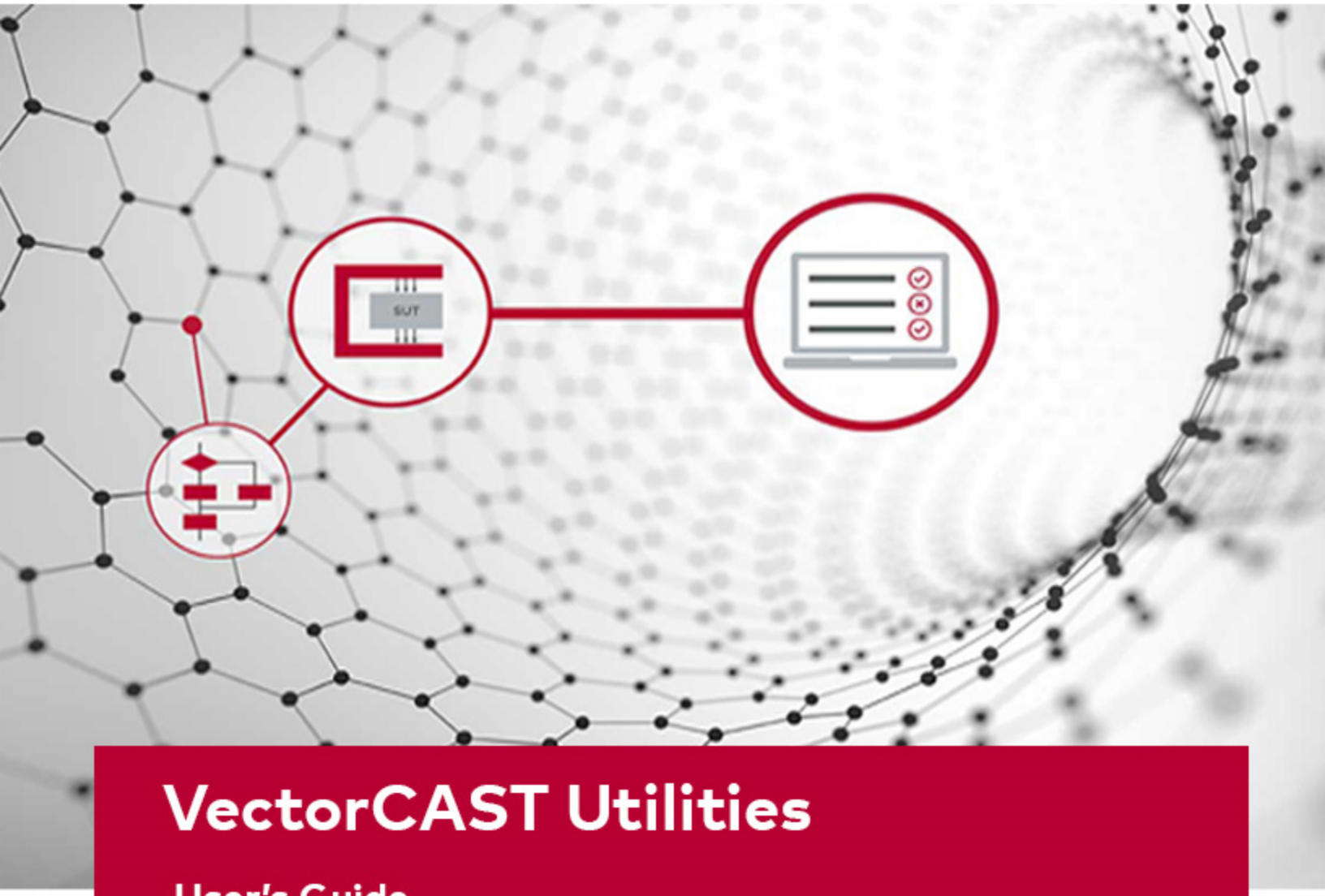
VECTOR



VectorCAST Utilities

User's Guide

VectorCAST 2026

New editions of this guide incorporate all material added or changed since the previous edition. Update packages may be used between editions. The manual printing date changes when a new edition is printed. The contents and format of this manual are subject to change without notice.

Generated: 10/19/2026, 8:29 PM

Rev: 6be4125

Part Number: VectorCAST Utilities User's Guide for VectorCAST 2026 SP5

U.S. Government Restricted Rights

This computer software and related documentation are provided with Restricted Rights. Use, duplication or disclosure by the Government is subject to restrictions as set forth in the governing Rights in Technical Data and Computer Software clause of

DFARS 252.227-7015 (June 1995) and DFARS 227.7202-3(b).

Manufacturer is Vector North America, Inc. East Greenwich RI 02818, USA.

Imprint

Vector reserves the right to modify any information and/or data in this user documentation without notice. Vector disclaims all liabilities for the completeness or correctness of the contents and for damages which may result from the use of this documentation. This documentation nor any of its parts may be reproduced in any form or by any means without the prior written consent of Vector. To the maximum extent permitted under law, all technical data, texts, graphics, images and their design are protected by copyright law, various international treaties and other applicable laws. Any unauthorized use may violate copyright and other applicable laws or regulations.

© Copyright 2026, Vector Informatik, GmbH. All rights reserved.

Protected Trademarks

All brand names in this documentation are either registered or non-registered trademarks of their respective owners.

Third-Party copyright notices are contained in the folder: 3rdPartyLicenses, located in the VectorCAST installation directory.

