

GLOBAL: If global is specified, the selection and/or creation of <varname> and optionally <target_var> is global scope only.

If neither local or global is specified, selection of <varname> follows normal scoping rules.

target_var

Optional variable name that is updated with the result and <varname> is left unchanged. If it does not exist it is created in the same scope as <varname>.

log|nolog

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.53.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.53.5 Prerequisites

The <varname> value and <divisor> must be valid signed 32bit integers.

If <divisor> is 0 a "TST*WRN-23" message is issued.

If the result exceeds the value of a signed 32bit integer a "TST*WRN-66" message is issued.

2.53.6 See Also

add, subtract, multiply, divide, incr and decr Ora*Tst commands.

2.53.7 Known Bugs

None.

2.54 MULTIPLY

2.54.1 Synopsis

```
MULTIPLY [LOCAL|GLOBAL] <varname> <multiplier> [<target_var>]
         [log|nolog]
```

2.54.2 Description

Multiplies <varname> by <multiplier> and stores the result in <varname>. If variable <target_var> is specified it is updated with the result and <varname> is left unchanged. The <varname> must be an Ora*tst variable and <multiplier> is a literal value or dereferenced variable. The <varname> value and <multiplier> must be integers in the range of signed 32bit values.

If <varname> does not exist it is created with an initial value of 0 and a default scope of global.

If <target_var> is specified but does not exist it is created in the same scope as <varname>.

2.54.3 Options

LOCAL| GLOBAL

LOCAL: If local is specified, the selection and/or creation of <varname> and optionally <target_var> is local scope only.

GLOBAL: If global is specified, the selection and/or creation of <varname> and optionally <target_var> is global scope only.

If neither local or global is specified, selection of <varname> follows normal scoping rules.

target_var

Optional variable name that is updated with the result and <varname> is left unchanged. If it does not exist it is created in the same scope as <varname>.

log|nolog

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.54.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.54.5 Prerequisites

The <varname> value and <multiplier> must be valid signed 32bit integers.

If the result exceeds the value of a signed 32bit integer a "TST*WRN-66" message is issued.

2.54.6 See Also

add, subtract, divide, modulo, incr and decr Ora*Tst commands.

2.54.7 Known Bugs

None.

2.55 MVFILE

2.55.1 Synopsis

```
mvfile [<srcarea>:]<srcfile>.<srctype>
       [<tgtarea>:]<tgtfile>.<tgttype>
```

2.55.2 Description

Renames a file <srcfile> of type <srctype> in area <srcarea> to a file <tgtfile> of type <tgttype> in area <tgtarea>. The default area for both area <srcarea> and area <tgtarea> is the working area T_WORK.

The specification of a target area only is sufficient.

2.55.3 Options

None.

2.55.4 Notes

The specification of any portion of a platform specific path component (i.e. dirname / filename) for <srcfile> or <tgtfile> is not portable and is not supported.

Source or target file names longer than 128 bytes are truncated without warning.

2.55.5 Availability

Ora*Tst built-in command since V1.0.6.

2.55.6 Prerequisites

The file <srcfile> of type <srctype> in area <srcarea> should exist.

The target file specification must be different from the source file specification.

2.55.7 See Also

cpfile, rmfile and findfile Ora*Tst commands.

For more detailed information on Ora*Tst file specifications, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

2.55.8 Known Bugs

None.

2.56 OSDDIRNAME

2.56.1 Synopsis

```
osddirname[<area>:]<dirname> <local_var>
```

2.56.2 Description

Converts an Ora*Tst portable directory specification [<area>:]<dirname> to the full native, OS-specific directory name and stores it in the specified local variable <local_var>. This is useful for echoing a dirname to a log or script file, or passing a dirname parameter to an external command, as external OS commands and utilities will only understand native OS-specific filenames, and not Ora*Tst internal, portable concepts like areas and filetypes.

Note: This command treats <dirname> differently than other Ora*Tst file related commands in that it is case sensitive. This is a bug , but existing usage prevents changing the behavior. With other Ora*Tst portable file specifications, the dirname is treated in a case-insensitive manner; on UNIX, this means the name is lowercased before being translated to the OS-specific dirname.

The actual OS-specific directory returned corresponding to the specified area is not altered.

If not specified, the area <area> defaults to T_WORK.

The OSDDIRNAME command searches the list of directories in <area> until it finds the specified <dirname> and returns the directory path. If the directory <dirname> is not found, <local_var> is set to a FALSE value.

The <dirname> is not currently optional. However, it is not flagged as an error to leave it blank because of existing usage. This can trigger a portability problem on Windows on versions prior to 3.5.5.3.0. See the Known Bugs section.

Note that this is not guaranteed to be meaningful for all ports, and is not 100% portable. Please wrap this kind of usage in an OSD macro to aid script porters.

2.56.3 Options

None.

2.56.4 Notes

The specification of any portion of a platform specific path component (i.e. dir1/dir2) for <dirname> is not portable and is not supported.

2.56.5 Availability

Ora*Tst built-in command since V3.4.0.

2.56.6 Prerequisites

None.

2.56.7 See Also

osdfilename, execute, runutl, mkdir Ora*Tst commands.

2.56.8 Known Bugs

Bug 3893063 - changes command to be case insensitive. Version 3.5.4.0.0 only.

Bug 5717502 - restores case sensitive behavior, V3.5.4.1.0 and later.

Bug 7575977 - unable to locate directory on Windows when <dirname> is not specified. Fixed at V3.5.5.3.0.

2.57 OSDFILENAME

2.57.1 Synopsis

```
osdfilename [<area>:]<file>.<type> <local_var>
```

2.57.2 Description

Converts an Ora*Tst portable file specification [<area>:]<file>.<type> to the full native, OS-specific filename (directory pathname and filename) and stores it in the specified local Ora*Tst variable <local_var>. This is useful for echoing a filename to a log or script file, or passing a filename parameter to an external command, as external OS commands and utilities will only understand native OS-specific filenames, and not Ora*Tst internal, portable concepts like areas and filetypes.

As with other Ora*Tst portable file specifications, the filename is treated in a case-insensitive manner; on UNIX, this means the basename etc. is lowercased before being translated to the OS-specific filename. The actual OS-specific directory returned corresponding to the specified area is not altered.

If not specified, the default area is T_WORK.

The OSDFILENAME command uses a mix of the file mapping algorithms to determine the correct pathname from the supplied area, as Ora*Tst does not know if the file is a new one to be created, or an existing file to be found. Thus, it will search the list of directories listed in the area until it finds the file, and will return that directory as the path. If the file is not found (or only the area is specified, as above), it is treated as a file to be created, and thus the directory returned is the first one listed in the area path. See the section on File Mapping for more information.

Note that this command will also work to convert an area into an OS-specific directory, if this is meaningful for your port, if you simply specify the area without any basename or filetype, i.e. OSDFILENAME T_WORK:. On UNIX, this leaves the directory stored in the variable specified, with an appended `/'. If the area path lists several directories, it will return the first one. Note that this is not guaranteed to be meaningful for all ports, and is not 100% portable. Please wrap this kind of usage in an OSD macro to aid script porters.

Any filename to be passed to the external (OSD) environment, via the ECHO/RUNUTL commands and the like, must use OSDFILENAME to first convert a portable file specification, in order to make the test script portable. There should be no hard-coded references to raw OSD filenames in a test script (this can even include a simple basename plus extension, as some platforms do not support extensions). All filename references should either be Ora*Tst portable file specifications, or translated through OSDFILENAME; OSD file references should also be encapsulated in an OSD macro, if possible. Ora*Tst commands like CPFILE, of course, use the portable file specification directly, so you do not need to use OSDFILENAME for them.

2.57.3 Options

None.

2.57.4 Notes

The specification of any portion of a platform specific path component (i.e. dirname/filename) for <file> is not portable and is not supported.

2.57.5 Availability

Ora*Tst built-in command since V2.1.

2.57.6 Prerequisites

None.

2.57.7 Examples

```
Running UNIX; T_FOO = /home/sqlstuff;
HOME and PATH env. vars. set normally
foo.tsc file:
-------------------------------------------------
...
osdfilename T_FOO:testsql.sql foovar
echo Variable “foovar” is ^foovar^
execute T_SYSTEM:sqldba.exe command=@^foovar^

echo >HOME:sqlsetup.com
   > echo “Setting up SQLDBA environment...”
   > sqldba command=@^foovar^
endecho
...



foo.tlg file:
-------------------------------------------------
...
Variable “foovar” is /home/sqlstuff/testsql.sql
...
> EXECARGV /private/RDBMS/sqldba
         command=@/home/sqlstuff/testsql.sql

$HOME/sqlsetup.sh:
-------------------------------------------------
echo “Setting up SQLDBA environment...”
sqldba command=@/home/sqlstuff/testsql.sql
```

2.57.8 See Also

osddirname, execute, runutl Ora*Tst commands.

2.57.9 Known Bugs

None.

2.58 POST

2.58.1 Synopsis

```
post <event_name>
```

2.58.2 Description

Publicly posts a unique synchronization event identified by the name <event_name> and continues. No feedback is expected or returned by the post Ora*Tst command with regard to whether the event was received (or in other words consumed) by another Ora*Tst process. Event names <event name> must adhere to the naming conventions of file basenames.

