LDRA tool suite Command Line Mode

C/C++ 9.7.5 (Windows (10/8/7)) Manual Revision 6 - November 2018.

Table of Contents

929TBED1122118

Command Line Mode Overview

LDRA tool suite applications can be ran from the command line without invoking any GUI.

There are many applications that can be ran on the command line.

- **Contestbed**, the command line version of LDRA Testbed, used for Static and Dynamic analysis.
- **Contbrun**, the command line version of LDRA TBrun, used for Unit Testing and Regression Analysis.
- **TBini**, used to set or alter INI flag settings from the command line.
- **TBmanager**, Requirements and Traceability, can also be used on the command line.
- **Contbbuildimport**, the command line version of TBbuildimport, used to import files and configuration settings from build commands.
- **GLH_support**, used for extracting Configuration and Analysis data files from the GLH.

Each application uses unique arguments and command line qualifiers as described in the following sections.

Some analysis phases and features are license dependent.

Information on installing *LDRA tool suite* on the command line can be found in the *Installation Guide*.

Contestbed

Contestbed is the console version of *LDRA Testbed, t*he main Analysis engine. This can be used on the command line to run the following phases:

Main Static Analysis
Complexity Analysis
Static Data Flow Analysis
Cross Reference
Information Flow Analysis
Data Object Analysis
MC/DC Test Case Planner
Generate Instrumented Program(s)
Enable Exact Semantic Analysis
Build Instrumented Programs
Dynamic Coverage Analysis
Dynamic Dataset Analysis
Profile Analysis
Dynamic Data Flow Coverage

Consider the following command:

```
start /wait contestbed source -arg1 -arg2...
```

The `start /wait` command is required when running multiple command lines. This command is not compulsory for single command usage.

Contestbed should always be passed the source in to be analysed, this can be name of the Set or the path to the file or TCF.

As well as the source to be analysed contestbed requires the analysis phases to be run. Analysis options are accessed via the Command Line by pre-typing menu selections. Specific examples can be seen in later subsections, but below is a list of all menu options and their relevant selection letters/numbers.

Main Options (top level menu)

- **0** Ignored
- **q** Quit Testbed
- **1** Select Static Phase Menu
- **2** Select the Instrumentation Phase Menu
- **3** Select the Dynamic Phase Menu

Static Options (Static Phase menu)

- **0** Return to the top level menu
- **q** Quit Testbed
- **1** Perform Main Static Analysis
- **2** Perform Complexity Analysis (this must be followed by an indication of whether full output is required for all or none of the procedures)
  - **a** full output for all procedures
  - **n** full output for no procedures
- **3** Perform Static Data Flow Analysis
- **4** Perform Cross-Reference
- **5** Perform Information Flow Analysis
- **6** Perform Data Object Analysis
- **7** Perform MCDC Test Case Planner

Please Note: Some analysis options cannot be performed until the successful completion of previous ones, i.e. Complexity Analysis requires Main Static Analysis to be performed first.

Instrumentation Options (Instrumentation Phase menu)

- `0` Return to the top level menu
- `q` Quit Testbed
- `1` Instrument the program
- `3` Enable Exact Semantic Analysis
- `2` Build the instrumented program

Please Note: Appropriate paths and environment variables must be set for compilation and linking.

Dynamic Options (Dynamic Phase Menu)

- `0` Return to the top level menu
- `q` Quit Testbed
- `1` Execute instrumented program
- `2` Perform Dynamic Coverage Analysis, followed by:

  - `p` Perform a procedure by procedure analysis
  - `f` Perform a full file analysis

  If Procedure by procedure was selected:
  - `a` Provide details for all procedures
  - `n` Provide details for no procedures

  If Procedure by procedure was selected:
  - `n` Do not provide a trace (mandatory for procedure based analysis)

  If full file was selected:
  - `n` Do not provide a trace
  - `y` Provide a trace

  If full file and trace was selected:
  - `a` Trace all lines (mandatory)

- `3` Perform Dynamic Data Set Analysis
- `4` Perform Profile Analysis
- `5` Perform Dynamic Data Flow

Examples of Usage

The below examples show the usage on a single file, **triangle.c/cpp**, however the syntax is the same for sets, replacing the file name with the set name, e.g. **myset**. You can also specify the TCF file instead of the file or set name. For more information on creating sets see Creating a Set on the Command Line on page 10.

These options should be passed as an argument along with the source to be analysed as a string of characters which relate to analysis routines required, e.g.:

```
start /wait contestbed triangle.c -112a34567q
```

Would mean:

- `1` Go to Static Analysis Phase
- `1` Perform Main Static Analysis
- `2` Perform Complexity Analysis
- `a` Analyse All procedures in source files
- `3` Perform Static Data Flow Analysis
- `4` Perform Cross Reference
- `5` Perform Information Flow Analysis

6	Perform Data Object Analysis
7	Perform MCDC Test Case Planner
q	Quit

This, then executes *LDRA Testbed* and performs full Static Analysis (not all options may be available on your implementation).

Command Line Mode can also perform both Instrumentation and Dynamic Analysis of the source file. Adding to the previous example:

```
start /wait contestbed triangle.c -112a3456702132q
```

Would mean:

1	Go to Static Analysis Phase
1	Perform Main Static Analysis
2	Perform Complexity Analysis
a	Analyse All procedures in source files
3	Perform Static Data Flow Analysis
4	Perform Cross Reference
5	Perform Information Flow Analysis
6	Perform Data Object Analysis
7	Perform MCDC Test Case Planner
0	Return to Main Menu
2	Select Instrumentation Phase
1	Instrument Source File
3	Enable Exact Semantic Analysis
2	Build the instrumented program
q	Quit *LDRA Testbed*

The above example will statically analyse the source file, instrument it, Enable Exact Semantic Analysis and then Build the instrumented program.

The instrumented program can also be executed and have Dynamic Coverage Analysis performed on it via Command Line Mode, e.g.

```
start /wait contestbed triangle.c -32pan34q
```

Would mean:

3	Go to Dynamic Analysis Menu
2	Dynamic Coverage Analysis
p	Produce procedure by procedure analysis
a	Analyse All procedures in source file
n	Select No Trace
3	Perform Dynamic Data Set Analysis
4	Perform Profile Analysis
q	Quit *LDRA Testbed*

More examples for popular analysis phases.

Perform Main Static Analysis, Complexity, Data Flow and Cross Reference:

```
start /wait contestbed sourcefilename.c -112a34q
```

Perform Main Static Analysis, Complexity, Data Flow, Cross Reference, MCDC Test Case Planner:

```
start /wait contestbed sourcefilename.c -112a347q
```

Perform all Static Analysis, Instrumentation and Build process:

```
start /wait contestbed sourcefilename.c -112a345670212q
```

Same process as above, all Static Analysis, Instrumentation and Build process, but in two commands to separate Static and Instrumentation phases from each other:

```
start /wait contestbed sourcefilename.c -112a34567q
start /wait contestbed sourcefilename.c -212q
```

Perform Main Static Analysis, Instrumentation, Build, Execution and Dynamic Coverage Analysis:

```
start /wait contestbed sourcefilename.c -112a345670212032panq
```

Same process as above, all Static Analysis, Instrumentation and Build Execution and Dynamic Coverage Analysis, but in three commands to separate Static and Instrumentation phases from each other:

```
start /wait contestbed sourcefilename.c -112a34567q
start /wait contestbed sourcefilename.c -212q
start /wait contestbed sourcefilename.c -32panq
```

For more command line arguments see Contestbed Command Line Qualifiers on page 14

Creating a Set on the Command Line

Sets can be created from the command line in different ways:

- Manually by creating the set and adding each file to the set
- Automatically from a TCF or project file
- Using the Build Import mechanism, see the section on **Contbbuildimport**

Creating Sets manually via the command line.