TABLE OF CONTENTS

Introduction 4

VectorCAST Utilities 5
Tool Components 5

Work Flow Scenarios 6

Create a vcshell Database Using a Script or Makefile 7
Create a vcshell Database Manually Using putcommand 7
Make Substitutions to a vcshell Database 8

vcshell Command Line Reference 10

vcshell Overview 11
vcshell Command 11
vcshell Options 11
Examples Using the vcshell Command 16
putcommand Command 17
putcommand Options 17
vcshell Limitations 18

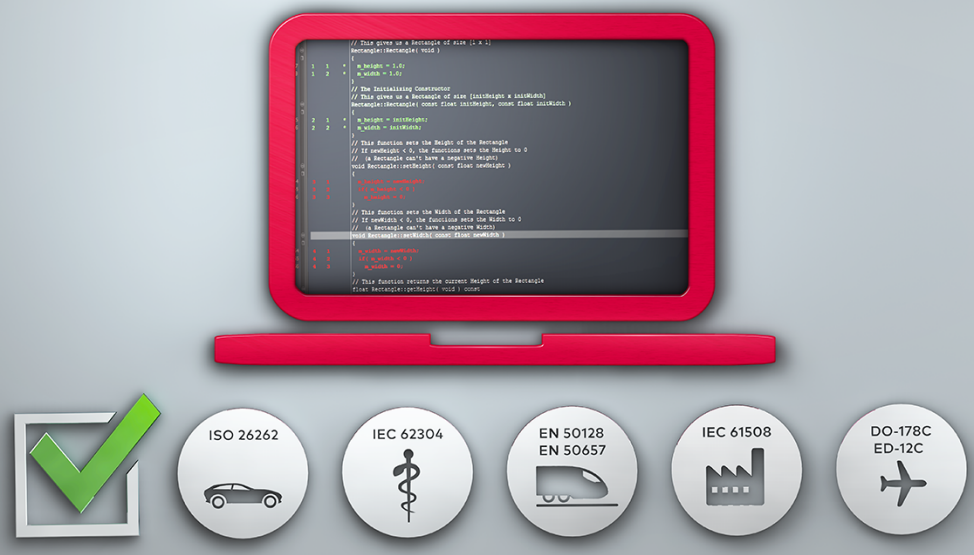
vcdb Command Line Reference 19

vcdb Overview 20
vcdb Command Syntax 20
vcdb Options 20
vcdb Commands 27

vcutil Command Line Reference 38

vcutil Overview 39
vcutil Command Syntax 39
vcutil Options 40
vcutil Commands 45

Index 49



Introduction

VectorCAST Utilities

VectorCAST Utilities integrate with any existing build process to capture information such as filenames, compiler flags, and linker arguments. This information is saved into the vcshell database file (with a default name of **`vcshell.db`**), which the VectorCAST testing and code coverage tools use as a foundation for a variety of advanced features.

The VectorCAST Utilities simplify locating relevant configuration information, providing maximum ease of configuration with minimum effort.

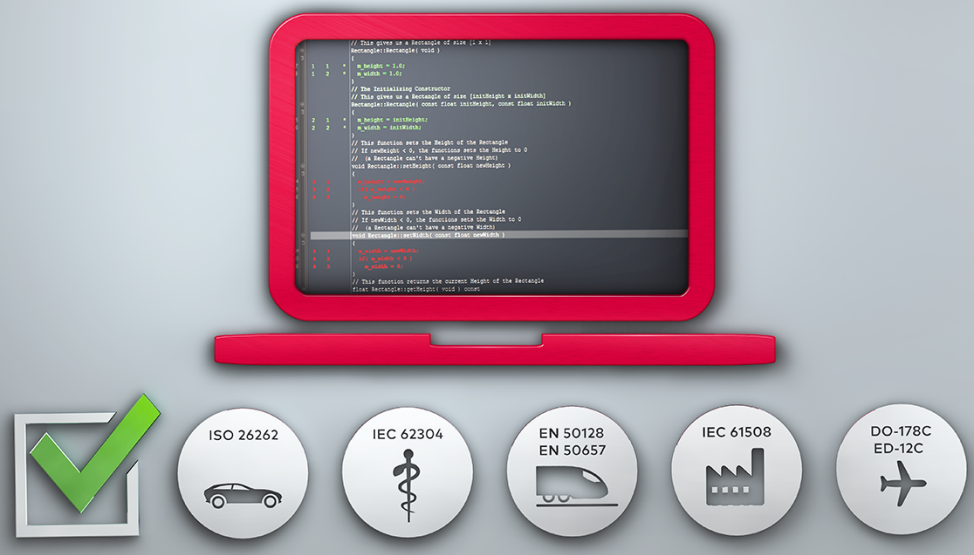
Tool Components

The Utilities are composed of the following components:

- **vcshell** - a wrapper utility that monitors the build process and creates the build settings database (referred to as the vcshell database).
- **vcshell database** - the database file (named **`vcshell.db`**) created by **vcshell** to store the captured build information.
- **vcdb** - a utility that provides command line access to the vcshell database.
- **vcutil** - a utility that allows you to perform common actions to files after the creation of the vcshell database.

A typical workflow for using the Utilities is shown below. The vcshell command line is run in your environment, creating a database of build files (**`vcshell.db`**). VectorCAST tools then connect to the vcshell database to build test environments and instrument code.





Work Flow Scenarios

Create a vcshell Database Using a Script or Makefile

An example build script, **build.bat**, is used to build a C application. The build script compiles the source code and then creates an executable called **testIt**.

```
set BASE=%CD%
set oldpath=%path%
set path=%VECTORCAST_DIR%\MinGW\bin;%path%
del *.o
del *.exe
gcc -c -g -I%BASE%\ss1 -I%BASE%\ss2 %BASE%\ss1\subSystem1.cpp
gcc -c -g -I%BASE%\ss1 -I%BASE%\ss2 %BASE%\ss2\subSystem2.cpp
gcc -c -g -I%BASE%\ss1 -I%BASE%\ss2 %BASE%\Main\callingSequences.cpp
gcc callingSequences.o subSystem1.o subSystem2.o -o testIt
set path=%oldpath%
```

To create a vcshell database, run **vcshell** with the **build.bat** batch file:

```
%VECTORCAST_DIR%\vcshell build.bat
```

When the command completes, you will see a new **vcshell.db** in the directory where you ran the command.

Alternatively, you can supply a makefile and that will be used to create the **vcshell.db** file. For example:

```
%VECTORCAST_DIR%\vcshell mingw32-make generic
```

Create a vcshell Database Manually Using putcommand

The **vcshell.db** file can also be created manually using **putcommand**. Using the example in "Create a vcshell Database Using a Script or Makefile" on page 7 as a reference, here is how you would create the vcshell database using a text file.