2.58.3 Options

None.

2.58.4 Availability

Ora*Tst built-in command since V1.0.6.

2.58.5 Potential changes in future versions

The implementation of this basic synchronization mechanism is currently being reviewed and subject to change. Please do not rely on port-specific implementation details.

2.58.6 Prerequisites

An event must not be posted already.

2.58.7 See Also

lookfor, until Ora*Tst commands.

2.58.8 Known Bugs

Generic bug (#): Ora*Tst currently accepts an optional restrictive specification, such as [<group>:]<event_name>. This is a due to an outdated implementation and is no longer supported.

Generic bug (#): Re-posting an event is destructive and no error / warning is issued.

2.59 PUTFILE

2.59.1 Synopsis

```
putfile <h_remotevar> [<local_srcarea>:]<srcfile>.<srctype>
        [<remote_destarea>:][<destfile>.<desttype>]
        [legacy|text|binary]
```

2.59.2 Description

Copies a file <srcfile> of type <srctype> in area <local_srcarea> from the local host to file <destfile> of type <desttype> in area <remote_destarea> on the remote host <h_remotevar>. The default area for both <local_srcarea> and <remote_destarea> is T_WORK. The <h_remotevar> must be a previously defined Ora*Tst variable specifying a remote machine and port number that is running Ora*Tst in server mode.

Legacy is the default transfer mode.

2.59.3 Options

LEGACY (default)

If specified (or allowed to default), legacy mode supports the copy of text files. No data translation of any kind is performed between hosts. This implies that the source and target platforms should be the same OS type. Null bytes (binary zeros) are NOT allowed in the data. This is the mode of operation available in Ora*Tst versions prior to 3.6.0.0.0.

TEXT

Text mode transfers text files, correcting the end of line (EOL) character(s) depending on the platform. It is expected that UNIX, Linux and MAC OS X based platforms use an EOL character of LF (line feed, hex 0A). It is expected that Windows platforms use EOL characters CRLF (carriage return, hex 0D, line feed, hex 0A). When transferring text files between hosts of different types (UNIX-like and Windows) the EOL is updated as appropriate for the platform.

Text mode is only supported on ASCII platforms. Transferring files between ASCII/EBCDIC platforms may cause unpredicatable results and is not currently supported.

No other translation of any kind is supported in text mode. No character set translation is made or supported. Null bytes (binary zeros) are NOT allowed in the data.

BINARY

Binary mode transfers files without data translation of any kind. The data can contain Null bytes (binary zeros).

2.59.4 Notes

The legacy, text & binary options are new for Ora*Tst version 3.6.0.0.0. Prior versions only support the equivalent of legacy mode.

Compatibility Note: As of Oratst Version 3.5.3.3.0, this command is not downward compatible with previous Oratst versions. Both the client (host running putfile) and server (oratst -server) versions must be at Version 3.5.3.3.0 and above -OR- Version 3.5.3.2.0 and below to function properly. Mixing the client and server versions between 3.5.3.3.0 (and newer) and 3.5.3.2.0 (and older) will cause unpredicable results. When using a 3.6.0.0.0 or later version client against a pre 3.6.0.0.0 server, only the legacy mode flag is supported.

2.59.5 Availability

Ora*Tst built-in command since V3.3.0

The legacy, text & binary modes are available starting with V3.6.0.0.0.

2.59.6 Prerequisites

The remotehost must have Ora*Tst running is server mode (oratst -server <portnumber>). The <h_remotevar> must be a previously defined Ora*Tst variable specifying the remote host in the format <hostname:portnumber>. The remote host area <destarea> must exist and be defined to the remotely running Ora*Tst server.

2.59.7 See Also

The Ora*Tst Installation and User's guide for details on the Ora*Tst -server command.

getfile, remotefindfile, runremote Ora*Tst commands.

2.59.8 Known Bugs

Ora*Tst bug 6070919 - Intermittent failures with Getfile / Putfile. This bug affects version 3.5.3.3.0 thru 3.5.4.1.0 and is fixed starting at 3.5.4.1.2.

2.60 REINIT

2.60.1 Synopsis

```
reinit
```

2.60.2 Description

Reinitializes all global variables to their initial startup values. The command can only be run from the top most run level.

Example:

runtest a

reinit # clear global variables set by test a

runtest b

2.60.3 Options

None.

2.60.4 Notes

Avoid using this command as it may not be supported in future releases.

2.60.5 Availability

Ora*Tst built-in command since V3.3.1

2.60.6 Prerequisites

Can only be run from the top run level.

2.60.7 Known Bugs

None.

2.61 REMOTEFINDFILE

2.61.1 Synopsis

```
remotefindfile <h_remotevar> [<area>:]<name>.<type> <local_var>
```

2.61.2 Description

Determines whether the specified file <name> of type <type> exists in the area <area> on a remote host defined by <h_remotevar>. If it does, a local variable <local_var> is set to a value evaluating to TRUE, otherwise to a value evaluating to FALSE. If <area> is not specified, it defaults to T_WORK. The <h_remotevar> must be a previously defined Ora*Tst variable specifying a remote machine and port number that is running Ora*Tst in server mode.

2.61.3 Options

None.

2.61.4 Availability

Ora*Tst built-in command since V3.3.0

2.61.5 Prerequisites

The remotehost must have Ora*Tst running is server mode (oratst -server <portnumber>). The <h_remotevar> must be a previously defined Ora*Tst variable specifying the remote host in the format <hostname:portnumber>. The remote host area <destarea> must exist and be defined to the remotely running Ora*Tst server.

2.61.6 See Also

The Ora*Tst Installation and User's guide for details on the Ora*Tst -server command.

getfile, putfile, runremote Ora*Tst commands.

2.61.7 Known Bugs

None.

2.62 REMOTLOOKFOR

2.62.1 Synopsis

```
remotelookfor <h_remotevar> [<area>:]<name>.<type> [<timeout>]
```

2.62.2 Description

Primitive synchronization command. Keeps looking for the specified file <name> of type <type> in the area <area> for <timeout> seconds on a remote host defined by <h_remotevar>. If no <area> is specified it defaults to T_WORK. The default <timeout> is 3600 seconds (60 min).

As with the UNTIL command, once the file is found, it is removed to allow for future synchronization events.

2.62.3 Options

None.

2.62.4 Availability

Ora*Tst built-in command since V3.5.3.2.0.

2.62.5 Prerequisites

The remotehost must have Ora*Tst running is server mode (oratst -server <portnumber>). The <h_remotevar> must be a previously defined Ora*Tst variable specifying the remote host in the format <hostname:portnumber>. The remote host area <area> must exist and be defined to the remotely running Ora*Tst server.

2.62.6 See Also

The Ora*Tst Installation and User's guide for details on the Ora*Tst -server command.

getfile, putfile, remotefindfile, runremote, lookfor Ora*Tst commands.

2.62.7 Known Bugs

None.

2.63 REMOTERUN

Note: As of Ora*Tst Version 3.6.0.0.0 the remoterun command is no longer supported and has been permanently disabled. See bug 4056118 for details on how to convert existing remoterun commands to runremote.

2.63.1 Synopsis

```
remoterun <connect_string> <program>[.EXE|.COM] <arg>...
```

2.63.2 Description

Invokes an executable with the specified arguments on a remote machine.

The <connect_string> is an Oracle SID which identifies a server on a remote (or local) machine to which Ora*Tst will send the <program> and <args>. The remote machine does not have to be of the same type as the local machine. For example, the local machine can be a Sun while the remote machine can be a Windows NT machine as long as Ora*Tst V3.2 (or later) is available on both platforms.

For greater portability of tests, the <connect_string> should not be hard coded and scattered throughout your tests. Instead, store all <connect_string> values into variables which are initialized in your startup file. This will make it easier to update the <connect_strings> when tests are moved from one machine (or platform) to another.

<program> can be a linked program (portable filetype EXE) or an executable OSD script (portable filetype COM). On Unix, the server will expect to find the <program> in the bin directory under the ORACLE_HOME directory. Please refer to your platform specific documentation to see where the server will expect to find <program>.

Multiple <arg> can be specified. For portability, limit the total length of the arguments to 1024 characters.

The return code for <program> will be stored in the variable TST_REMOTE_STATUS. Currently, the return code is the only thing that is returned from the server.

2.63.3 Options

None.

2.63.4 Notes

This command is obsolete and no longer supported. See the RUNREMOTE command instead.

2.63.5 Availability

New Ora*Tst built-in command in V3.2.0.

2.63.6 Prerequisites

A server machine must be set up with:

- Sql*Net (listener needs to be running)
- Ora*Tst server
- <program> to be executed.

2.63.7 See Also

The Remoterun Setup section in the Ora*Tst Installation and User's Guide.

2.63.8 Known Bugs

None.

2.64 REPORT

2.64.1 Synopsis

```
report [BYCLASS|BYSCRIPT] [FLAT|TREED] [VERBOSE] [> <rpt_file>]
```

2.64.2 Description

Creates a textual report on stored result event data for the current test run so far.