1. To create a system set called "**myset**" run the following command:

```
start /wait contestbed myset -create_set=system -1q
```

Alternatively you can create a group set by running the following command:

```
start /wait contestbed myset -create_set=group -1q
```

2. To add files to the set run the following command:

```
start /wait contestbed myset -add_set_file="path_to_file\file.c" -1q
```

It is possible to combine the above commands from stages 1 and 2, to create the set and add files to it in one go.

E.g.

```
start /wait contestbed myset /create_set=system /add_set_file="path_to_file
\file1.c"-add_set_file="path_to_file\file2.c" -add_set_file="path_to
_file\file3.c" -1q
```

The above command line will create a System Set named “myset” and add the files file1.c, file2.c and file3.c

Having successfully created the set, analysis can now be performed from the command line in the same way as you would for a single file. e.g. for the set “myset”:

Perform Main Static Analysis, Complexity, Data Flow and Cross Reference:

```
start /wait contestbed myset -112a34q
```

Creating Sets using a TCF via the command line.

Sets can be created easily from TCF files as they contain the set name and the files in the set.

A TCF file can also contain configuration settings used in analysis.

Consider the TCF Cashregister.tcf.

```
# Begin Testbed Set

   SET_TYPE = SYSTEM
   SET_NAME = Cashregister
   GENERATED_BY = SCRIPT

   # Begin Source Files

      File = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\Cashregister.c
      File = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\Main.c
      File = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\Productdatabase.c
      File = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\Specialoffer.c
      File = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\Userinterface.c

   # End Source Files

   # Begin Sysearch Include File Entries

      SearchPath = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src

   # End Sysearch Include File Entries

   # Begin Sysppvar Preprocessor Macros

      MacroEntry = TUTORIAL 1

   # End Sysppvar Preprocessor Macros

# End Testbed Set

# Begin Options

 $ Options for static analysis
   include = True
   open_all_includes = False
   shorten = True
   cstandards_model = MISRA-C:2012/AMD1
   cexternal_standard = MISRA-C:2012/AMD1

 $ Options for dynamic analysis
   nb_substitute_source = True
   nb_mechanism = makefile
   nb_makefile_name = Cashregister.mak
   nb_start_in_dir = C:\LDRA_Workarea\Examples\Toolsuite\Cashregister_5.0\Src\
   nb_makefile_command = mingw32-make -f "$(Makefile)" $(MakeGoal) $(MakeArgs)

# End Options

```

Using this TCF file to create a set will create a System Set named “Cashregister” and add the 5 files listed into the set. There are extra configuration options in this TCF that will be used.

The same command line syntax is used when performing analysis from a TCF.

```
start /wait contestbed Cashregister.tcf -112a34q
```

To create the set but not perform any analysis, use just the argument -1q.

```
start /wait contestbed Cashregister.tcf -1q
```

Removing Files from a Set via Command Line

Use the following command line qualifier to remove a specific file from the set:

```
-remove_set_file=path\to\file.c
```

If using spaces in file names remember to use quotation marks around the location:

```
/remove_set_file="C:\Source Files\remove_me.c"
```

Worked examples:

```
start /wait contestbed tbsdemo -1q -remove_set_file="C:\tbsdemo\tbsdem1.c"
```

You should first delete any workfiles for your set if you have done any system wide analysis such as Static Data Flow Analysis. You will be unable to delete the file from the set via the command line if these results exist.

Deleting Sets via the Command Line

Use the following command line qualifier to delete a set:

```
-delete_set=set_name
```

Worked example:

```
start /wait contestbed -delete_set=tbsdemo
```

Deleting Workfiles via the Command Line

To delete workfiles via the command line use the command line argument **-94q**, this will delete all workfiles for the specified file or set, e.g.

contestbed <file or set> -94q

Note: use **-delete_set** or **-delete_single_file** to completely remove the file or set from LDRA.

Additional workfile deletion arguments can also be used

-91q	Delete Workfiles
-92q	Delete Workfiles and Results
-93q	Delete Instrumented Soure and Executable Programs
-94q	Delete All
-95q	Delete Dynamic Coverage Analysis Results

Confirming Analysis Results via the Command Line

LDRA Testbed uses the following flag to test whether results for the selected phases exist. If results exist for a selected phase, it will not be run.

```
start /wait contestbed <path>\Dispense.cpp -112q
-batch_mode_verify_results
```

Example Output:

```
-------------------------
Command Line Mode Started
-------------------------
Main Static Analysis results verified
Complexity Analysis results verified
```

Partial Instrumentation in Command Line Mode

For users wishing to only instrument certain files/procedures in their analysis scope, the following command line qualifiers can be used when analysing from the command line.

To force the Instrumenter to not instrument a file in a Set use the following qualifier:

```
-no_inst_file=<path>\/<filename.ext>
```

E.g.

```
start /wait contestbed tbsdemo -1120212q -no_inst_file=C:\ldra_workar
```

```
ea\examples\tbsdem1.c
```

To force the Instrumenter to not instrument a specific procedure, use the following qualifier:

```
-no_inst_proc=<path>\/<filename.ext>:<procedure>
```

E.g.

```
start /wait contestbed testrian.c -1120212q
-no_inst_proc=C:\ldra_workarea\examples\testrian.c:equal_sides
```

Contestbed Command Line Qualifiers

Below is a comprehensive list of the command line qualifiers, including those mentioned already.

General

-create_set=<system|group>	Creates the set as a system or group
-add_set_file=	Adds the specified file to the Set
-remove_set_file	Removed the specified file from the Set
-delete_set	Deletes the specified Set
-reanalyse_changed_set	Re-analyses the set after adding a file to a set that has previously been analysed
-delete_single_file	Deletes a single file. e.g. -delete_single_file=C:\Project\Myfile.c. This will delete all workfiles for this file including the GLH
-quit	Equivalent to -1q
-permdir	Specifies the location of the Permdir, default is the workdir. Using this Command Line Qualifier modifies the permdir entry in INI causing this setting to persist for all future analysis until it is modified again.
-reset_options	Resets all options to their defaults for the file/set named on the command line. Option changes are typically made through the User Interface or by importing settings from a TCF file.
-compiler_reset=<compiler>	Resets the compiler and all options are reloaded.
-tcf	Used to specify a TCF file containing Options, Sysearch Include File Entries or Sysppvar Preprocessor Macros sections. These will override existing settings for a file or set specified on the command line.
-getstaticid	Use with a source file or set command line argument -getstaticid= file. Writes the staticid number of the named source file to the standar error channel. Use -getsid_outfile=outfile to write the staticid number to a named file. The result is less than 1 if the staticid number cannot be found. This qualifier should be used in place of the utility TBgetstaticid.
-getsid_outfile=outfile	Used with -getstaticid.
-batch_mode_verify_results	When used with command line analysis, tests whether results for the selected phases exist. If results exist for a selected phase, it will not be run.
-force_analysis	Overrides result checking and forces analysis to be re-ran even if results are present and up-to-date. Use: contestbed testrian.c -112a345q -force_analysis