For example, you might have captured the output from the build.bat file shown above into a text file called: **build_db.txt**:

Below is an example input command file, **build_db.txt**:

```
dir:: C:\work\vcdb\src
cmd::gcc -c -g -IC:\work\vcdb\src\ss2 C:\work\vcdb\src\ss1\subSystem1.cpp
cmd::gcc -c -g -IC:\work\vcdb\src\ss C:\work\vcdb\src\ss2\subSystem2.cpp
cmd::gcc -c -g -IC:\some\othe C:\work\vcdb\src\Main\callingSequences.cpp
dir:: C:\some\other\dir
cmd::gcc callingSequences.o subSystem1.o subSystem2.o -o testIt
```

The input command file consists of a series of directory entries (where the commands are executed) and command entries. Note that a single directory entry can correspond to multiple commands. In our example above, all the commands are executed in the same directory: **C:\work\vcdb\src**. Larger build environments will contain multiple directory entries and corresponding command entries.

Enter the following command to create a vcshell database:

```
%VECTORCAST_DIR%\vcshell --inputcmds=build_db.txt putcommand
```

The power of this tool comes into play where you may want to alter the commands that are used to build the software. For example, you may want to use additional compiler options or even change the compiler command altogether.

Using the example above, if the compiler used for testing is "**g++**" instead of "**gcc**", the input commands file would look like this:

```
dir:: C:\work\vcdb\src
cmd::g++ -c -g -IC:\work\vcdb\src\ss2 C:\work\vcdb\src\ss1\subSystem1.cpp
cmd::g++ -c -g -IC:\work\vcdb\src\ss1 C:\work\vcdb\src\ss2\subSystem2.cpp
cmd::g++ -c -g -IC:\some\other C:\work\vcdb\src\Main\callingSequences.cpp
dir:: C:\some\other\dir
cmd::g++ callingSequences.o subSystem1.o subSystem2.o -o testIt
```

You can edit the file and then build a vcshell database from the updated file, giving you precise control of the contents of the build settings database.

For more information, see the complementary command *getallcmdlines*, which fetches all of the commands from the database.

Make Substitutions to a vcshell Database

VectorCAST provides a helpful python script which is useful when a vcshell database needs some simple substitutions made. The script is named **vcdb_search_and_replace.py**, and it is located at **$VECTORCAST_DIR/python/vector/apps/vcdb/vcdb_search_and_replace.py**.

For example, it may be desirable to change the flag **-Wp,-isystem,** to **-isystem** in a command in the vcshell database. This can be accomplished using the python script:

```
$VECTORCAST_DIR/vpython $VECTORCAST_DIR/python/vector/apps/vcdb/vcdb_search_and_
replace.py "-Wp,-isystem," "-isystem"
```

By default, the script assumes the vcshell database is named **vcshell.db**, but if not, pass in the argument **--vcdb <vcshell database>**.

The vcshell database is backed up to **<vcshell database>.bak** by default, but if no backup is desired, use the **--no-backup** argument.

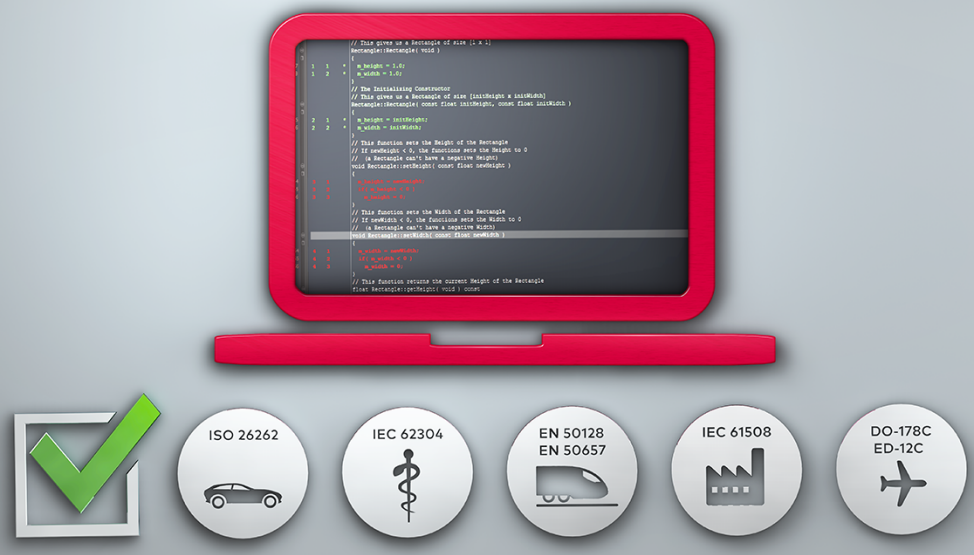
Use **-h** or **--help** to get help.

```
Usage:
  vcdb_search_and_replace.py [-h] [--vcdb <vcdb> =vcshell.db] [--no-backup]
search replace

Positional arguments:
  search     the pattern to be replaced
  replace    the pattern which will replace the old pattern

Options:
  --vcdb <vcdb> (=vcshell.db)  The vcdb to be modified.
  --no-backup   Do not create a backup of the original vcdb.
  -h --help   Show the help message and exit.
```

```
Example:
  vcdb_search_and_replace.py --no-backup "-Wp,-isystem," "-isystem"
```



vcshell Command Line Reference

vcshell Overview

vcshell is a build settings harvester that is used as a wrapper for your normal build process (e.g., **vcshell make**). Each command issued by the build process is intercepted, recorded in the vcshell database (named **vcshell.db**), and then passed to the operating system for execution.

VectorCAST tools then can connect to the vcshell database to enable the fully automated creation of test environments and instrumented applications. Automating the capture of the configuration settings streamlines the automation of tests and greatly reduces testing time.

vcshell Command

vcshell records the commands of the build process by running the actual build command under its control and intercepting the calls to the underlying compiler.