Each event collected (all events except those specified by the customization variable TST_EVENT_SKIP at the time) and specified by event class name in the TST_REPORTON customization variable will be reported, including the event type and result, the test script name and line number the event occurred in, and the date and time the event occurred. Other information may be printed depending on the options specified and the type of event.

Events may be grouped by event class, or by test script; a summary count of events per group will usually be printed at the bottom of the group output. Output may be formatted into a summary-like listing (FLAT), or into a call-tree like listing (TREED). The VERBOSE option will include more detail for each event printed.

The report text will be output into the TLG log file, but can be redirected into a separate file by using the output redirection syntax, > <rpt_file>.

Note: If TST_REPORTON is set to a non-FALSE value, an implicit REPORT command happens at the end of every test run.

2.64.3 Options

BYCLASS (default)

This option groups reported results by event class; that is, for each of the event class names in the TST_REPORTON variable (results, succresults, ldiffs, warns, etc.), all events of that class will be printed in one group in chronological order, then the group will reset and the next event class will be printed, etc.

This option is useful for summaries as it organizes like events together, where the user can find all failures in one place, all warnings, etc. The reporting of subcommand events ("subcmds") are not supported with this option.

BYSCRIPT

This option groups reported results by test script; that is, for each test script executed, all events whose class names are listed in TST_REPORTON and which occurred in that test script will be printed in chronological order. Then the next script name will be printed, and all its events, etc.

This option is useful for debugging as it presents a call-tree type of listing, showing where an event actually occurred in the test run execution, together with a context of other scripts and their events that ran before and after the event in question.

FLAT (default)

This option prints events in a summary listing type fashion, no indentation.

TREED

This option will print events in a call-tree type fashion, including indentation to denote the current "script call" level (i.e., subscripts invoked with RUN will be indented, as will their subscripts from them, etc.) Each test script invoked with RUN will be printed, even if no events occurred during their execution, to provide the complete call-tree picture and proper context for printed events.

To aid in finding events among the clutter of script names and events, markers will be printed in the left margin:

=== Marks the start of a new test script's execution

--> Marks an event, such as a warning

:-) (happy face) Marks a "good" or success event, such as a RESULT SUCC, or LDIFF suc

X-((unhappy face) Marks a "bad" event, such as a RESULT FAIL / ERR, or LDIFF dif / err

=@ Marks the start of a subcmd event

VERBOSE

This option will print more detailed information about each event, such as RESULT & warning descriptions, RESULT unit name, LDIFF reference file name, TLG line # the event was logged in / near, and OTRM database lookup descriptions, etc.

2.64.4 Availability

New Ora*Tst built-in command in V3.0.

2.64.5 Prerequisites

None.

2.64.6 See Also

result, resultclear Ora*Tst commands.

2.64.7 Known Bugs

None.

2.65 RESULT

2.65.1 Synopsis

```
result [<unitnm>] SUCC|FAIL|ERR|WARN|FLAG [<description>]
```

2.65.2 Description

Creates a Result event for reporting, to manually flag a test unit passing (SUCC), failing (FAIL), or containing an error (ERR.) The result code will be noted in the TLG file. Other result codes, mainly for informational purposes and debugging, include a Warning code (WARN), and a Flag code (FLAG- to denote that the event should be investigated further, for example.)

A test unit name <unitnm> and/or a description of the result <description> can be optionally specified, which will be stored and associated with the result for reporting later with the REPORT command, or stored to the database. Note that if specified, the unit name must be the first argument, and the description argument must be the last argument. Note also that the description is a single argument, so if the string contains spaces, it must be single-quoted.

2.65.3 Options

None.

2.65.4 Availability

New Ora*Tst built-in command in V3.0. FLAG and WARN codes added in V3.0.

2.65.5 Prerequisites

The appropriate RESULT event class name- results (all codes), badresults (FAIL/ERR codes), succresults (SUCC codes), or miscresults (WARN/FLAG codes)- should _not_ be listed in the TST_EVENT_SKIP customization variable for the event to be collected by the reporting system.

2.65.6 See Also

report, resultclear Ora*Tst commands.

2.65.7 Known Bugs

None.

2.66 RESULTCLEAR

2.66.1 Synopsis

```
RESULTCLEAR [ALL|SCRIPT]
```

2.66.2 Description

Clears result event data from internal storage, for either the entire test run so far, or only the current test script (TSC file.)

Currently run statistics are not cleared.

2.66.3 Options

ALL (default)

This option will clear result data for the entire test run.

SCRIPT

This option will clear result data only from the current script. It is currently not recursive; i.e., if child scripts have been called from the current script, they will not have their results cleared.

2.66.4 Availability

New Ora*Tst built-in command in V3.0.

2.66.5 Prerequisites

None.

2.66.6 See Also

For more detailed information, see the section on the Ora*Tst Reporting System in the Ora*Tst Installation and User's Guide.

result, report Ora*Tst commands.

2.66.7 Known Bugs

None.

2.67 RETURN

2.67.1 Synopsis

```
return [<return_value>]
```

2.67.2 Description

The RETURN command forces a return from the currently executing script, macro or included file, to the parent script. Upon return, the built-in TST_RETURN variable is set. If no <return_value> is specified, the TST_RETURN value is set to "0" (the default). Otherwise, the TST_RETURN value is set to <return_value>. If the RETURN command is encountered in the main driving script, execution is terminated and control returns to the Operating System.

There is no requirement or even the suggestion that the RETURN command be used at the end of each test script, macro or included file. Instead, the recommendation is to let the logic "run off the end" of the file. The end-of-file is in effect a valid "return". However, the RETURN command may be usefel to force the return from a script prior to reaching the end-of-file. This might be helpful when a minor, recoverable error or specific condition is encountered.

2.67.3 Options

```
<return_value>
```

The <return_value> is an optional literal value (which means it can be a de-referenced Oratst variable). The TST_RETURN variable is set to this value on return. The value can be a number or a string. If <return_value> contains spaces it must be quoted with single quotes.

2.67.4 Notes

The RETURN command was created in response to the often incorrect use of the EXIT command in recoverable and/or non-fatal conditions. That is, the EXIT command is intended for use when serious errors are encountered and script termination is imminent and/or appropriate. The observation of existing test code indicates the EXIT command being used inappropriately in many cases.

2.67.5 Availability

Ora*Tst built-in command since V3.5.5.4.0.

2.67.6 Prerequisites

The RETURN command cannot be used within a mult-line ECHO command.

2.67.7 See Also

exit Ora*Tst command.

2.67.8 Known Bugs

None.

2.68 RMDIR

2.68.1 Synopsis

```
rmdir [FORCE] <area>
```

2.68.2 Description

Removes the Ora*Tst AREA specification and physical directory associated with the <area>. Only empty directories are removed unless the FORCE flag is specified. <area> must refer to a single directory. That is, <area> cannot be defined with more than one directory.

2.68.3 Options

FORCE

By default, this flag is not set so that only empty directories are removed. If this flag is specified, the directory and everything - files and directories - under it are removed.

2.68.4 Notes

After running RMDIR, if you reference the Ora*Tst area name <area>, the behavior is undefined.

2.68.5 Availability

Ora*Tst built-in command since V3.1.0

2.68.6 Prerequisites

The area specification <area> must exist and refer to an existing single directory.

2.68.7 See Also

For more detailed information on the Ora*Tst areas concept, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

cleanout, finddir and mkdir Ora*Tst commands.

2.68.8 Known Bugs

None.

2.69 RMFILE

2.69.1 Synopsis

```
rmfile [<area>:]<file>.<type> [[<area>:]<file>.<type> ...]
       [nowarn]
```

2.69.2 Description

Removes the file <file> of type <type> in area <area>. The default area <area> is the working area T_WORK.

A maximum of five (5) file arguments can be specified. If an error is encountered for any of the files specified, the return status of the command is that of the last error encountered. Thus, rmfile may return a failing status even if some files were removed successfully. Specifying multiple file arguments does not significantly improve performance and is not compatible with versions prior to 3.5.5.2.0.

2.69.3 Options

`nowarn`

This optional parameter indicates to not issue "TST*WRN-4: cannot find file" when the <file> does not exist. Even though the warning is not issued, the TST_STATUS variable is set to a non-0 value. It is better to use the nowarn option on this command instead of globally suppressing the "TST*WRN-4" message.

2.69.4 Notes

The specification of any portion of a platform specific path component (i.e. dirname/filename) for <file> is not portable and is not supported.

Ora*Tst versions prior to V3.5.3.4.0 can not remove files larger than 2GB.

A file name longer than 128 bytes is truncated without warning.

Windows Specific Enhancements

Because Windows platforms can sometimes run into issues with "File in use" errors, Ora*Tst enhancement 6893658 supports the retry of the remove. The default is to retry once after a one second delay. The defaults can be changed by setting (SET/LET) the following internal Ora*Tst variables:

TST_RMFILE_RETRY_CNT - integer value from 0-9999 indicating the number of retries. Default is 1. 0 = no retries. Values less than 0 are reset to the default. Values greater than 9999 are reset to 9999.

TST_RMFILE_RETRY_TIME - integer value from 0-9999 indicating the time in seconds to wait between retries. Default is 1. Values less than 0 are reset to the default. Values greater than 9999 are reset to 9999.