-dosnames	Used to allow long files names to be used on Windows when they exceed the 8.3 limitation.
-nodosnames	Overrides usage of dos names if disabled in INI file as default
-export_tbed_tcf=<tcf>	Generates a Testbed TCF at the specified location
-workdir=<directory>	The LDRA Testbed will use an alternative directory to store its' workfiles. The default directory is: ".tbwrkfls".
-keep	Do not erase any workfiles.
-nokeep	Erase workfiles when possible.
-gen_tcf_source_dir=	Generates a TCF for source files from the specified Directory
-gen_tcf_source_files=	Generates a TCF for source files from the specified Files, use ; as a separator, can use wild cards
-gen_tcf_set_name=	Specify the Set name in the TCF file
-gen_tcf_fname=	Specify the name of the TCF file.
-gen_tcf_recurse_dirs	Generates a TCF for all files from sub-directories
-generate_email	creates a text file in the workfiles directory that can be used as a template to email LDRA support
-default_lang_ext	Defines the default equivalent extension for files of unknown type analysed or added to a set via the command line
-auto_baseline	When previously analysed source code is modified, if necessary a new baseline is created from existing results before running batch mode analysis.
-delete_last_baseline	Deletes the highest numbered, latest, baseline
-startindir=	Contestbed will attempt to create the directory if it does not exist and then change the current working directory to that named. Some other arguments or Testbed.ini flags that are relative will be relative to the named directory.
-tb_workfiledir=<directory>	Specifies/creates the directory for analysis results and workfiles. If a file or set is named on the command line, applies only to that file or set, otherwise applies to new files/sets during Testbed session.
-force_val=	Processing of abort or exit as special keywords when not including stdlib.h. Set to exit or abort. Same as INI flags FORCE_EXIT_VAL=TRUE and FORCE_ABORT_VAL=TRUE

Analysis Phases

-review	Runs Static analysis phases, Equivalent to -1120345q

-build_instrumented	Runs static analysis, instrumentation and build phases
-run_required_dynamic	Runs phases required for Dynamic Analysis: If no results exist is equivalent to the command: /112a302120312panq If results already exist for a phase will not repeat analysis for that phase.
-run_required_dyndflow	Runs phases required for Dynamic Dataflow Coverage. Replaces the command -112a34021203125q e.g. contestbed testrian.c -run_required_dyndflow If results already exist for a phase will not repeat analysis for that phase.
-run_obj_box_phase	Runs Object Box phase

Static Analysis

-cpenfile=<file>	Specify filename of (non-standard) penalty file
-cpppenfile=<file>	Specify filename of (non-standard) penalty file
-c_dialect=	Change dialect for analysis. E.g. C_DIALECT=RHAPDY
-cpp_dialect=	Change dialect for analysis. E.g. CPP_DIALECT=RHAPDY
-cstandards_model=	Sets the CSTANDARDS_MODEL in INI File
-cppstandards_model=	Sets the CPPSTANDARDS_MODEL in INI File
-crepfile=<file>	Specifies the creport.dat
-cpprepfile=<file>	Specifies the cppreport.dat
-csyscallsfile=<file>	Specify Language specific syscalls data file. e.g. CSYSCALLSFILE=msvc_syscalls.dat
-cppsyscallsfile=<file>	"Specify Language specific syscalls data file. e.g. CPPSYSCALLSFILE=my_syscalls.dat
-cvalsfile=<file>	Specify (non-standard) filename and location of the file named cvals.dat as default
-cppvalsfile=<file>	Specify (non-standard) filename and location of the file named cppvals.dat as default
-ctbendfile=<file>	Specify non-standard) filename and location of the file named ctbend.dat as default
-cpptbendfile=<file>	Specify non-standard) filename and location of the file named cpptbend.dat as default
-cmacdatfile=<file>	Specify non-standard) filename and location of the file named cmacdat.dat as default

-cppmacdatfile=<file>	Specify non-standard) filename and location of the file named cppmacdat.dat as default
-include	Expand include files where possible.
-noinclude	Do not expand include files
-sys_inc_mode=<n>	Opening of system includes, 0 open none, 1 open only if #include has path, 2 open if relative path, 3 open all
-sys_inc_local	If not opening all system includes, allow opening if system include is found same directory as source file.
-shorten_include_nest_limit=<n>	Maximum nest depth of includes
-unexpanded_inc_action=	Specify the action on unexpanded included statements =0 leave include statement unmodified =1 add full path to nested include statements =2 remove include statement with comment =3 add full path to nested include statements (Auto Mode)
-shorten	Create temporary version of source file (sourcefilename.cod) which has code length shortened, carriage returns added and white-space eliminated. Part of Main Static Analysis, performs some preprocessing and include file search. Generated .cod file retained with KEEP qualifier. This is useful when analysing preprocessed files.
-noshorten	This Command Line Qualifier can be utilised if LDRA Testbed defaults are altered. See -shorten
-auto_macro	Switches on the Auto Macro facility (C\C++ only). With this facility any undefined macros are automatically defined in a sysppvar.dat file. The sysppvar.dat file is decided by the normal search mechanism.
-noauto_macro	overrides auto macro feature if enabled in INI file as default
-auto_macro_value="0"	Specifies the value given to the macros, used with -auto_macro
-forcedataflow	Enables LDRA Testbed to analyse a package specification through the Static Analysis phase up to and including Static Data Flow Analysis, depending on the menu options selected. This extra analysis is generally not worthwhile, though a Static Flowgraph and other analysis results can be gathered if the user so wishes.
-noforcedataflow	This Command Line Qualifier can be utilised if LDRA Testbed defaults are altered.
-preprocess	This option causes the LDRA Testbed to preprocess the source file before lexical analysis - usually with the language preprocessor.
-nopreprocess	Do not pre-process source file before lexical analysis.

-preprocess=<...>	This option allows the user to name the output file generated by running the C language preprocessor.
-comments	Keep comments in reformatted code.
-nocomments	Remove comments in reformatted code.
-continue_system_analysis	Continues analysis for System Sets if one or more file fails analysis.
-nocontinue_system_analysis	Overrides CONTINUE_SYSTEM_ANALYSIS=TRUE if set in the INI
-parsing_progress_op=<file>	The cod, ref and oln files for the selected file are displayed after analysis. This also disables the intermediate pass of main static analysis for C++.
-full_parsing_progress_op=<file>	Incorporates the intermediate pass of main static analysis for C++.
-oln_level=coarse|fine	Used in conjunction with -parsing_progress_op or -full_parsing_progress_op

Instrumentation

-no_inst_file=	Specifies a file to be excluded from instrumentation, see Partial Instrumentation
-clear_no_inst_file=	Restores the named file to normal instrumentation. This is used in combination with -no_inst_file=<filename> to switch the status of files in a set between normal instrumentation and no instrumentation via the command line. Note that -clear_no_inst_files switches all files to normal instrumentation. If instrumentation results already exist, they must be removed using -93q before applying these command line qualifiers.
-no_inst_proc=	Specifies a procedure to be excluded from instrumentation, see Partial Instrumentation
-clear_no_inst_proc=	Restores the named procedure to normal instrumentation. This is used in combination with -no_inst_proc=<filename>:<procname> to switch the status of procedures between normal instrumentation and no instrumentation via the command line. Note that -clear_no_inst_procs switches all procedures to normal instrumentation. If instrumentation results already exist, they must be removed using -93q before applying these command line qualifiers.
-cinstrfile=<file>	Specify filename of (non-standard) instrumentation data file. The filename may include its path. Once used, this option persists for the file under analysis until the analysis results for this file are deleted.
-cppinstrfile=<file>	Specify filename of (non-standard) instrumentation data file. The filename may include its path. Once used, this option persists for the file under analysis until the analysis results for this file are deleted.