The syntax for **vcshell** is as follows:

```
vcshell [options] build-cmd
OR
vcshell [--db=<dbname>] [--tag=<tag-name>] --inputcmds=file putcommand
```

Where **build-cmd** is the specific command line used to invoke the build process being captured.

With **vcshell**, build commands having command-line arguments can either be quoted or prefixed with '**--**'. Note that **vcshell** treats everything followed by '**--**' as part of the build command. Without the '**--**', **vcshell** continues to look for its recognized command line arguments in the build command.

Examples using '**--**' to specify a command-line argument:

```
vcshell --db=specific.db --tag=custom.tag msbuild vcexample.sln

vcshell --echo make -f vcast.makefile

vcshell --cmd=log.sh make -f vcast.makefile

vcshell --inputcmds=commands.txt putcommand

vcshell "python b.py --db my.db"
```

Examples using '**--**' to specify the build command:
(Note that in the two examples below, the text "**--db**" and "**--build**" are not passed to **vcshell**, but rather are just part of the build command.)

```
vcshell -- python b.py --db my.db

vcshell -- cmake --build foldername
```

vcshell Options

The options for **vcshell** are:

[[--cmd=<aux-cmd>] other-options <user-cmd>]	**<aux-cmd>** specified with **--cmd** is executed before the execution of each of the recorded commands. - If **<user-cmd>** is **gcc**, then **<aux-cmd>** is executed before invoking the **gcc** command. - If **<user-cmd>** is **make**, then **<aux-cmd>** is executed for all the corresponding compilation commands recorded by vcshell. **<aux-cmd>** is expected to be a complete independent shell-level quoted command along with the required options. Special characters in **<aux-cmd>** are to be quoted appropriately. Vcshell executes the command exactly as written. Commands with complex uses of special characters can be put in a script and the script can be provided as the **<aux-cmd>**. A file containing attribute-value pairs is accessible using the command: **VC_PARAM_CMDLINE: <quoted-user-cmd>** **VC_PARAM_PWD: <directory of execution>** **VC_PARAM_SOURCE_FILE: <full filename>** **VC_PARAM_CFG_FILE: <CCAST_.CFG location>** Example (Instrumenting the source code before compilation): **vcshell --cmd="vcinst --coverage_ type=statement" --db=myown.db make all**
[--cmd-verb=<cmdList>]	Comma separated list of the compile/link command verbs to be captured. By default, all commands are captured. **--cmd-verb** can be used to record only specific commands. A string comparison is done between the specified command verbs and the compiler name. Hence, a command such as **vcshell -cmd-verb=/usr/bin/gcc make** may not record any command. Example: **vcshell --cmd-verb=gcc-4.3, g++-4.3 make**
[--db=<dbname>]	Applies a specific name to a database. The default name is **vcshell.db**. It is recommended practice to use the default name for the database. Once named, the reference is required for any transaction. The associated parameter is **--tag**. **Note:** database files are cumulative, and each time the **vcshell** command is run pointing to a database, that database file will increase in size. Example: **vcshell --db=specific.db msbuild project.sln** **vcdb --db=specific.db getfiles**

[--echo]	Captures the build process in text output. Can be used to access the core compilation / link commands (which is especially useful when the build system hides these commands from being emitted). The default output file is **vcecho.out**. The default can be overridden with the **--out** option. **vcshell.db** is also generated by default with **--echo**. Example: **vcshell --echo --out=myown.out msbuild project.sln**
[--expand]	Used to expand the "@file" option in various compilers during vcshell recording. By default, **vcshell** expands files specified with the @prefix. Recursive includes are supported. In cases where the "@" option and the filename are separated by a delimiter, **--expand="-@=1"** should be used. In the case where '=' is used in the command line to specify the file, **--expand=″-@=″** should be used The syntax is **--expand=[cmdVerbList:]<option-list>[=1|=]** For example, in **'gcc @all-options myfile.c'** all options are in '**all-options**'. If the compiler **armcc** has the option **-f** to specify this, then **--expand=iccarm:-f=1** indicates that the option following **-f** is expanded to get the corresponding option. The specification of **--expand=iccarm:-f=** implies the expansion of the option following **-f=**. That is, the = character is considered as part of the specification. Example: **vcshell --expand=iccarm,icclink:-f=1 "iccarm -f myoptions x.c"** **vcshell --expand=iccarm:-f= "iccarm -f=myoptions x.c"** **vcshell --expand=cmdverb-x:optone=1,opttwo=1 "cmdverb-x -optone myoptions x.c"**
[--ext=<specialExts>]	**vcshell** automatically processes all source files with standard extensions (.C, .CPP, .CC, .cpp, .cc, .c++, .cp, .cxx, .tcc, .c). To specify source files with special extensions, use the **--ext** option. Note that the "." must be specified in the extension. Entries corresponding to the SOURCE_EXTENSIONS attribute in CCAST_.CFG are also considered.

	Example: **vcshell --ext=.xyz gcc -xc -c cfile.xyz**
[--inputcmds]	**--inputcmds** is a required option for **putcommand**. It contains a set of directory entries and command entries. The directory entry is prefixed with **dir::** and the command entry with **cmd::.** Each command entry requires a directory entry to precede it and a single directory entry can service multiple command entries. In cygwin, **cygwindir::** also can be provided to specify the location of **cygwin1.dll**. Any **cygwindir** specification is expected to hold for all the commands specified before another **cygwindir** specification. Commands can be gathered from an existing DB through a **getallcmdlines** command. Or they can be gathered from a build command such as: **make -n**. Guidelines to correctly format the command: 1. Command with spaces in Windows should be quoted. For example: **cmd::"c:\Program Files\..\cl.exe" xyz.c** 2. An option containing spaces should be quoted. For example:**'-I/Path with space/example'** 3. While recording cygwin commands, ALL paths SHOULD be windows style paths. Example: **vcshell --inputcmds=xyz.txt putcommand** Contents of **xyz.txt**: **cygwindir::c:\cygwin-17.11\cygwin** **dir::C:\ghs\bin\ccppc** **cmd::gcc -DHAVE_C -I../lib -MT runp.o -MD -c -o runp.o runp.c** **cmd::gcc -c -I\some\path -DMYDEF="STRING" some-name.c**
[--out<filename>]	Specifies the echo text output file. The default is **vcecho.out** located in the build directory. Example: **vcshell --echo --out=vcastoutput.txt --db=vcshell.db make -f vcast.makefile**
[--tag=<tag-name>]	Text string allowing the user to specify a unique identifier for the data collected. The data is recorded in a specific db partition. The **--tag** is equivalent to a logical database. The default tag is **vcshell**. Once tagged, the reference is required for any transaction.