The retry logic only applies to Windows platforms and is available starting at version 3.5.5.2.0. A message is written to the test log (tlg) for each retry. The retry is attempted only if the remove encounters a "File in use" error. No other errors are retried.

In addition to retrying the remove, an external program can be specified to help determine what process or processes have the file in use. The output of this command is written to the test log (tlg) to help with debugging. See Ora*Tst enhancement 6893658 for additional details on how to enable this feature.

2.69.5 Availability

Ora*Tst built-in command since V1.0.6.

Multiple file arguments allowed starting with V3.5.5.2.0.

Retry of remove for Windows platforms available starting with V3.5.5.2.0.

Nowarn option available starting with V3.5.5.4.0.

2.69.6 Prerequisites

The file <file> of type <type> in area <area> should exist.

For more detailed information on Ora*Tst file specifications, refer to the File Specifications section of the Ora*Tst Installation and User's Guide.

2.69.7 See Also

cpfile, mvfile and findfile Ora*Tst commands.

2.69.8 Known Bugs

None.

2.70 RUN, RUNTEST

2.70.1 Synopsis

```
run [<src_area>:]<scr_name>[.TSC] [NOFORCE|FORCE] [-s <startup>]
        [-c <cleanup>] { [ <local_var1> ] |
        [<local_var2>[=<value2]] ] } ...
```

2.70.2 Description

Creates an additional layer of nesting with regard to test script execution; in particular with regard to local variables. The command line arguments (with the exception of the redirection syntax) are handled exactly as command line arguments to macros are. Command line arguments are essentially set as local variables at the newly created level.

If startup (<startup>) or cleanup (<cleanup>) scripts are specified, they will be run before and after executing the specified script, respectively.

The run Ora*Tst command was designed to allow for imposing a hierarchical structure on the test scripts. Note that this command merely creates a new nested scoping level and starts executing the specified script in the middle of the current script; it does not create another Ora*Tst process (see the fork command.)

The test run continues to be logged in the current TLG file. The filetype of the script <scr_name> is forced to be TSC. If the area <src_area> is not explicitly specified, it defaults to the area T_SOURCE.

The RUN and RUNTEST commands are synonymous.

2.70.3 Options

NOFORCE (default)

By default, the RUN Ora*Tst command will not execute the specified script, if the name <scr name> is currently logged in the TSC file tkbroken in the area T_SOURCE suggesting that the script be skipped.

FORCE

The RUN Ora*Tst command will force execution of the specified script, even though the name <scr name> is currently logged in the TSC file tkbroken in the area T_SOURCE suggesting that the script be skipped.

2.70.4 Notes

A test script name longer than 128 bytes is truncated without warning.

It is possible that the script to run may not execute because of the ORATST_SKIP_TEST environment setting or as the result of subtest hook/intercept logic. See the "Advanced Concepts and Features" section of the Ora*Tst Installation and User's Guide for more information on ORATST_SKIP_TEST and the hook/intercept feature.

2.70.5 Availability

Ora*Tst built-in command since V1.0.6. Startup and cleanup features added in V3.0.1.

2.70.6 Prerequisites

The specified test script <scr name> must be a TSC file which exists in the area specified.

The maximum number of nested (sub)test invocations is 18. This value includes any nesting of macros. For versions prior to V3.5.5.4.0 the limit was 16.

2.70.7 See Also

fork and include Ora*Tst commands.

2.70.8 Known Bugs

None.

2.71 RUNREMOTE

2.71.1 Synopsis

```
runremote <h_remotevar> [<exearea>:]<program>[.EXE|.COM] <arg> ...
```

2.71.2 Description

Invokes an executable with the specified arguments on the remote machine specified by <h_remotevar>. The <h_remotevar> must be a previously defined Ora*Tst variable specifying a remote machine and port number that is running Ora*Tst in server mode. If the area <exearea> is not specified it defaults to T_SYSTEM.

<program> can be a linked program (portable filetype EXE) or an executable OSD script (portable filetype COM). Multiple <arg> can be specified. For portability, the total length of the entire command is limited to 1024 characters. The maximum number of arguments to <program> is 50 (prior versions had different limits. See the Notes and Known Bugs).

2.71.3 Options

None.

2.71.4 Notes

This command is similar to remoterun but uses TCP socket communication instead of Sql*Net. The remoterun command is being phased out in favor of runremote.

The practical limit to the number of arguments to <program> was about 45 prior to V3.6.0.0.0. At version 3.6.0.2.0 the limit is enforced at a maximum of 50. See the Known Bugs below.

2.71.5 Availability

Ora*Tst built-in command in V3.3.0.

2.71.6 Prerequisites

The remotehost must have Ora*Tst running is server mode (oratst -server <portnumber>). The <h_remotevar> must be a previously defined Ora*Tst variable specifying the remote host in the format <hostname:portnumber>. The remote host area <exearea> must exist and be defined to the remotely running Ora*Tst server.

2.71.7 See Also

The Ora*Tst Installation and User's guide for details on the Ora*Tst -server command.

getfile, putfile, remotefindfile Ora*Tst commands.

2.71.8 Known Bugs

Bug 14628959: The number of arguments to <program> is incorrectly limited to 5 at V3.6.0.0.0 & 3.6.0.1.0. This is fixed at V3.6.0.2.0. Versions prior to this have a limit of about 45.

Bug 7566403: Executable file suffix not automatically resolved on Windows. Fixed at V3.5.5.3.0.

2.72 RUNUTL

2.72.1 Synopsis

```
runutl [ASIS] [SETRESULT] [<exe_area>:]<exe_name>[.EXE|.COM]
         { [ <exe cmdline arg> ] } ...
         [< <inarea>:<inname>.<intype>]
         [>[>>] <outarea>:<outname>.<outtype>]
         [>>[>>] <errarea>:<errname>.<errtype>]
```

2.72.2 Description

Invokes an external executable with the specified command line arguments. The executable may either be a compiled and, if applicable, linked program (portable filetype EXE) or an executable OSD script (portable filetype COM). If the ASIS option is used, the command line arguments and filenames (including redirection) are left as-is, i.e. are not lowercased. (Note that this is not portable; it is meant as a work-around only. See Options below.) For more information on command line arguments and how they are passed to the executable, refer to the Command Line Arguments section of the Ora*Tst Installation and User's Guide.

This command is the same as EXECUTE except:

- <exe_area> defaults to T_SYSTEM
- No TST*WRN-40 message is issued for non-zero return codes

If the <exe area> is not explicitly specified, it defaults to T_SYSTEM.

The current working directory for the external command is T_WORK.

<inarea>, <outarea> and <errarea> default to T_WORK if not specified.

By default, standard output will be redirected to the file T_WORK:<exe_name>.log. Redirected output can overwrite (>) or append (>>>) (note the difference between common UNIX redirection) to the specified file.

By default, standard error is likely to print to the terminal (there is no default redirection to a file for standard error). Redirected error can overwrite (>>) or append (>>>>) (note the difference between common UNIX redirection) to the specified file. The default behavior for stderr output can be changed by specifying the -e parameter on Ora*Tst which causes stderr to be written to <testname>.ter. See section 2.7 of the Ora*Tst Installation and User's Guide.

Note that I/O redirection (as well as the concept of standard output and input) are not always portable and may not be available on all ports of Ora*Tst. I/O redirection during the execution of an OSD script is supported in the sense that, if an executable is invoked from the script, I/O will be redirected for that executable in the way described above.

The purpose of the runutl Ora*Tst command is the invocation of an executable which either has been previously compiled via the compile Ora*Tst command or pre-exists in the OS environment. Since it is meant to run an executable, it will only accept the COM and EXE filetypes.

The return code from the executable is **not** checked, so no TST*WRN-40 messages is written to the tlg for non-zero return codes. Consider using the EXECUTE command if the return code from the exernal command is important.

The OSD return code is stored in the variable TST_EXE_STATUS. This variable is changed by each invocation of EXECUTE and RUNUTL. Do not compare the return code with absolute numbers since different operating systems have different meanings for success and failure.

On UNIX, the OSD return code of 0 is returned, if and only if the executable exited successfully.

On VMS, the OSD return code of 1 is returned, if and only if the executable exited successfully. As of Ora*Tst Version 3.5.4.0.0, the value of TST_EXE_STATUS is set to 0 on success. See bug 3843511 for details.

If you need to pass a filename to the external command, use the OSDFILENAME command (V2.1) to convert the file specification to an OSD filename first, otherwise your script may not be portable, and you may not be able to specify areas etc. as the external OSD environment does not understand Ora*Tst portable file specifications.

Starting from V3.5.4.0.0, changes have been implemented to allow the RUNUTL command to return a string value that is stored in a predefined Oratst internal variable, TST_EXE_RESULT. This variable is changed by each invocation of EXECUTE or RUNUTL. The resulting string is obtained by reading the standard output file of the command that is run by RUNUTL. This functionality is enabled by specifying the SETRESULT flag (available 3.5.4.1.0) or by setting the internal Oratst variable TST_GET_EXE_RESULT to a true value (the default value is false).

2.72.3 Options

ASIS

If specified, Ora*Tst will leave all command line options command as-is, i.e. it will not lowercase them. This includes filenames (redirection too.) To retain the case of the filenames, but lowercase the arguments as normal, use the -i or -u Ora*Tst command line options (see Invoking Ora*Tst in the Installation and User's Guide).