-no_coverage_inst	No Instrumentation for Coverage, no procedures will be instrumented. Testbed will use the original file(s) Can be used with [cod|ref|off] e.f. **-no_coverage_inst=of**f
-instr_flush	Switches flushing of exh file instrumentation on
-instr_template_io	Switches template I/O instrumentation on
-iprog_switch_os_format	Switches Unix format instrumentation on
-preproc_iprog	Switches preprocessing of instrumented file on
-iprog="inszt_$$o"	This option tells LDRA Testbed to write the instrumented source to the specifically named file. No -idir option is allowed in combination when using a full filename. This option is not usable with Sets. If -iprog is not used the instrumented file is placed, by default, in the -idir directory using a name derived from the source filename with the prefix ""inszt_"". It is the users responsibility to ensure that the -iprog file can be written. i.e. any named directories in the full file name must exist and the -iprog must be writeable.
-thisdir	Puts instrumented program and execution history in current directory using LDRA Testbed, not the directory of the original source code. See also IPROG.
-idir=<directory>	The LDRA Testbed will attempt to create instrumented files in the named directory. The default is the current working directory. Note that the user must have permission to write in the -idir directory. This option is not allowed with -iprog when using a full filename. Once used, this option persists for the file under analysis until the analysis results for this file are deleted.
-exhdir=<directory>	Specify the directory where LDRA Testbed will look for an execution history e.g. -exhdir=C:\my_exhdir
iname_prefix=<...>	Allows the user to specify that they wish to differentiate the instrumented source code from the original source code with a prefix of a specified string.e.g.: iname_prefix=inst_ will produce an instrumented version of triangle.c called: inst_triangle.c This qualifier cannot be used in conjunction with iprog.
iname_suffix=<...>	Allows the user to specify that they wish to differentiate the instrumented source code from the original source code with a suffix of a specified string.
-instr_no_atexit	Disables the use of atexit, can also be set via the INI flag INSTR_NO_ATEXIT=TRUE

Dynamic Analysis

-coverage="n"	Sets the Coverage level for the Dynamic Report

1 = Statement
2 = Statement & Branch/Decision
3 = Statement & Branch/Decision & LCSAJ

-dataset="Run 1" Specify name of data set used by Dynamic Coverage Analysis

-dyninit Initialise profiles

-nodyninit Append profiles
This option can only be used with the command line interface described below. It ensures that any previously created Dynamic Coverage Analysis profiles are deleted. Default action is to append the results. e.g.:
`iname_suffix=_inst`
will produce an instrumented version of `triangle.c` called:
`triangle_inst.c`
This qualifier cannot be used in conjunction with `iprog`.

-iname_extension=<...> - allows the user to specify that they wish to differentiate the instrumented source code from the original source code with an extension of a specified string.
e.g.: `iname_extension=i_c`
will produce an instrumented version of `triangle.c` called:
`triangle.i_c`
This qualifier cannot be used in conjunction with `iprog`.

-iname=same - allows the user to specify that they wish to use the source file name for the instrumented source code. This is still a safe option as the user will have to use the qualifier `idir=<directory>`.
This qualifier can only be used in conjunction with `idir=<directory>`.
This qualifier cannot be used in conjunction with `iprog`.

-multi_set_history_exh Use to process history.exh containing records from more than one set

-use_archive_exh Enables batch mode dynamic coverage analysis to be run using previously archived execution histories, if no new execution histories are found.

-dyn_proc=<procname> Specify a procedure to use when using 32psn

-clear_dyn_procs Clear previously selected procedures

-clear_no_inst_files Clears any files that have been set as not instrumented

-clear_no_inst_procs Clears any procedures that have been set as not instrumented

-build_type=<type> Specify the Build type. Valid values: compile, build, ide, host_target, makefile, project_makefile, project_build

Reports

The below arguments can be used as they are to generate the report in ASCII or HTML per the default, or the report type can be specifed, e.g. **-generate_code_review=ASCII** both ASCII and HTML can be generated by using the command **-generate_code_review=ASCII,HTML**

-generate_code_review	Generates the Code Review Report
-generate_doa_report	Generates the Data Object Analysis Report
-generate_dyndflow_report	Generates the Dynamic Data Flow Report
-generate_quality_review	Generates the Quality Review Report
-generate_test_manager	Generates the Test Manager Report

The below arguments require a TBpublish License.

-publish_as_txt	Publishes the report as a .txt file, can also be set in the INI with flag PUBLISH_AS_TXT=TRUE
-nopublish_as_txt	overrides if the INI flag PUBLISH_AS_TXT=TRUE is set
-publish_new_only	
-publish_rep_type=	Specifies the report type, ASCII or HTML
-publish_to_dir=<directory>	Specifes the location for TBpublish Reports
-publish_to default_dir	Publishes the reports to the default directory as defined by the INI flag PUBDIR=
-html_index_template=	Specifies an Index Template from: MISRA, DYNAMIC, MANAGEMENT, AV_STANDARD or FULL. Can be set via INI flag HTML_INDEX_TEMPLATE

Additional

-congendir=<dir>	Specifies the directory where congensysppvar should generate the TCF

Exit Codes

When using contestbed on the command line exit codes can be used to check for errors.

64	Invalid Command Line
66	An input file does not exist or is not readable
70	An internal software limitation has been detected.
73	An output file or directory cannot be created
80	Main Static Analysis incomplete
81	Instrumentation incomplete

82	Dynamic Coverage Analysis incomplete
83	Other Analysis incomplete
84	Build failure due to execution of command
85	Failed to execute Instrumented program
86	Build command returned a non-zero value, usually indicating an error
87	Error in sysppvar generation phase. This exit code can be overridden by later exit codes such as incomplete main static analysis.
103	Licensing error

These exit codes can be checked by using %error_level% in your scripts.

Contbrun

Overview

Basic Command Line Format

To use *TBrun*'s command line mode, **contbrun** needs to be invoked via the command line with arguments that specify:

- The file or set to be analysed, unless specified in the TCF.
- Where *TBrun* can get its TCF information from.
- Any additional arguments.

In the examples below the following points should be noted:

1. Commands are run from the *LDRA Tool Suite* installation directory. If you wish to run them from elsewhere then the location of **contbrun.exe** and **contestbed.exe** need to be specified.
2. The `start /wait` command is required when running multiple command lines. This command is not compulsory for single command usage.
3. Contbrun, like *TBrun*, requires certain analysis phases to be ran, if no analysis has been run **contbrun** will automatically run the minimum required analysis.
4. In most of the examples absolute paths have been used, however, relative paths can also be utilised.

   Relative paths on the command line are relative to the workarea directory (`C:\LDRA_Workarea` by default).

   If the **Testbed.ini** flag **SWITCH_TO_WORKAREA=FALSE** is set, relative paths will be relative to the directory the command is ran from.

   Relative paths contained inside TCF files are relative to the location of the TCF file.

Running *TBrun* Tests on a Single File

Contbrun follows the format:

```
start /wait contbrun <file_name> -tcf=<Path_to_TCF> -<argument> -quit
```

Where the full paths to **contbrun**, the *source file* and the *TCF* file are used where required.