	Example: **vcshell --tag=version-1.0k msbuild project.sln** **vcdb --db=vcshell.db --tag=version-1.0k --file=myfile.cpp getcommand**
[--vcaliases]	Specifies two compilers as equivalent. All the compiler commands associated with one compiler are tagged to both compilers. All commands with either of the compilers in **cmd-verb** will produce the same result. This command is also available in **vcdb**. The syntax is: **vcshell[--db=<dbname>] [--tag=<tagname>] --vcaliases=<aliaslist> make-command** Here, **<aliaslist>** refers to a list of comma separated aliases of the form: **new_compiler=cfg_compiler**. **cfg_compiler** is the compiler specified in CCAST_.CFG Example: **vcshell --vcaliases="cc=gcc,mycc=cfgcc" make**
[--vcbg]	Records asynchronous processes. For example, if the command: **vcshell <options> build-cmd** executes so that **build-cmd** spawns asynchronous compilations and exits, and if **vcshell** has not recorded any of the child compilations, then **vcshell -vcbg <options> build-cmd** records the compilations. Example: **vcshell --vcbg C:\WindRiver\workbench-3.1\wrwb-x86-win32.exe**
[--vcdebug]	An option to generate **vctools** debug info. Log messages are available in **<current-dir>/vcshell.<pid>.log**. A few user level messages are also emitted on **stdout**. Example: **vcshell --vcdebug --db=xyz.db make**
[--vcenv]	An option to specifically manage propagation of **vcshell**-managed environment variables across processes. This is required in recording of commands in some environments such as **bitbake** and **scons**. These environments do not allow/encourage usage of environment variables and clear them before spawning the build command. Example:

	`vcshell --vcenv --db=xyz.db scons`
`[--version]`	Provides the `vcshell` version. Example: `vcshell --version`

Examples Using the vcshell Command

Typical examples of `vcshell` commands are discussed below.

Command	Standalone Compilation Command
Description	Captures commands used in the compilation process.

Examples	`vcshell gcc -c hello.c`
	
	`vcshell cl /c hello.c`
	
	`vcshell "gcc -pthread -fPIC -Wno-unused-result -DNDEBUG=1 -g -fwrapv -O3 -Wall -Wstrict-prototypes -DHAVE_EXPAT_CONFIG_H=1 -DUSE_PYEXPAT_CAPI=1 -I/software/apps/Python-3.3.0/Modules/expat -I./Include -I. -I/usr/include/i386-linux-gnu -I/usr/local/include -I/software/apps/Python-3.3.0/Include -I/software/apps/Python-3.3.0 -c /software/apps/Python-3.3.0/Modules/expat/xmlparse.c -o build/temp.linux-i686-3.3/software/apps/Python-3.3.0/Modules/expat/xmlparse.o"`

Command	Standalone Shell Command
Description	A shell command is executed using the command interpreter `cmd` or a shell. This may be required when executing stand-alone commands having special characters.

Examples	`vcshell cmd /c cl *.c`
	
	`vcshell sh -c "gcc -pthread -c -Wno-unusedresult -DNDEBUG=1 -g -fwrapv -O3 -Wall -Wstrict-prototypes -I. -I./Include -DPy_BUILD_CORE=1 -o Python/pythonrun.o Python/pythonrun.c"`

Command	Linux make Command
Description	Captures commands used in the Linux compilation process, including internal shell commands (`sh gcc hello.c`), commands used in the creation and manipulation of temporary files (`cp XXX YYY`), and compilation commands (`cc1 hello.c, as hello.s, ld hello.o`).
Examples	`vcshell make`

Command	Windows msbuild Command
Description	Captures commands used in the `msbuild` build process.

Examples	**vcshell msbuild hello.sln**

Command	Windows devenv Command
Description	Captures commands used in the **devenv** build process.
Examples	**vcshell devenv any.sln**

Command	Build Inside Eclipse Command
Description	Compilation commands under an Eclipse infrastructure are captured just like other build-commands.
Examples	**vcshell eclipsec.exe -nosplash -application org.eclipse.cdt.managedbuilder.core.headlessbuild -data c:\my_wsp -build k64f**

Command	Echo Command
Description	Captures commands in text form. Both text and database outputs are created with the **--echo** command. The default output file is **vcecho.out**. Option **--out** can be used to specify custom output file.
Examples	**vcshell --echo make -f custom.makefile** **vcshell --echo -out=myoutput.txt msbuild myapp.sln** **vcshell -echo --out=rawfile.txt -db=dbfile-also.db make -f custom.makefile** **vcshell -echo -db=custom.db devenv xyz.sln**

putcommand Command

The **putcommand** command is used to populate the database with a command in text form. The syntax for **putcommand** is as follows:

```
vcshell [--db=<dbname>] [--tag=<tag-name>] --inputcmds=<filename>
putcommand
```

putcommand Options

--inputcmds is a required option for **putcommand** that contains command and directory entries. The execution directory is specified first with the prefix **dir::** and the command is specified with the prefix **cmd::**. A directory entry must precede each command. One directory entry can be used for multiple commands executed in the directory. These commands are expected to be gathered from the output of a build command. For more information, please refer to the complementary command *getallcmdlines*, which fetches all of the commands from the database.