Note that using this option, since it is case-sensitive, is not portable and will likely break on other platforms. It is meant as a workaround solution feature only. Please do not use this feature unless absolutely necessary in base development test scripts, and as with any non-portable feature, put it in an OSD macro. You should be putting external executable invocations in an OSD macro in any case (see below.)

SETRESULT

If specified, Ora*Tst stores the output of the external command into the Ora*Tst internal variable TST_EXE_RESULT. This provides functionality similar to the UNIX shell command substitution with back quotes (i.e. result=`cmd`).

Starting with Version 3.5.4.0.0, changes were implemented to allow the RUNUTL command to return a string value that is stored in a predefined Oratst internal global variable, TST_EXE_RESULT. This variable is changed by each invocation of EXECUTE or RUNUTL Thus, the value should be saved into a separate Ora*Tst variable immediately after the command. The resulting string is obtained by reading the Ora*Tst standard output file of the RUNUTL command. This functionality was originally enabled by setting the internal Oratst variable TST_GET_EXE_RESULT to a true value (the default value is false). The same action can now be specified with the SETRESULT flag. The SETRESULT flag is the preferred method for returning results into TST_EXE_RESULT.

The Ora*Tst standard output file is the default file T_WORK:<exename>.log or the the output file specified by <outname>.<outtype>.

Additional TST_EXE_RESULT details:

When SETRESULT is specified, the standard output of the RUNUTL command is processed and the TST_EXE_RESULT variable set accordingly. Each execution of this command starts by setting TST_EXE_RESULT to null. In most cases, TST_EXE_RESULT is set to null when errors are encountered. However, there is the possibility that some error conditions may leave garbage in the TST_EXE_RESULT variable. If this possibility presents a problem for your script consider adding logic to validate the TST_EXE_RESULT value.

When SETRESULT is specified (or TST_GET_EXE_RESULT is set to a true value), additional processing is required to read the Ora*Tst standard output file and set the TST_EXE_RESULT variable. It is suggested to use this flag only for those invocations of RUNUTL that require string result processing.

A maximum of 2048 bytes are read from the Ora*Tst standard output file and assigned to the TST_EXE_RESULT variable. Before the assignment to TST_EXE_RESULT:

* Newline characters (CR/LF) are converted to spaces.
* All non-printable characters are converted to spaces.
* Leading and trailing spaces are removed.
* Multiple spaces are trimmed to a single space.

By default, the RUNUTL command creates a standard output file containing the stdout of the external command (T_WORK:<exename>.log). Thus, it is not necessary to explicitly specify a standard output file for TST_EXE_RESULT to be set. The standard output is processed after the external command specified by RUNUTL completes. Thus, the TST_EXE_RESULT variable is assigned what exists in the standard output file after any appends or redirection from stderr.

2.72.4 Notes

The following characters should always be quoted in an argument list to prevent Ora*Tst interpretation:

(cross hatch/number sign) - must be quoted or it comments out any remaining characters.

< (less than sign/left angle bracket) - must be quoted or it is treated as a file redirection.

> (greater than sign/right angle bracket) - must be quoted or it is treated as a file redirection (this includes >>, >>>, >>>>).

^ (circumflex/caret) - must be quoted unless dereferencing a variable.

See the Ora*Tst ECHO command:Known Bugs for details and examples on quoting.

On Windows a command argument ending with a backslash (\) needs to be quoted or an additional backslash needs to be added. This is a common requirement when invoking Java and -Dsep is specified with a backslash. Failure to do so can result in a truncated command line. For example, specify -Dsep=\ \ instead of -Dsep=\

On Windows each command line argument is enclosed in double quotes before it is passed to the operating system for execution. It is possible to turn off this behavior by setting the TST_NOQUOTE_WIN_CMD variable to a TRUE value. The default is FALSE. Only use this if absolutely necessary and to turn the switch off after execution. This switch is available starting at V3.5.5.3.0. See bug 7686482 for more details.

Keep in mind that when executing scripts such as Perl or Java (i.e execute T_SYSTEM:perl mycmd.pl) the default output file will match the <exe_name> (i.e. perl or java) and not the script name (which is an argument). Thus, it is common to end up with an output file of perl.log, which is then overwritten on subsequent Perl commands using the same format.

2.72.5 Availability

The command was an Ora*Tst built-in command in V1.0.6. The ability to redirect standard error was added in V2.0; the ability to execute OSD scripts, if applicable, was added in V2.0.

ASIS option added in V2.1.

The return of a string value into TST_EXE_RESULT was added in V3.5.4.0.0.

SETRESULT option added v3.5.4.1.0.

2.72.6 Prerequisites

It is strongly recommended that the invocation of any executable be encapsulated in an OSD Ora*Tst macro, as external executable names, options etc. may vary according to platform.

The specified executable <exe_name> must exist.

2.72.7 See Also

For more detailed information on OSD macros, refer to the Ora*Tst OSD Macros section of the Ora*Tst Installation and User's Guide.

execute, osdfilename Ora*Tst commands.

2.72.8 Example usages

```
RUNUTL T_WORK:myprog abc 123 -recur < T_SOURCE:input
RUNUTL ASIS PATH:ls -F > T_WORK:listing.dat
RUNUTL ASIS PATH:grep '^xyz' ^osdfname^
```

2.72.9 Known Bugs

Ora*Tst bugs #3975903 and #2621432: The Ora*Tst parsing routine adds spaces around specific characters which can affect the output of the RUNUTLcommand. The solution is to enclose these characters in single quotes. See the Ora*Tst ECHO command:Known Bugs for additional details and the list of special characters.

See bug 7686482 regarding double quotes added to arguments for Windows.

2.73 SAVE (macro)

2.73.1 Synopsis

```
save <srcfile>[.<srctype>]
```

2.73.2 Description

Copies a file <srcfile> of type <srctype> in the working area to a file in the area T_HIST. Note that there is no “default” for the area; it specifically copies from T_WORK.

The defaults are the same as for the CPFILE Ora*Tst command. The protection mode of the target file and the source file are the same, with the exception that the target file will be writable by the owner.

2.73.3 Options

None.

2.73.4 Availability

Ora*Tst built-in command since V1.0.6. There is an automatic default macro definition.

2.73.5 Prerequisites

None.

2.73.6 See Also

For more detailed information on Ora*Tst macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User’s Guide.

For more information on the implementation of the portable Ora*Tst default macro, refer to the file ctm03.tsc.

cpfile Ora*Tst command.

2.73.7 Known Bugs

None.

2.74 SET

2.74.1 Synopsis

```
set <var> [<value>]
```

2.74.2 Description

Creates (if necessary) and sets the global Ora*Tst variable <var> to the value <value>. If <value> is not specified, the variable <var> defaults to TRUE.

2.74.3 Options

None.

2.74.4 Notes

It is possible to have a Global (set) variable with the same name as a local (let) variable. In this situation, the local (let) variable takes precedence. Use the getglobal command to obtain the value of the global variable of the same name. It is best to avoid name conflicts between global and local variables when possible.

Global (set) variables are visible in all subtests or macro calls and parent scripts.

When specifying a numeric value many Ora*Tst commands such as INCR, DECR and IF limit the range to a signed 32-bit integer.

2.74.5 Availability

Ora*Tst built-in command since V1.0.6.

Changes from previous version:

`SET var' without a value sets <var> to TRUE.

2.74.6 Prerequisites

None.

2.74.7 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

unset, let, unlet, getlocal, getglobal Ora*Tst commands.

2.74.8 Known Bugs

None.

2.75 SETAREA

2.75.1 Synopsis

```
setarea [<area>:][<name>][.<type>] <new_area> <local_var>
```

2.75.2 Description

Replaces the <area> portion of the Ora*Tst file specification with the new area string <new_area>, placing the result in local variable <local_var>. If <area> is not specified, the value of <new_area> is simply prepended to the file specification.

Do not include a trailing semicolon (:) in the <new_area> string.

2.75.3 Options

None.

2.75.4 Notes

This command does not define or allocate a new AREA. It is used to modify the string representing the Ora*Tst file specification.

2.75.5 Availability

New Ora*Tst built-in command in V3.0.

2.75.6 Prerequisites

None.

2.75.7 See Also

For more detailed information on Ora*Tst file specifications and variables, refer to the File Specifications and Ora*Tst Variables sections of the Ora*Tst Installation and User's Guide.

getarea, getfiletype, getbasename, setdefarea, setdeftype, setfiletype and mkdir Ora*Tst commands.

2.75.8 Known Bugs

None.

2.76 SETDEFAREA

2.76.1 Synopsis

```
setdefarea [<area>:][<name>][.<type>] <default_area> <local_var>
```

2.76.2 Description

Sets the area definition of an Ora*Tst file specification to a default value if necessary. Interprets the first argument [<area>:]<name[.<type>] as an Ora*Tst file specification. If area <area> is specified, then the local variable <local_var> is set to the unaltered file specification. If the area <area> is omitted, then the area <default_area> provides the appropriate default and the local variable <local_var> is set to the new Ora*Tst file specification with the <default_area> prepended to the file specification.