Where the file is passed along with the TCF the TCF should be preceded with the qualifier **-tcf=**

For example to run regression on `ggrocers.c` using the `ggrocers.tcf` file, use the following command:

```
start /wait contbrun C:\LDRA_Workarea\Examples\C_tbrun_examples\Ggrocers
.c -tcf=C:\LDRA_Workarea\Examples\C_tbrun_examples\Ggrocers.tcf -regress
-quit
```

If your TCF file contains the file information then the following command line format can be used:

```
start /wait contbrun <Path_to_TCF> -<argument> -quit
```

e.g.

```
start /wait contbrun C:\LDRA_Workarea\Examples\C_tbrun_examples\Ggrocers
.tcf -regress -quit
```

Note, if using the TCF to specify the files, make sure that if they contain relative paths that they are relative to the location of the TCF file. If unsure use absolute paths.

Multiple arguments can be used to perform more than one task at once for example, adding `-tas` will create the TBrun Analysis Scope Report:

```
start /wait contbrun C:\LDRA_Workarea\Examples\C_tbrun_examples\Ggrocers
.tcf -regress -tas -quit
```

See Contbrun Command Line Qualifiers on page 27 for more information on the command line arguments.

Running *TBrun* Tests on a Set

To run a set from the command line the set has to first be created and files added. Sets can be created on the command line and files added in two ways.

Creating a set from a TCF, BTF or PTF

The best way to create a set is via a TCF or BTF, a BTF can be created using the Build Import System to parse your build logs and create a BTF containing all your source files and configuration settings. See the Build_Import.pdf for more information, you can also see Contbbuildimport on page 34 of this manual for information on using this feature from the command line.

To create a set from a TCF or BTF use the command:

```
start /wait contbrun <path_to_TCF/BTF> -quit
```

For example

```
start /wait contbrun C:\LDRA_workarea\Examples\myset.tcf -quit
```

The above command will automatically create the set with the name and settings specified in the TCF and run the required analysis

Creating a Set Manually

To create a set manually from the command line **contestbed** needs to be used.

```
start /wait contestbed <set_name> -create_set=group -1q
```

Where **group** is the type of set (**group** or **system**).

Add files to a set with the parameter:

```
start /wait contestbed <set_name> -add_set_file=<path_to_file> -1q
```

These can be combined to create the set and add files in the same command.

```
start /wait contestbed <set_name> -create_set=group
-add_set_file=<path_to_file> -1q
```

Repeat -add_set_file=<path_to_file> for each file.

Running Regression on a Set.

Contbrun follows the format:

```
start /wait contbrun <Path_to_TCF> -<arguments> -quit
```

```
start /wait contbrun <set_name> -<arguments> -quit
```

for example:

```
start /wait contbrun C:\LDRA_workarea\Examples\myset.tcf -regress -quit
```

```
start /wait contbrun myset -regress -quit
```

When specifying the TCF, TBrun will create the set if the set does not already exist, run any required analysis and run regression on the set using the test case values specified in the TCF. **This assumes that the TCF contains the set information.**

When specifying the set, the set must already exist.

Using Multiple TCFs with CSP files

CSP files provide a way to run multiple TCFs on the same file or set sequentially, removing the need to invoke *TBrun* separately for each TCF.

```
start /w contbrun C:\LDRA_Workarea\Examples\C_TBrun_Examples\Ggrocers.c
-csp=C:\LDRA_Workarea\Examples\C_TBrun_examples\Ggrocers.csp -quit
```

Note: If you are using **contestbed** to run regression replace the argument **-csp** with **-tbruncsp** and **quit** with **-1q**

Where `Ggrocers.csp` contains the following:

```
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\Ggrocers1.tcf
-regress
-close
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\Ggrocers2.tcf
-regress
-close
```

Note: the `-tcf` entry is the location of the TCF in relation to the *LDRA Testbed* start-in directory, if unsure enter the full path to the TCF.

CSPs use a subset of the available command line arguments:

`-tcf`	Loads the TCF into *TBrun*.
`-regress`	Runs the regression harness.
`-close`	Closes the sequence.

Users with a great number of test cases may wish to reduce the size of their TCFs by extracting the common sequencer and harness properties and options into a separate TCF. You can then use this TCF in conjunction with those that contain only the test cases in a CSP.

```
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\Ggrocers1.tcf
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\GgrocersProperties.tcf
-regress
-close
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\Ggrocers2.tcf
-tcf=C:\LDRA_Workarea\Examples\C_TBrun_Examples\GgrocersProperties.tcf
-regress
-close
```

Contbrun Command Line Qualifiers

Some arguments require a sequence to be selected, this can be specified on the command line with -seq= or -newseq=, or when using a TCF/BTF a sequence will be created.

General

-seq=	Specifies an existing Sequence when not using a TCF/BTF
-newseq=	Creates a new Sequence when not using a TCF/BTF
-newseqcheck	when used with -newseq=<name> checks for an existing sequence called <name> and loads it rather than creating <name>_1
-tcf=Tcf_File_name	Used in conjunction with a File or Set, if the TCF/BTF contains the File or Set information then the File or Set does not need to be the command line and this argument is not required.
-sequencer	Runs the sequencer program for existing test cases if a sequence has been selected.
-regress	Runs the harness if a sequence has been selected.
-quit	Quits TBrun after the analysis and test have been completed
-delete_results	Deletes all results and workfiles for the specified File or Set
-delete_results=d	Deletes Dynamic Coverage results only for the specified File or Set
-delete_results=w	Deletes workfiles for the specified File or Set
-delete_results=i	Deletes Instrumented Source and Executable Programs for the specified File or Set

When using **tbrun** instead of **contbrun** the following arguments can be used.

-newinstance	Forces the invocation of a new instance of TBrun rather than activating the current version.
-guimin	invokes TBrun minimized
-guimax	Invokes TBrun maximised.
-nogui	Invokes TBrun without displaying the main GUI.

Note: using TBrun, even with **-nogui**, may prompt dialogs, for full automation on the command line, **contbrun** should be used.

Reports

-unit_publish_to=<dir>	Publishes regression and coverage reports
-rsi	Generates the diagnostic report file (sif details etc).

-ppg	Generates the Procedure Parameters and Globals report for the files.
-phc	Generates the Procedure Header Comments file.
-chr=Module_Number	Generates a class hierarchy report for the file with the supplied module number.
-udt=Module Number	Generates a User Defined Types report for the file with the supplied module number.
-ugr	Generates the Unresolved Globals Report.
-upr	Generates the Unresolved Procedures Report.
-tas	Generates the TBrun Analysis Scope Report

Box Modes

-greymode	TBrun goes into isolation mode for new sequences. Not needed when using TCFs that contain box information.
-box=black	Sets all the files in the sequence to black box if a sequence has been selected.
-box=white	Sets all the files in the sequence to white box if a sequence has been selected.
-ibox=<box>	Requires -box=white to be set. The option <box> is one of: w = white l = light grey d = dark grey. p = pale grey s = strong grey r = rose m = mauve E.g. `contbrun file.tcf -box=white -ibox=d -regress -quit`

Additional

-lang=lang_code	For analysis of multi language files this will force the sequence created by **-tcf=** to be of the given language code. Some files must be of this language code for the code to be accepted.
-gentcf=<num>	Creates TCF, where **<num>** is: **=-3** default for import **=-2** current ini flag settings generated by export mask