For example:

```
dir:: C:\work\vcdb\src
cmd::gcc -c -g -IC:\work\vcdb\src\ss2 C:\work\vcdb\src\ss1\subSystem1.cpp
```

```
cmd::gcc -c -g -IC:\work\vcdb\src\ss1 C:\work\vcdb\src\ss2\subSystem2.cpp
cmd::gcc -c -g -IC:\work\some\dir C:\work\vcdb\src\Main\callingSequences.cpp
dir::c:\path\to\compilation\directory
cmd::gcc callingSequences.o subSystem1.o subSystem2.o -o testIt
```

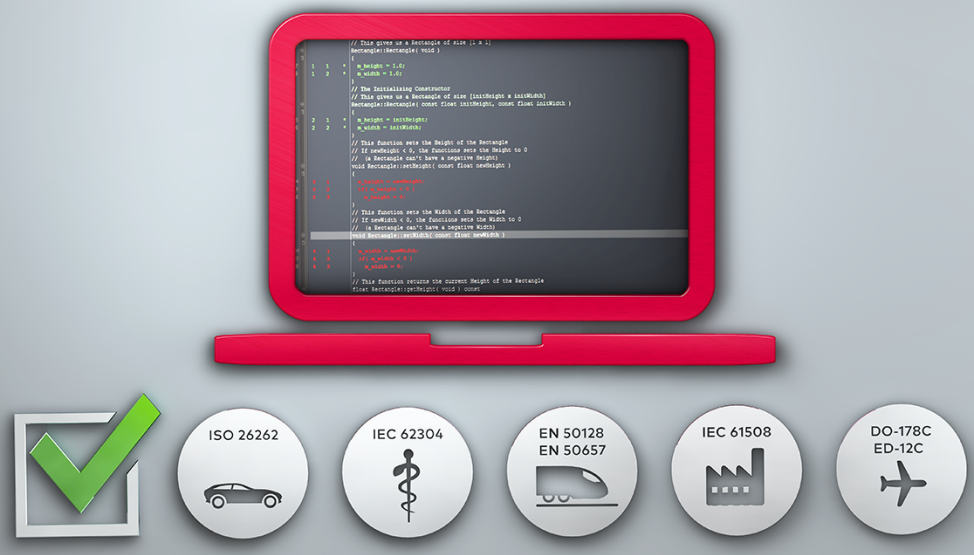
Some guidelines to correctly format the command:

- Command with spaces in Windows should be quoted
  Example: **`cmd::"c:\Program Files\..\cl.exe" xyz.c`**
- An option containing spaces should be quoted
  Example: **`'-I/Path with space/example'`**

Once populated, any **`vcdb`** commands can be used to access the command and its components.

vcshell Limitations

- Only VectorCAST/C, C++ and VectorCAST/Cover are supported by **vcshell**. VectorCAST/Ada is not supported.
- Only Windows and Linux are supported by **vcshell**. Solaris is not supported.
- **vcshell** records only the sub-processes of a build command.
  If the build command sends the compilation commands to a server and the server executes the compilation commands, **vcshell** will not be able to record the compilation commands.
- **vcshell** records commands only.
  vcshell cannot record internal functions of command interpreter or shells. For example, bash functions are not recorded.
- All cygwin-related path specifications must be specified as Windows paths.
- Build commands containing special characters may lose some of those characters when the command interpreter passes them to **vcshell**.
  Workaround:
  Put the command in a batch file or shell script and pass the batch file or shell script to **vcshell**.
- "**`build`**" as a build command for **`build.bat/build.cmd`** is not supported.
  Workaround:
  `%VECTORCAST_DIR%\vcshell cmd /c build`
  `%VECTORCAST_DIR%\vcshell build.bat`
- "**`build`**" as a command for **`build.bat`** in PATH is not supported.
  Workaround:
  `%VECTORCAST_DIR%\vcshell cmd /c build`
  `%VECTORCAST_DIR%\vcshell C:Abspath\of\build.bat`



vcdb Command Line Reference

vcdb Overview

The **vcdb** utility provides an interface to the data stored in the vcshell database. For example, **vcdb** commands and associated options allow the user to query the database and retrieve lists of processed files, lists of include paths, or the full compile command for a file.

vcdb Command Syntax

The **vcdb** commands provide an interface to the data stored in the vcshell database and are used to retrieve commands and options recorded by **vcshell**. The **vcdb** tool requires the CCAST_.CFG file be present in the current working directory. The CCAST_CFG attributes are used to parse the commands. Specific user options can be used to override the attributes. The syntax for **vcdb** is as follows:

```
vcdb [options] vcdb-command
```

Some examples are:

```
vcdb --db=specific.db getfiles
```

```
vcdb --includes="-I/newpath/spec, -xyz abc" --file=xyz.c getincludes
```

```
vcdb --cmd-verb="mycomp" --outputflag="-o" --cflag="-c" getapps
```

vcdb Options

[--abspath]	Used to change relative paths to absolute paths. If **--abspath** is not used, then the returned paths are as given in the command line. The associated option **--needabspath** can be used as required when path references of an option are not changed. Example: **vcdb --file=xyz.c --abspath --needabspath="-MF" getcommand**
[--aliases]	Used along with **setaliases** to replace the **cfg_compiler** with a new compiler. New alias can be passed using **cmd-verb** with vcdb and vcutil commands. Example: **vcdb --aliases="cc=gcc" setaliases**
[--app]	Used in **getfiles** to retrieve all source files that are required to build the specified application. For example, if **a.c, b.c, c.c** and **d.c** are used to build app **a.exe**, then **vcdb --app=a.exe getfiles** will retrieve **a.c, b.c, c.c** and **d.c.** If **--app** is not specified, then **vcdb** will retrieve the pathnames of all the source files for a tag from database. Either the full path or just the app name can be specified.