2.76.3 Options

None.

2.76.4 Notes

This command does not define or allocate a new AREA. It is used to modify the string representing the Ora*Tst file specification.

2.76.5 Availability

New Ora*Tst built-in command in V3.0.

2.76.6 Prerequisites

None.

2.76.7 See Also

For more detailed information on Ora*Tst file specifications and variables, refer to the File Specifications and Ora*Tst Variables sections of the Ora*Tst Installation and User's Guide.

getarea, getfiletype, getbasename, setarea, setfiletype, setdeftype, mkdir and osdfilename Ora*Tst commands.

2.76.8 Known Bugs

None.

2.77 SETDEFTYPE

2.77.1 Synopsis

```
setdeftype [<area>:]<name>[.<type>] <default_type> <local_var>
```

2.77.2 Description

Sets the file type definition of an Ora*Tst file specification to a default value if necessary. Interprets the first argument [<area>:]<name[.<type>] as an Ora*Tst file specification. If a filetype <type> is specified, then the local variable <local_var> is set to the unaltered file specification. If the filetype <type> is omitted, then the type <default_type> provides the appropriate default and the local variable <local_var> is set to a new Ora*Tst file specification with the <default_type> for the filetype.

If an area is not specified, it does not default to anything; i.e., it does not add on T_WORK: to the file specification.

2.77.3 Options

None.

2.77.4 Notes

This command does not define or allocate a new AREA. It is used to modify the string representing the Ora*Tst file specification.

2.77.5 Availability

Ora*Tst built-in command since V1.0.6.

2.77.6 Prerequisites

None.

2.77.7 See Also

For more detailed information on Ora*Tst file specifications and variables, refer to the File Specifications and Ora*Tst Variables sections of the Ora*Tst Installation and User's Guide.

getfiletype, getbasename, setfiletype, and osdfilename Ora*Tst commands.

2.77.8 Known Bugs

None.

2.78 SETFILETYPE

2.78.1 Synopsis

```
setfiletype [<area>:]<name>[.<type>] <new_type> <local_var>
```

2.78.2 Description

Replaces the <type> portion of the Ora*Tst file specification with the value defined by <new_type>, placing the result in local variable <local_var>. If <type> is not specified, the value of <new_type> is appended to the file specification.

If an area <area> is not specified, it does not default to anything; i.e., it does not add on T_WORK: to the file specification.

Do not include a leading period (.) in the <new_type> string.

2.78.3 Options

None.

2.78.4 Notes

This command does not define or allocate a new AREA. It is used to modify the string representing the Ora*Tst file specification.

2.78.5 Availability

Ora*Tst built-in command since V1.0.6.

2.78.6 Prerequisites

None.

2.78.7 See Also

For more detailed information on Ora*Tst file specifications and variables, refer to the File Specifications and Ora*Tst Variables sections of the Ora*Tst Installation and User's Guide.

getfiletype, getbasename, setdeftype Ora*Tst commands.

2.78.8 Known Bugs

None.

2.79 SKIPTEST

2.79.1 Synopsis

```
skiptest
```

2.79.2 Description

Works in conjunction with the subtest Hook/Intercept logic and indicates that execution of the target RUN/RUNTEST script is to be skipped.

2.79.3 Options

None.

2.79.4 Notes

Execution continues after the SKIPTEST command. It does not terminate or force a return of the currently executing intercept script.

See the "Advanced Concepts and Features" section in the Ora*Tst Installation and User's Guide for more information.

2.79.5 Availability

Ora*Tst built-in command since V3.6.0.0.0.

2.79.6 Prerequisites

Can only be issued from the "subtest start" script identified by the ORATST_SUBTEST_START environment variable or the TST_SUBTEST_START variable. This command can not be issued from any other test script, macro or included file.

This command can not be coded within a multi-line ECHO command.

2.79.7 Known Bugs

None.

2.80 SLEEP

2.80.1 Synopsis

```
sleep <seconds>
```

2.80.2 Description

Puts the calling Ora*Tst process to sleep for the specified number of seconds; useful as a delay feature when forking other Ora*Tst processes, etc. Used internally in Ora*Tst test suites to let FORKed processes finish.

Granularity is only to integer seconds; e.g., “sleep 3.2” will be evaluated as “sleep 3”.

2.80.3 Options

None.

2.80.4 Availability

Ora*Tst built-in command since V2.1.

2.80.5 Prerequisites

The value of <seconds> should be in the range of 1to max signed 32-bit integer.

2.80.6 See Also

fork, isactive, wait, stop Ora*Tst commands.

2.80.7 Known Bugs

None.

2.81 SORTFILE

2.81.1 Synopsis

```
sortfile [<area>:]<srcfile>.<type> [<newarea>:]<newfile>.<newtype>
```

2.81.2 Description

Sorts the contents of <srcfile> between the boundaries marked with YODA_BEGIN_SORT_BLOCK and YODA_END_SORT_BLOCK and writes the result to <newfile>. If either area is not specified it defaults to T_WORK. The Yoda "markers" are not written to the <newfile>.

2.81.3 Options

None.

2.81.4 Availability

Ora*Tst built-in command since V3.2.5.

2.81.5 Prerequisites

None.

2.81.6 See Also

ldiff Ora*Tst command.

2.81.7 Known Bugs

None.

2.82 SQL (macro)

2.82.1 Synopsis

```
SQL [<area>:]<sql_file>[.<type>] [connect]
        [[<area>:]<logfile>[.<type>]]
```

2.82.2 Description

Executes the specified SQL file, storing the output in a log file. The default area for the SQL file is T_SOURCE and its default type is .SQL. Connect is an Ora*Tst variable containing the connect string and defaults to '/nolog' if not specified. If the log file is not provided as a parameter, the output is stored in a file called 'spoolfile.log' in the T_WORK area. The log file type (extension) is ignored and always defaults to LOG. Explicit input redirection is ignored.

2.82.3 Options

None.

2.82.4 Notes

A default version of this macro is provided by Ora*Tst. It is common to find this macro overridden in the RDBMS Server environment. Thus, the options and behavior of the overriding macro may differ substantially from that described here.

2.82.5 Availability

Ora*Tst macro since V1.0.6. This macro has been made more flexible as for Ora*Tst V3.1. It accepts a log file as optional parameter and sets defaults for the sql file area and type. It is backward compatible. This macro is provided as a template, and it is not directly supported by the Ora*Tst Development group.

2.82.6 Prerequisites

None.

2.82.7 See Also

For more detailed information on macros, refer to the Ora*Tst Macros section of the Ora*Tst Installation and User's Guide. The Ora*Tst FAQ doc has additional information on how to determine the source file that defines a macro. For information specifically regarding the SQL macro, see the file ctm02.tsc.

2.82.8 Known Bugs

None.

2.83 STOP

2.83.1 Synopsis

```
stop ^<var>^
```

2.83.2 Description

Stops a (child) Ora*Tst process previously created by the FORK Ora*Tst command. The actual value of the respective process handle (stored in an Ora*Tst variable <var>, such as ENTRY) needs to be specified as first argument.

The specification of the stop (kill) signal is configurable for UNIX platforms. Bug 3108294 allows for a UNIX SIGTERM signal to be used on the kill (the default is SIGKILL). The Ora*Tst variable TST_STOP_KILLTYPE can be set to KILL or TERM (not case sensitive, default is KILL) to specify the UNIX signal to use. See the Bug text for additional details.

2.83.3 Options

None.

2.83.4 Notes

The STOP command may not be effective at stopping child processes of the target process on all platforms. This is known to be an issue on Windows.

If the specified child process has already finished, a TST*WRN-38 warning message is issued indicating the child could not be stopped. Starting at V3.6.0.2.0, no warning message is issued when the child has already finished and instead a "STOP Entry nnnn is not active" message is written to the .tlg.

2.83.5 Availability

Ora*Tst built-in command since V1.0.6. The configurable UNIX kill signal was available in V3.5.3.2.0.

2.83.6 Prerequisites

The (child) Ora*Tst process must have been created and started and still be running. The variable <var> must be set to a valid, port-specific process handle, i.e. a value of the global Ora*Tst variable ENTRY upon successful completion of a FORK Ora*Tst command.

2.83.7 See Also

fork, isactive, wait, and sleep Ora*Tst commands.

2.83.8 Known Bugs

None.

2.84 STRCAT

2.84.1 Synopsis

```
strcat <str_var1> <str_var2> <target_var> [log|nolog]
```

2.84.2 Description

Concatenates the string values defined to variable <str_var1> and variable <str_var2> together and places the result in local variable <target_var>.

2.84.3 Options

log|nolog

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log.

2.84.4 Notes

Ora*Tst versions prior to 3.5.4.1.0 were limited to 1024 bytes and no warning was given when the string was truncated. Version 3.5.4.1.0 to 3.5.5.2.0 allow a maximum of 4096 bytes to be copied and a warning is issued when the target is truncated. Starting at version 3.5.5.3.0 there is no length limit.

2.84.5 Availability

Ora*Tst built-in command since V3.3.0.

The log / nolog parameter is available at V3.5.5.3.0.

2.84.6 Prerequisites

None.

2.84.7 See Also

findstr, findrstr, substr, subword, varlen, word, words Ora*Tst commands.