	=-1 test cases, user globals, file based code, and text sequence based code and text. **=0** all (default) **=1** existing test cases **=2** user globals **=3** stubs **=4** isolated procedures **=5** new test case **=6** file based code and text **=7** sequence based code and text **=8** options and properties **=9** excluded and white box files
-ugr+	Runs the automatic user global creation facility.
-msupr+	Runs the managed stub creation facility.
-alias_relative_tcf=T	Automatically alias files relative to the current TCF/BTF file
-alias_by_directory=T	Automatically alias files in a directory of an already aliased file.
-test_build	Performs a Test Build
-import_tbed_commands=t|f	Set to "t" to set the build & execution commands as the Testbed defaults, set to "f" to set the build & execution commands as the TBrun defaults
-excludefile=<File>	Exclude the specified file from the Sequence
-excludeallexceptfile=<File>	Exclude all files except the specified file from the Sequence
-includefile=<File>	Include the specified file in the Sequence
-tcf_mode=	If "retain" is specified then if the sequence name already exists the new sequence is not created, the existing sequence is loaded instead. If "overwrite" is specified then the named sequence is backed up, and then the tcf replaces the named sequence. If "new" is specified then a new sequence is created.

Regression - Contbrun vs Contestbed

LDRA Testbed, either through its standard executable, **testbed.exe**, or its console mode executable **contestbed.exe,** can be used to regress TCF's.

This can often be a more efficient method of TCF regression since it allows *LDRA Testbed* to set up the proper start-in directory and invocation flags for calling the *TBrun* executable `tbrun.exe` or the console executable **contbrun.exe** (if the INI flag `USE_CONTBRUN=TRUE` is set).

However, regressing TCF's through **tbrun.exe** or **contbrun.exe** on the command line often yields useful diagnostic output during the regression, and can be very useful when trying to understand the behaviour of your TCF regression.

When using **testbed.exe** or **contestbed.exe** replace the command line parameter:

/tcf=<Path_to_TCF>

with

/tbruntcf=<Path_to_TCF>

When using **contbrun** the command line arguments need to be specified on the command line e.g.

```
start /wait contbrun ggrocers.tcf -regress -quit
```

When using **contestbed** if no arguments are specified on the command line the default arguments will be performed. The default arguments are `regress quit`.

You can set the default arguments via the INI Flag `TB_TBRUN_TCF_ARGS` e.g.

```
TB_TBRUN_TCF_ARGS=regress tas quit
```

Then the command:

```
start /wait contestbed ggrocers.tcf
```

is equal to:

```
start /wait contestbed ggrocers.tcf -regress -tas -quit
```

Prompt or Batch files

The *TBrun* command line mode can be used from a DOS Box or Command Prompt or from a batch file.

Creating and Exporting Tests

Tests should be created via interactive use of *TBrun* and exported in a TCF. Experienced users can create the TCFs themselves via a text editor, therefore skipping the use of the *TBrun* GUI.

Exit Codes

When using contbrun on the command line exit codes can be used to check for errors.

64	Invalid Command Line
65	The input data was incorrect
70	An internal software limitation has been detected.
73	An output file or directory cannot be found
90	Regression Failure
91	Build Failure
92	Unable to execute Harness Program.
103	Licensing error.

These exit codes can be checked by using %error_level% in your scripts.

See also Exit Codes contestbed can return during analysis.

Command Line Sequence Naming Conflicts

One complication that needs to be handled when regressing a sequence in a tcf through the command line is how to handle an existing sequence of the same name.

The –tcf_mode flag

By default, when a tcf is invoked from the command line, if a particular file or set already has a sequence of that same name, then a new sequence will be created with an "_1" appended, unless there is already a sequence with that new name, if that is the case "_2" will be used, or "_3","_4", etc. until there is no name conflict.

The "-tcf_mode=<mode>" flag allows a user to override this behaviour. This argument can be used either directly from the *TBrun* command line or through the "/tbruntcfargs="<args>"" option of the *Testbed* command line. There is also an associated Testbed.ini flag "TBRUN_SEQ_TCF_CREATE_MODE" for changing what behaviour is used as the default.

For example, if sequences "original" and "original_1" exist, and a tcf containing the sequence name "original" is regressed from the command line, then sequence "original_2" will be created.

By using the "-tcf_mode=<mode>" argument, this behaviour can be changed. "-tcf_mode" has the following options:

-tcf_mode=new

The default behaviour of creating a new sequence when there is a naming conflict. Using this explicitly will override the behaviour specified by the TBRUN_SEQ_TCF_CREATE_MODE Testbed.ini flag.

-tcf_mode=retain

If the sequence name already exists, then new sequence is not created, instead the existing sequence is loaded instead. If the command line also contains the "-regress" flag, then this sequence will be regressed as well.

-tcf_mode=overwrite

The named sequence is backed up, and then the tcf replaces the named sequence.

The values for TBRUN_SEQ_TCF_CREATE_MODE are NEW / RETAIN / OVERWRITE and correspond to the "-tcf_mode" values "new"/"retain"/"overwrite" respectively.

The -newseqcheck Argument

In addition, in cases where a sequence is being created from the command line, but a tcf is not providing the sequence name, *TBrun* can use the argument "-newseqcheck", with the argument "-newseq=<name>", to help to avoid naming conflicts. When "-newseqcheck" is used, then *TBrun* will check if there is an existing sequence with the name specified by the "-newseq" option, and load that one, instead of creating a new sequence of the name "<name>_1".

TBini

TBini can be used to modify the settings in the **Testbed.ini** file. This file is situated in the directory as defined via the TESTBED environment variable is instrumental to the behaviour of *LDRA Testbed* for any analysis. A change of some of the control flags within the file will cause the results of a *LDRA Testbed* analysis to differ.

LDRA has developed a program for automation called **tbini.exe**, that has the capability of changing the **Testbed.ini** file from the command line, or via batch or script. At the start of each analysis, **tbini.exe** can be invoked to change or add any flag that the user requires.

For example, if the user wishes the "BUILD_OPTIONS_FILE=" flag to use a different compiler data file, then

```
start /wait TBini BUILD_OPTIONS_FILE=c:\ldra_toolsuite\my_testbed.dat
```

should be used.

TBini.exe is ideal when a large number of source analyses are required to have various LDRA set-ups.

For a full list of flags available, look at the chapter on Testbed or TBrun INI flags in the Testbed or TBrun manuals.

By default, all entries passed to **tbini.exe** go into the **[<lang> LDRA Testbed]** section of the **Testbed.ini** file. Users with multiple language LDRA installations or multiple compiler configurations may need to specify the section of the **Testbed.ini** file to write to.

For example, to define a full section name, use the *Section* argument:

```
-Section="C/C++ LDRA Testbed"

start /wait TBini -Section="C/C++ LDRA Testbed" WORKDIR=g:\testbed\

tbini -Section="C/C++ LDRA Testbed" WORKDIR=g:\testbed\
```

Alternatively use the *Profile* argument to specify the section:

```
-Profile="C/C++"

start /wait TBini -Profile="C/C++" WORKDIR=g:\testbed\

tbini -Profile="C/C++" WORKDIR=g:\testbed\
```

To reset the flag to the default, simply give the flag no value, take care when using this as the default value may not be the same as what the flag was previously.