	Example: **vcdb --app=/path/to/file/xyz.exe getfiles**
[--appo]	Add specific pre-processor options to the **getppoptions** output. These options are added verbatim to the output. The complementary option **--oppo** is used to omit specified options in the output. Example: **vcdb --appo="-DVCAST_BUILD -std=gnu++0x" --file=xyz.c getppoptions**
[--cfg]	Specifies the location of the CCAST_.CFG file. Example: **vcdb --cfg="/path/to/mycfgs" getfiles**
[--cflag]	Compile only flag. Used in **getapps** and **getappfiles** to filter out or select compile-only commands. Example: **vcdb --cmd-verb="mycomp" --cflag="-c" getapps**
[--clicast]	Used when CFG attributes are accessed via Clicast. The CFG interface is by default a file interface. Example: **vcdb --clicast --db=any.db getfiles**
[--cmd]	Used to add custom options to the **getcommand** output. Any command name should be the first entry in the string. Example: **vcdb --file=xyz.c --cmd="command -DNEW=55 -DOLD=99" getcommand**
[--cmddir]	Retrieves the build directory of a file along with other data. If used with **getcommand**, the compilation directory is returned in a separate line after the command. Example: **vcdb --cmddir --file=xyz.c getcommand**
[--cmd-verb]	Used to specify commands of interest. **--cmd-verb** only retrieves specified commands using a string comparison between the specified **cmd-verbs** and the database entry, and CCAST_.CFG is not used. If **--cmd-verb** is not specified, **vcdb** queries CCAST_.CFG for C_COMPILE_CMD and ALT_C_COMPILE_CMD to get the

	compilation command verbs. Example: **vcdb --cmd-verb=custom-gcc,specific-g++ getfiles**
[--db=<dbname>]	Applies a specific name to a database. The default is **vcshell.db**. Once named, the reference is required for any transaction. The associated parameter is **--tag**. Example: **vcdb --db=specific.db getfiles**
[--defines]	Used in **getdefines** to add new entries and change existing entries. In the example below, a new option **-DNEW=55** is added and an existing **-DOLD=NN** is changed to **-DOLD=99**. Example: **vcdb --file=xyz.c --defines=" -DNEW=55 -DOLD=99 " getdefines**
[--delimiter]	By default, **vcdb** handles comma-separated lists. The option **--delimiter** is used to specify any delimiter other than comma. Example: **vcdb --file=hello.c --delimiter=; getincludes**
[--ext]	**vcdb** automatically finds all source files with standard extensions (.C, .CPP, .CC, .cpp, .cc, .c++, .cp, .cxx, .tcc, .c). To specify source files with special extensions, use the **--ext** option. Note that the "." must be specified in the extension. Entries corresponding to the SOURCE_EXTENSIONS attribute in CCAST_.CFG are also considered. Example: **vcdb --ext=.newext getappfiles**
[--file]	Specify a file for which command line options are required. Mandatory in all **vcdb** commands where attributes of the specific file are required. Can be either a bare filename (**foo.c**) or a full path to a file (**/home/abc/source/foo.c**). If multiple entries are found in the database, the full path must be specified. Example: **vcdb --file=\path\to\file\xyz.c getincludes**
[--filter]	This option is used with **dumpcommands** to get only the commands associated with the given string. Note that all commands containing the filter string are emitted. If word search is desired, a

	space can be used at the end of the word. String can be a **cmdverb** or a word in the command line. Examples: **vcdb --filter=cl.exe dumpcommands** **vcdb --filter="make " dumpcommands** **vcdb --filter="/subdir/" dumpcommands**
[--flags]	Used as a mask in **getoptions** to get specific options from the command line. Options with parameters (**-o <filename>**, for example) should be specified with the **=1** suffix. Also, if multiple flags match a specific option, the longest match is considered. Consider this example: **vcdb --file=xyz.c --flags="-c=1,-cpu=1,-g" getoption**. Here, both **-c=1** and **-cpu=1** will match the option '**-cpu arm**'. With matching of **-c=1**, the output would be **-cpu**, and with matching of **-cpu=1**, the output would be '**-cpu arm**'. Since **-cpu** is longer than **-c**, **-cpu=1** will be matched and the **getoption** output would contain: '**-cpu arm**'. Example: **vcdb --file=xyz.c --flags="-I=1,-D=1,-c=1,-cpu=1,-g" getoption**
[--include_env_var]	Used to specify environment variables that hold the include directories. For example, - CPATH, C_INCLUDE_PATH, OBJC_ INCLUDE_PATH, etc., that are used by GNU tool chains. The include directories specified by these environment variables during vcshell database creation will be appended to the vcdb output. Examples: **vcdb --file=anyfilename.cpp --include_env_ var=CPATH getincludes** **vcdb --file=anyfilename.cpp --include_env_var=C_ INCLUDE_PATH getcommand**
[--includes]	Add new include paths in the output of **getincludes**. The provided string is added to the list of includes in the command line and the values in **--includes** are emitted at the END. Example: **vcdb --includes="-I/newpath/spec, -xyz abc" --file=xyz.c getincludes**
[--libext]	Specify library extensions of interest in **getappfiles**. Note that the "." must be specified in the extension. Recognized extensions are: **.so, .dll, .lib, .a**.

	Example: **vcdb --libext=.newext getappfiles**
[--libraries]	Can be used in **vcdb** commands **getlinkcmd** and **getallcmdlines**. This option can be used to retrieve all the link commands, including the ones creating libraries. Without this option, only the link commands creating applications are displayed. Example: **vcdb --libraries getlinkcmd**
[--needabspath]	Associated with **--abspath** option. Used when path references of an option are not emitted as an absolute path. For example, in the command: **gcc -XYZ subdir -c xyz.c** even if **--abspath** is specified, **subdir** may not be expanded in the output. The following command ensures that full path is emitted. (Note that **=1** suffix is not needed here.) **vcdb --file=xyz.c --abspath --needabspath="-XYZ" getcommand**
[--nocache]	Forces **vcdb getpaths** to compute the includes and not use the cached data. Can be used in **vcdb getpaths**. Example: **vcdb --nocache getpaths**
[--oext]	Used to specify object extensions. Default object extensions such as **.o** or **.obj** are automatically used to parse command lines. The option can be used to specifiy custom object extensions. Example: **vcdb --oext=".XYZ" --app=APPNAME getfiles**
[--omit]	Used in **getcommand** to omit the specified options in the output. Any option can be specified to be omitted. **=1** suffix specifies that a parameter associated to the option also is omitted. In the example below: **cmd: gcc -DX=55 -I. -c -MD -MF opt.x -O3 -o x.exe xyz.c** The emitted output for the command **vcdb --file=xyz.c --omit="-MD,-MF=1,-o=1,-O=1" --abspath getcommand** is: **gcc -DX=55 -I/cur/dir -c xyz.c**
[--oppo]	Used to omit specific options from the emitted pre-processor options. =1 suffix specifies that a parameter associated with the option is also omitted.