2.84.8 Known Bugs

None.

2.85 STRLOWER

2.85.1 Synopsis

```
STRLOWER <source_var> [<target_var>] [log|nolog]
```

2.85.2 Description

Sets the string value contained in <source_var> to lower case. By default, <source_var> is updated in place. If <target_var> is specified, the result is placed in <target_var> and <source_var> remains unchanged. If <target_var> is specified but does not exist it is created as a local variable.

2.85.3 Options

`target_var`

If <target_var> is specified it is updated with the lower case result of <source_var> and <source_var> is unchanged. If <target_var> does not exist it is created as a local variable.

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.85.4 Notes

Uses the ORACORE STring to LOwer case routine (lstlo). Does not use NLS.

2.85.5 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.85.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

strupper, findstr, findrstr, strcat, substr, subword, word, varlen Ora*Tst commands.

2.85.7 Known Bugs

None.

2.86 STRUPPER

2.86.1 Synopsis

```
STRUPPER <source_var> [<target_var>] [log|nolog]
```

2.86.2 Description

Sets the string value contained in <source_var> to uppercase. By default, <source_var> is updated in place. If <target_var> is specified, the result is placed in <target_var> and <source_var> remains unchanged. If <target_var> is specified but does not exist it is created as a local variable.

2.86.3 Options

`target_var`

If <target_var> is specified it is updated with the upper case result of <source_var> and <source_var> is unchanged. If <target_var> does not exist it is created as a local variable.

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.86.4 Notes

Uses the ORACORE STring to UPpercase routine (lstup). Does not use NLS.

2.86.5 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.86.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

strlower, findstr, findrstr, strcat, substr, subword, word, varlen Ora*Tst commands.

2.86.7 Known Bugs

None.

2.87 STRVERIFY

2.87.1 Synopsis

```
STRVERIFY <source_var> <reference_var>|<reference_literal>
        <result_var> [MATCH|NOMATCH] [<start_pos>] [log|nolog]
```

2.87.2 Description

Verifies the contents of variable <source_var> with the contents of variable <reference_var> or literal <reference_literal>. Determines the position in <source_var> of the first character that either matches (match) or does not match (nomatch) the characters in <reference_var> or <reference_literal> and returns the result in <result_var>. If <result_var> does not exist it is created as a local variable.

The first character in <source_var> is considered position 1. The default is NOMATCH. The default <start_pos> is 1. If <source_var> is a null (empty) string the return value is 0 for either MATCH or NOMATCH. If <reference_var> is a null (empty) string then 0 is returned for MATCH and <start_pos> is returned for NOMATCH.

The <reference_literal> can be one of the following literal strings:

__ALNUM__, __ALPHA__, __CNTRL__, __DIGIT__, __GRAPH__, __LOWER__, __PRINT__, __PUNCT__, __SPACE__, __UPPER__, __XDIGIT__

These literals are synonymous with the following C Language Character classification functions, respectively:

isalnum(), isalpha(), iscntrl(), isdigit(), isgraph(), islower(), isprint(), ispunct(), isspace(), isupper(), isxdigit()

In addition, the following two literals are supported:

__INTNUM__, __HEXNUM__

These allow a leading minus (-) and leading 0x/X for numbers and are otherwise identical to **__DIGIT__** and **__XDIGIT__**, respectively.

2.87.3 Options

MATCH| NOMATCH

MATCH: Returns 0 if none of the characters in <source_var> are in <reference_var>. Otherwise, returns the position of the first character in <source_var> that IS in <reference_var>.

NOMATCH: Returns 0 if all of the characters in <source_var> are in <reference_var>. Otherwise, returns the position of the first character in <source_var> that is NOT in <reference_var>. This is the default.

start_pos

A literal value indicating the starting position in <source_var> to begin verification. The default is 1. The first character of <source_var> is considered position 1.

log|nolog

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.87.4 Notes

There is currently no support for specifying a range of values in <reference_var>. That is, specifying something like [0-9] is NOT supported.

The reference literal checks (and underlying C functions) behave as though the C locale is specified. Reference the appropriate C specification for details on the character classification functions.

All strings should consist of characters in the range of 7-bit ASCII or EBCDIC-1047.

2.87.5 Availability

Ora*Tst built-in command since V3.5.5.3.0.

The <reference_literal> support is available starting with V3.5.5.4.0.

2.87.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

findstr, findrstr, strcat, substr, subword, word, varlen Ora*Tst commands.

2.87.7 Example usage

The examples below show the literal strings for simplicity. The actual strings must be contained in the <source_var> and <reference_var> variables, respectively.

```
            <source_var>  <reference_var>                 <result_var>
STRVERIFY '123'         '0123456789'                          0
STRVERIFY '2X4'         '0123456789'                          2
STRVERIFY 'AB4T'        '0123456789'                          1
STRVERIFY 'AB4T'        '0123456789'  MATCH                   3
STRVERIFY 'AB4T'        '0123456789'  NOMATCH                 1
STRVERIFY '1P3Q4'       '0123456789'            '3'           4
STRVERIFY '567'         '0123456789'  MATCH     '2'           2
STRVERIFY 'ABCDE'       '0123456789'            '3'           3
STRVERIFY 'AB3CD5'      '0123456789'  MATCH     '4'           6
STRVERIFY ''            '0123456789'                          0
```

2.87.8 Known Bugs

None.

2.88 SUBSTR

2.88.1 Synopsis

```
substr <sourcevar> <destvar> <offset> [<len>] [log|nolog]
```

2.88.2 Description

Retrieves a substring from the string defined to variable <sourcevar>, starting from character <offset>, of length <len> (or until the end of the string, if <len> is omitted), and places it into local variable <destvar>. <offset> and <len> are literal values.

The first character of <sourcevar> is considered offset 0.

2.88.3 Options

`len`

The optional length of the substring to copy. This must be a literal value or dereferenced variable. The default is to copy the remaining length of the string. If <len> is greater than the length of <sourcevar> then the value is ignored.

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log.

2.88.4 Notes

Prior to V3.5.5.3.0 the substring was limited to a length of 1024 bytes and truncated without warning. At V3.5.5.3.0 there is no limit on the string length.

If <offset> is outside the range of <sourcevar> or <sourcevar> is null / empty a warning is issued and <destvar> is not created.

2.88.5 Availability

New Ora*Tst built-in command in V3.0.

The log / nolog parameter is available at V3.5.5.3.0.

2.88.6 Prerequisites

The values for <offset> and <len> must be in the range of 0 to max signed 32-bit integer.

2.88.7 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

findstr, findrstr, varlen, strcat, incr, decr, subword, word, words Ora*Tst commands.

2.88.8 Known Bugs

None.

2.89 SUBTRACT

2.89.1 Synopsis

```
SUBTRACT[LOCAL|GLOBAL] <varname> <subtrahend> [<target_var>]
        [log|nolog]
```

2.89.2 Description

Subtracts <subtrahend> from <varname> and stores the result in <varname>. If variable <target_var> is specified it is updated with the result and <varname> is left unchanged. The <varname> must be an Ora*Tst variable and <subtrahend> is a literal value or dereferenced variable. The <varname> value and <subtrahend> must be integers in the range of signed 32bit values.

If <varname> does not exist it is created with an initial value of 0 and a default scope of global.

If <target_var> is specified but does not exist it is created in the same scope as <varname>.

2.89.3 Options

`local | global`

LOCAL: If local is specified, the selection and/or creation of <varname> and optionally <target_var> is local scope only.

GLOBAL: If global is specified, the selection and/or creation of <varname> and optionally <target_var> is global scope only.

If neither local or global is specified, selection of <varname> follows normal scoping rules.

`target_var`

Optional variable name that is updated with the result and <varname> is left unchanged. If it does not exist it is created in the same scope as <varname>.

`log|nolog`

The optional log/nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.89.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.89.5 Prerequisites

The <varname> value and <subtrahend> must be valid signed 32bit integers.

If the result exceeds the value of a signed 32bit integer a "TST*WRN-66" message is issued.

2.89.6 See Also

add, multiply, divide, modulo, incr and decr Ora*Tst commands.

2.89.7 Known Bugs

None.

2.90 SUBWORD

2.90.1 Synopsis

```
SUBWORD <source_var> <start_word> <result_var> [<word_cnt>]
        [<sep_var>] [log|nolog]
```

2.90.2 Description

Retrieves the subword(s) from <source_var> starting from <start_word> for <word_cnt> words and stores the result in <result_var>. The first word is word 1. If <start_word> is greater than the number of words in <source_var> then a null value (empty string) is returned in <result_var>. The <result_var> is created as a local variable if it does not exist.

Words are separated by white space by default or a single character specified in variable <sep_var>. All leading / trailing separator characters are removed when selecting words. The <word_cnt> defaults to the remaining number of words in <source_var> if not specified.

The result is returned with no leading or trailing separator characters but includes all separator characters between words. <start_word> and <word_cnt> are literal values.

2.90.3 Options

`word_cnt`

The <word_cnt> is an optional literal value indicating the number of words to select from <source_var>. The default is to select the remaining words beginning at <start_word>. If <word_cnt> is greater than the number of remaining words in <source_var> then the value is ignored and the remaining words are returned.