Contbbuildimport

There are two executables available for Command Line execution:

- tbbuildimport
- contbbuildimport

The “contbbuildimport” executable is a console only version which is suitable for Command Line only environments such as Linux headless servers. It produces logging directly to the command line.

Example scripts can be found in /Utils/TBbuildimport.

The following pages describe the command line options available followed by examples

To provide the Build Command

```
-build_cmd="make"
```

To provide the Start-In Directory

```
-startin_dir="path"
```

To provide the path to the TBmakelogparser Settings file

```
-settings="path\tbmakelogparser.dat"
```

To Run Build Command and TBmakelogparser

```
-build
```

To Re-Run TBmakelogparser

```
-parse_only
```

To Close the dialog once completed

```
-quit
```

Specify a path to a text file that will be populated with the generated BTF files as a result of the build

```
-btf_listing_file="btf_listing.txt"
```

To provide the Pre Build Command

```
-pre_build_cmd="cmd"
```

To provide the Post Build Command

```
-post_build_cmd="cmd"
```

To run the Find Compiler Preprocessing option for the first build target found use

```
-compiler_preprocessing
```

To run the Find Compiler Preprocessing option for a specified build target use the following where target is the name from the Build Targets page on the dialog

```
-compiler_preprocessing_target="Program.exe"
```

To specify that Find Compiler Preprocessing should be run for all files within the build target use

```
-compiler_preprocessing_all_files
```

To specify that Find Compiler Preprocessing should be run only for a specific list of files use the following option specifying files by their filename and extension in a comma separated list

```
-compiler_preprocessing_files="lamptype.cpp,cell.cpp"
```

Example

```
tbbuildimport.exe -build_cmd="make" -startin_dir="C:\test" -build
-compiler_preprocessing -compiler_preprocessing_all_files -btf_listin
g_file="C:\test\listing.txt" -quit
```

Find Compiler Preprocessing can be run for all Build Targets found via a separate script. The script is output after a successful build when the command line argument is specified

```
-compiler_preprocessing_script
```

There are two scripts produced in the Start-In directory specified in the build:

```
run_tbbuildimport_compiler_preproc.bat

tbbuildimport_compiler_preproc.py
```

The first script is a wrapper script that configures python for the platform you are executing on and then calls the second script. You may invoke the python script directly if you wish. The python script can be edited prior to execution in order to configure which build targets will run the Find Compiler Preprocessing functionality (see the Command Line Example Usage section for an example of these scripts)

The example below runs a build process, processes the output from the build and passes each of the build targets found to LDRA for analysis

The processing is handled in Python for cross platform compatibility

The Find Compiler Preprocessing stage is optional and can be performed on a Project by Project basis

```
tbbuildimport.exe -build_cmd="make" -startin_dir="C:\test" -build -
compiler_preprocessing_script -btf_listing_file="C:\test\listing.txt"
-quit
```

When command above is executed, the GUI will invoke and the build will start in the directory C:\test by running the command “make”. When the build has completed, the build output will be processed and build target files (.btf) will be generated in the start in directory C:\test

Using TBmanager on the Command Line

TBmanager can be ran from the command line by calling tbmanager.exe with the required arguments. e.g.

tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" <additional_qualifiers> -close

The 1st argument must always be the path to the TBmanager Project File (.tbp).

When using the command line mode you must always pass the first and last name of the project user, e.g. **-first="Jane" -last="Wilson"**

Use the command line qualifier **-close**, to close the project when complete.

A TBmanager Project can be created from the command line using the qualifier **-new_project**, e.g.

tbmanager.exe C:\myproject\project.tbp -first="Jane" -last="Wilson" -new_project -close

The command line qualifiers can be combined to perform multiple actions in one TBmanager call.e.g.

tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" -apply_view_filter="path_to_view_profile" -verify_tci_grid -export_view_as_csv="path_to_view_profile" -close

The example above will open the Project and log in, Filter the Views according to the view profile. Verify all Visible TCI's in the TCI Grid. Export the Views specified in that view profile to a csv file. Close the project.

All the examples in this section use the Windows executable name "**tbmanager.exe**" on Linux systems repalce this with "**tbamanger**".

TBmanager Command Line Qualifiers

Below are the available command line qualifiers for use with tbmanager.exe, each qualifier includes a description and an example of usage. On Windows the qualifiers can be preceded by "/" or "-", on Linux they must be preceded with "-" e.g. -first=<name>.

-first=<name>

First name of project user to log in e.g. Jane

-last=<surname>

Last name of project user to log in e.g. Wilson

-close

Closes the project

-new_project

Creates a new project using the location specified and the First and Last name arguments e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" -new_project
-close
```

-export_source_desc="path_to.xml".

Exports the source description of the project to the file specified. Equivalent to ***Reports->Export Source Description*** e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" -export_sour
ce_desc="path_to.xml" -close
```

-export_tbmspec="path_to.tbmspec"

Exports a tbmspec file containing the contents of the project to the file specified. Equivalent to ***Reports->Export TBmanager Spec File*** e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" -export_tbms
pec="path_to.tbmspec" -close
```

-import_tbmspec="path_to.tbmspec"

Imports the contents of a tbmspec file into the project. Equivalent to **Import->Import TBmanager Spec File** e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-import_tbmspec="path_to.tbmspec" -close
```

-import_from_word="path_to.docx"

Imports from the Word document according to the settings already specified for the document. Please note you must have already added the document to your project and configured it for import from Word. This qualifier just runs the import process for existing documents. e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-import_from_word="path_to.docx" -close
```

-import_from_word_list="datafile.txt"

Imports from Word documents as the qualifier above 'import_from_word' does but allows you to specify multiple documents to import from via a data file. The order of the documents inside the data file is preserved for the import which is necessary if some documents reference Requirements in other documents. The data file should contain a list of documents separated by a new line character. The documents should be specified as filenames and extensions (test.docx) or full absolute paths to the documents.

The documents referenced inside the data file should already exist in the Project and be configured for Word Import. e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-import_from_word_list="datafile.txt" -close
```

-import_from_excel="path_to.xlsx"

Imports from the Excel document according to the settings already specified for the document. Please note you must have already added the document to your project and configured it for import from Excel. This qualifier just runs the import process for existing documents. e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-import_from_excel="path_to.xlsx" -close
```

-import_from_excel_list="datafile.txt"

Imports from Excel documents as the qualifier above 'import_from_excel' does but allows you to specify multiple documents to import from via a data file. The order of the documents inside the data file is preserved for the import which is necessary if some documents reference Requirements in other documents. The data file should contain a list of documents separated by a new line character. The documents should be specified as filenames and extensions (test.xlsx) or full absolute paths to the documents.

The documents referenced inside the data file should already exist in the Project and be configured for Excel Import. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-import_from_excel_list="datafile.txt" -close
```

-add_source_from_tcf="path_to.tcf"

Adds the source contained within the TCF file into the project. Equivalent to Source->Add File-Set from TCF e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-add_source_from_tcf="path_to.tcf" -close
```

-analyse_source

Analyses all the Source currently in the project. Equivalent to invoking the context menu on a Source File or Set and choosing Analyse Procedures. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-analyse_source -close
```

-traceability_matrix_req_report="GroupName,CoveringGroupName"

Generates a Requirement Traceability Matrix Report for the Group "GroupName" that is related (covered) by "CoveringGroupName". An example would be System Level Requirements covered by High Level Requirements "SYS,HLR".