`sep_var`

The <sep_var> is an optional Ora*Tst variable containing a single printable character (or space) indicating how to separate words. The default separator is white space.

`log|nolog`

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is nolog.

2.90.4 Availability

Ora*Tst built-in command since V3.5.5.3.0.

2.90.5 Prerequisites

The maximum value for <word_cnt> is max signed 32-bit integer.

2.90.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

findstr, findrstr, strcat, substr, word, varlen Ora*Tst commands.

2.90.7 Known Bugs

None.

2.91 UNLET

2.91.1 Synopsis

```
unlet <var>
```

2.91.2 Description

Resets the local Ora*Tst variable <var> to a default value, the empty string, which evaluates to FALSE.

2.91.3 Options

None.

2.91.4 Notes

This command does not destroy the variable. Once a local variable is created it cannot be removed from the current scoping level's symbol table, only reset to an empty (FALSE) state. When the current level of script nesting is exited, the local variable will be removed. Use the GETGLOBAL command to access a global variable that is hidden in the current scope if necessary.

2.91.5 Availability

Ora*Tst built-in command since V1.0.6.

2.91.6 Prerequisites

None.

2.91.7 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

let, set, unset, getlocal Ora*Tst commands.

2.91.8 Known Bugs

None.

2.92 UNSET

2.92.1 Synopsis

```
unset <var>
```

2.92.2 Description

Resets the global Ora*Tst variable <var> to a default value, the empty string, which evaluates to FALSE.

2.92.3 Options

None.

2.92.4 Notes

Once a global variable is created it cannot be explicitly destroyed, only reset to an empty (FALSE) state.

2.92.5 Availability

Ora*Tst built-in command since V1.0.6.

2.92.6 Prerequisites

None.

2.92.7 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

let, set, unlet, getglobal Ora*Tst commands.

2.92.8 Known Bugs

None.

2.93 UNTIL

2.93.1 Synopsis

```
until <event_name>
```

2.93.2 Description

Awaits a posted synchronization event identified by the name <event_name>. Execution continues if either the event is posted or a time-out occurs. If the event is posted (received), it is destroyed (or in other words consumed) by the executing process and the execution of the until Ora*Tst command succeeds. The command times out after approx. 13 minutes.

2.93.3 Options

None.

2.93.4 Availability

Ora*Tst built-in command since V1.0.6.

2.93.5 Changes from previous version

It is no longer supported to associate a variable with the event and receive a value with the event.

2.93.6 Potential changes in future versions

The implementation of this basic synchronization mechanism is currently being reviewed and subject to change. Please do not rely on port-specific implementation details.

2.93.7 Prerequisites

The event must be posted by another Ora*Tst process within the time limit of the time-out.

2.93.8 See Also

lookfor, post Ora*Tst command.

2.93.9 Known Bugs

Generic bug (#): Ora*Tst currently accepts an optional restrictive specification, such as [<group>:]<event_name>. This is due to an outdated implementation detail and is no longer supported.

2.94 VARLEN

2.94.1 Synopsis

```
varlen <variable> <local_var>
```

2.94.2 Description

Takes the variable name supplied as the first argument <variable>, dereferences it, and stores the length of the resulting string in the local variable <local_var>. If the <variable> is empty, being either unknown or is UNSET/UNLET, the length returned will be 0. If the variable is unknown, Ora*Tst will issue a warning that the variable cannot be found.

This command could for example be used to determine if a variable is set to some value, or empty (different from a TRUE/FALSE test), using the following code snippet:

```
VARLEN testvar len
IF !len
   ECHO testvar is empty, unset or unknown
ENDIF
IF len
   ECHO testvar has been set to something
   ECHO (it may still contain a FALSE string)
ENDIF
```

2.94.3 Options

None.

2.94.4 Availability

Ora*Tst built-in command since V2.1.

2.94.5 Prerequisites

None.

2.94.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

unset, set, unlet, let, findstr, strcat, substr, word, words Ora*Tst commands.

2.94.7 Known Bugs

None.

2.95 WAIT

2.95.1 Synopsis

```
wait [<timeout>]
```

2.95.2 Description

Signals all child Ora*Tst processes previously created by the current test script via the FORK command with the WAIT option to begin execution. The current script (parent) then waits for the child processes to finish. After all the children FORKed with WAIT are finished, any processes FORKed with the KILL option are killed and execution resumes in the parent script.

The parent test script will either wait indefinitely (the default) or wait <timeout> seconds for the FORK with WAIT child processes to complete execution.

If <timeout> is specified and all the child processes FORKed with WAIT have not exited after that interval, then the wait ends and execution resumes after the WAIT statement and the local variable TST_WAIT_TIMEOUT is set to TRUE. If all of the child processes exit before <timeout> seconds, then TST_WAIT_TIMEOUT is set to FALSE. The test script can use the ISACTIVE command to check the state of the child processes, the STOP command to end the child process, or run WAIT again to continue waiting until the children finish execution.

If there are no children FORKed with WAIT, then processes FORKed with KILL are killed immediately and execution resumes in the parent script.

2.95.3 Options

<timeout>

If specified, this is the maximum number of seconds that Ora*Tst will WAIT before resuming execution of the test script. By default, Ora*Tst will wait indefinitely. The value of <timeout> must be greater than 0 and less than max signed 32-bit integer.

2.95.4 Notes

The implementation of the fork / wait mechanism is very system specific.

The appropriate value of <timeout> (if specified) may vary depending on the speed of the platform. It is suggested to use a predefined Ora*Tst variable for <timeout> so that it can be configured by platform (instead of hardcoding a value).

Do not confuse WAIT with the SLEEP command. Only use WAIT when processes have been FORKed with the WAIT option (default) or the KILL option.

If <timeout> is specified and one or more child processes FORKed with WAIT have not finished after that interval, any children FORKed with KILL are *not* killed. A subsequent WAIT command can be specified and when all the children FORKed with WAIT are finished, those FORKed with KILL will be killed.

2.95.5 Availability

Ora*Tst built-in command since v1.0.6. The option for a maximum <timeout> of seconds to wait is available in v3.1.0.

2.95.6 Prerequisites

None.

2.95.7 See Also

fork, stop, isactive, and sleep Ora*Tst commands.

2.95.8 Known Bugs

Bug 8301705: Fork may not wait for WAIT command on Windows. Fixed at V3.5.5.3.0.

2.96 WORD

2.96.1 Synopsis

```
word <source_var> <word_cnt> <result_var> [<sep_var>] [log|nolog]
```

2.96.2 Description

Selects the nth (word_cnt) word from the string defined to variable <source_var> and returns the result in local variable <result_var>. The <result_var> variable is created as a local variable if it does not already exist. The first word is considered word 1. The word_cnt is a literal integer value or a dereferenced Ora*Tst variable containing the number of the word to be selected and must be greater than 0. If word_cnt is larger than the number of words in <source_var> the value returned in <result_var> is null. The default separator is white space. All leading / trailing separator characters and multiple occurrences of separator characters are removed before selecting words.

2.96.3 Options

sep_var

The <sep_var> is an optional Ora*Tst variable containing a single printable character (or space) indicating how to separate words. The default separator is white space.

log|nolog

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log.

2.96.4 Notes

Prior to V3.5.5.3.0 if word_cnt was greater than the number of words in <source_var> a TST*WRN-23 message was issued. Starting with V3.5.5.3.0 no message is issued and <result_var> is set to a null value.

Prior to V3.5.5.3.0 the log entry of the command execution included the text for the word selected. Starting with V3.5.5.3.0 the actual text of the word is no longer written to the log.

2.96.5 Availability

Ora*Tst built-in command since V3.5.4.0.0.

The log / nolog parameter is available at V3.5.5.3.0.

2.96.6 Prerequisites

The value of word_cnt must be within the range of 1 to max signed 32-bit integer.

2.96.7 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

findstr, findrstr, strcat, substr, subword, words, varlen Ora*Tst commands.

2.96.8 Known Bugs

None.

2.97 WORDS

2.97.1 Synopsis

```
words <source_var> <word_cnt> [<sep_var>] [log|nolog]
```

2.97.2 Description

Determines the number of words in the string defined to variable <source_var> and returns the result in local variable <word_cnt>. The <word_cnt> variable is created as a local variable if it does not already exist. The first word is considered word 1. The default separator is white space. All leading / trailing separator characters and multiple occurrences of separator characters are ignored when determining words.

2.97.3 Options

sep_var

The <sep_var> is an optional Ora*Tst variable containing a single printable character (or space) indicating how to separate words. The default separator is white space.

log|nolog

The optional log / nolog parameter indicates whether the command execution is logged in the Ora*Tst log (.tlg) file. The default is log.

2.97.4 Availability

Ora*Tst built-in command since V3.5.4.0.0.

The log / nolog parameter is available at V3.5.5.3.0.

2.97.5 Prerequisites

The maximum value returned for <word_cnt> is max signed 32-bit integer.

2.97.6 See Also

For more detailed information on Ora*Tst variables, refer to the Ora*Tst Variables section of the Ora*Tst Installation and User's Guide.

findstr, findrstr, strcat, substr, subword, word, varlen Ora*Tst commands.

2.97.7 Known Bugs

None.