Equivalent to invoking the context menu on a Group and choosing Traceability Matrix. If there are multiple relationships for the group then the selection in the dialog is provided by "CoveringGroupName". e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson" -traceabilit
y_matrix_req_report="SYS,HLR" -close
```

-traceability_matrix_tci_report="GroupName,CoveringGroupName"

Generates a Test Case Traceability Matrix Report for the Group "GroupName" that is related (covered) by "CoveringGroupName". Same as the qualifier above but for Test Cases rather than Requirements. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-traceability_matrix_tci_report="HLR,HLT" -close
```

-proj_coverage_report

Generates a Project Coverage Detailed Report. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-proj_coverage_report -close
```

-proj_coverage_summary_report

Generates a Project Coverage Summary Report. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-proj_coverage_summary_report -close
```

-traceability_report.

Generates a Traceability Summary Report. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-traceability_report -close
```

-objective_report

Generates a Objective Summary Report. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-objective_report -close
```

-user_summary_report

Generates a User Summary Report. e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-user_summary_report -close
```

-defect_report.

Generates a Defect Summary Report e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-defect_report -close
```

-unit_test_regression_report

Generates a TBmanager Unit Test Regression Report

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-unit_test_regression_report -close
```

-source_mapping_report

Generates a Source Mapping Report e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-source_mapping_report -close
```

-source_impact_report

Generates a Source Impact Report e.g.

```
tbmanager.exe <path_to.tbp> -first="Jane" -last="Wilson"
-source_impact_report -close
```

-source_impact_report_filter=<xml_file>

To be used in conjunction with **-source_impact_report** to filter the report to only show the impact of the procedures specified in the xml file. The format of the filter file is as follows:

```
<SourceImpactAnalysisFilter>
  <procedure name="main"/>
  <procedure ds="Float_64 TunnelData::Lamp::GetMaximumLumens();"/>
</SourceImpactAnalysisFilter>
```

Where name is just a procedure name to attempt to match or alternatively supply the ds flag of the procedure from the TBmanager Known Source export.

-source_impact_report_local_changes

This compares all source code in the project with the last analysed version and detects any files that may have changed.

The files are then included on the Source Code Impact Analysis report to display the impact of the local source code edits.

-new_project_baseline="baseline_name"

Creates a new Project Baseline with the specified name. e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-new_project_baseline="baseline_name" -close
```

-project_baseline_report="baseline_name_to_compare_against"

Generates a Project Baseline report comparing the current Project State against the baseline name specified on the command line. e.g.

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-project_baseline_report="baseline_name_to_compare_against" -close
```

-verify_code_review_tcis

Verifies all automatable Code Review TCI's in the project

```
tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_code_review_tcis -close
```

-verify_quality_review_tcis

Verifies all automatable Quality Review TCI's in the project

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_quality_review_tcis -close
```

-verify_code_coverage_tcis

Verifies all automatable Code Coverage TCI's in the project

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_code_coverage_tcis -close
```

-verify_unit_test_tcis

Verifies and Regresses all automatable Unit Test TCI's in the project

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_unti_test_tcis -close
```

-verify_external_task_tcis

Verifies all automatable External Task TCI's in the project

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_external_task_tcis -close
```

-verify_tci_grid

Verifies all automatable TCI's visible in the TCI Grid. This is affected by filters.

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_tci_grid -close
```

-verify_specific_tcis=path/to/file

Specifies a file containing a list of TCIs. The file should be a plain text file containing TCI Numbers on each line

e.g.
TCI_01
TCI_03

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-verify_specific_tcis=<file> -close
```

-apply_view_filter=path/to/file

Applies Filters to views specified in the View Profile. This filter affects -verify_tci_grid

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-apply_view_filter=<file> -close
```

-export_view_as_csv=path/to/file

Exports the views present in the View Profile with any filters applied. (Output into report_storage directory)

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-export_view_as_csv=<file> -close
```

-export_view_as_xml=path/to/file

Exports the views present in the View Profile with any filters applied. (Output into report_storage directory)

```
-tbmanager.exe      <path_to.tbp>      -first="Jane"      -last="Wilson"
-export_view_as_xml=<file> -close
```

Multiple Testing on the Same File

Within *LDRA Testbed*, multiple testing of a single source file would result in one execution history being produced for all of the executions of that file. When producing multiple execution histories from outside of the *LDRA Testbed* environment, the execution history file will only contain the last execution history. The batch file will need to append new executable histories to an arbitrary file that is not overwritten when the source file is being tested repeatedly.

When a new execution history file is produced the contents should be written to a master exh file, named so that it can not conflict with the testing. Every time the instrumented executable is run and a new execution history file is produced, the text from the file should be appended to the master exh file.

So *LDRA Testbed* will know when a new execution history has finished and a new one will start, the termination character

```
-1
```

should be written between each execution history. Please note that if you are using any instrumentation strategy other that standard single point streamed then you need to replace the -1 with the appropriate splitter marker. See the Instrumentation section for further details.

The text in the execution histories will resemble a stream of numbers, -1 will need to be placed in the identical column as these numbers. There should be no termination string after the final execution history has been written to the master exh file. When all execution histories have been collected, the master exh file should be renamed to the original execution history name. Only then can Dynamic Coverage Analysis be performed.

Here is an example of the above methodology with triangle.c as the source:

```
inszt_triangle
```

The instrumented exe is executed

```
copy triangle.exh master_triangle.txt
```

The resulting triangle.exh file is copied to a master text file

```
echo     -1>> master_triangle.txt
```

The termination string -1 is piped to the master text file, exactly 6 characters long.

```
inszt_triangle
```

The instrumented exe is executed for the second time

```
type triangle.exh >> master_triangle.txt
```

The body of the new triangle.exh is appended to the master text file

```
echo     -1>> ch_triangle.exh
```

The termination string -1 is appended to the master text file

```
inszt_triangle
```

The instrumented exe is executed for the last time. No termination string is needed

```
type triangle.exh >> master_triangle.txt
```

The body of the new triangle.exh is appended to the master text file

```
del triangle.exh
```

The new triangle.exh file is deleted

```
copy master_triangle.txt triangle.exh
```

The master text file become the execution history file

```
del master_triangle.txt
```

The master text file is then deleted

```
start %TBwait% %testbed_inst%\testbed triangle.c -32q
```

Dynamic Coverage Analysis is run with the complete execution history file of three test cases

GLH Support

The utility **glh_support.exe** can be used to extract Configuration and Analysis scope data from a GLH.

Usage is in the form:

glh_support.exe file.glh -args

The argument **-support** is used to specify the data files to extract, e.g. **static** or **dynamic**.

To extract **static** data files from a **file.glh**:

glh_support.exe file.glh -support static

To extract **dynamic** data files from a **file.glh**:

glh_support.exe file.glh -support dynamic

Extracting the **dynamic** data files also extracts the files that would be extracted using the **static** argument.

Use the argument **-sys_cod** to also extract the **sys.cod** and **sys.shr** files (These files are for documentation of system file includes).

glh_support.exe file.glh -support dynamic -sys_cod

These data files are extracted to the same directory as the GLH by default. The command line argument **-support_output** can be used to specify the location these data files should be extracted to.

For example, to extract to the directory **C:\my_work**

glh_support.exe file.glh -support dynamic -support_output C:\my_work

To extract to your *Desktop* you can specify **Desktop**

glh_support.exe file.glh -support dynamic -support_output Desktop

Any directories specified must already exist.