	The complementary option **--appo** is used to add specific pre-processor options to the **getppoptions** output. Example: **vcdb --oppo="-omit-this-and-param=1" --file=hello.c getppoptions**
[--optionprefix]	Normally, **getdefines** and **getincludes** commands return the options without any prefix. By using this option, the define prefix (for example, **-D**) and include prefix (for example, **-I**) is added to each entry in the output includes. An output of the type: **STR1=″AnyStr″** would be output as: **-DSTR1=″AnyStr″**. Example: **vcdb --optionprefix --file=\path\to\file\xyz.c getincludes**
[--outputflag]	Output specification flag used in **getapps** to select custom options that specify target destination. Normally, **-o** is used as target destination. Using the option, custom target options can be specified. These are needed to get the corresponding application names. Example: **vcdb --cmd-verb="mycomp" --outputflag="-appname" --cflag="-c" getapps**
[--ppcmd]	Used with **getppoptions** to specify the pre-processor command. In **gcc**, **-E** is the option to pre-process a file. This command is used to specify custom pre-processor option. This can be used to reconstruct a pre-processor option. Example: **vcdb --ppcmd="-E" --file=xyz.c getppoptions**
[--prefix]	Used to port a database from one location to another. For example, if the directory where **vcshell** is invoked is **/old-machine/existing/path,** that is recorded in the environment variable PWD, and **--prefix=/new-machine/my/new/path** is used in ANY command. ALL prefixes of "**/old-machine/existing/path**" are changed to "**/new-machine/my/new/path**" in any file specification. Related commands **list-prefixes** and **rebase** can be used to review existing prefixes. Example: **vcdb --prefix=/my/new/path getfiles** **Note**: Out of source build is not currently supported. For example,

	if the build is happening in directory **dir1** and **vcshell** is invoked in a separate directory, then the porting of the corresponding db will not be possible.
[--prefix-var]	Used to port a database from one location to another. For example, **--prefix-var=BUILD_DIR** will convert all prefixes with the recorded value of BUILD_DIR to the new value of the environment variable BUILD_DIR. Example: **vcdb --prefix-var=BUILD_DIR --file=xyz.c getcommand**
[--strd]	Used to override the C_DEFINE_FLAG value in CCAST_.CFG. Used only when **--cmd-verb** is specified. Example: **vcdb --cmd-verb="gcc" --strd="-D" --file=xyz.c getdefines**
[--stri]	Used to override the C_INCLUDE_FLAG value in CCAST_.CFG. Used only when **--cmd-verb** is specified. Example: **vcdb --cmd-verb="gcc" --stri="-I" --file=xyz.c getincludes**
[--tag=<tag-name>]	Text string allowing the user to specify a unique identifier for the data collected. The data is recorded in a specific db partition. The **--tag** is equivalent to a logical database. The default tag is **vcshell**. Once tagged, the reference is required for any transaction. Example: **vcshell --tag=version-1.0k msbuild project.sln** **vcdb --db=vcshell.db --tag=version-1.0k --file=myfile.cpp getcommand**.
[--vcdebug]	An option to generate vctools debug info. Debug info is emitted in **stdout**. Example: **vcdb --vcdebug --db=xyz.db getfiles**
[--vcpathstyle]	Passes unix style paths in Windows. With this option, unix style paths ('/' instead of '\') are supported in Windows. This is especially useful when a Linux database needs to be used in Windows and an input path must be specified. Examples: **vcdb --vcpathstyle="unix" --file=/unix/file.c**

	getdefines **vcdb --vcpathstyle="UNIX" --app=/xyz/name getlinkcmd**
[--version]	Provides the **vcdb** version. Example: **vcdb --version**

vcdb Commands

Usage of **vcdb** is:
vcdb --<OPTIONS> vcdb-command.
The **vcdb** commands are discussed below.

Dump Commands

Command	**dumpcommands**
Description	A command to get the list of command entries in the database.
Syntax	**vcdb [--db=<dbname>] [--tag=<tagname>] [--filter=<string>] dumpcommands**
Usage	**--db** and **--tag** can be used to point to a specific database.
Details	Useful when database accesses are not successful. The entries in the database can help to recognize the build context. For example, the compiler that is used may be **cc** and the C_COMPILE_CMD in CCAST_.CFG may be **gcc**. In this case, **vcdb getfiles** may not return any output. Using **dumpcommands**, we can see that **cc** commands are recorded and use **vcdb -cmd-verb=cc getfiles**.
Examples	**$VECTORCAST_DIR/vcdb dumpcommands** **$VECTORCAST_DIR/vcdb --filter=cl.exe dumpcommands**

Dump Verbs

Command	**dumpverbs**
Description	A command to get the list of command verbs registered in the database.
Syntax	**vcdb [--db=<dbname>] [--tag=<tagname>]dumpverbs**
Usage	**--db** and **--tag** can be used to point to a specific database.
Details	Useful when database accesses are not successful and **--cmd-verb** is to be used.
Examples	**$VECTORCAST_DIR/vcdb dumpverbs**

Get All Command Lines

Command	**getallcmdlines**
Description	A command to get ALL command lines from the database corresponding to specific command verb in CCAST_.CFG.
Syntax	**vcdb [--cmd-verb=<cmd-list>] [--db=<dbname>] [--tag=<tagname>] [--libraries] [--abspath] [--needabspath] [--multiple] [--all] getallcmdlines**
Usage	**--all** is used to get ALL commands in the database without checking for command verb. **--cmd-verb** is used to extract specific commands other than the verb in CCAST.CFG. **--needabspath** is used to expand paths for special options. **--multiple** is used to list all compile commands. This is helpful if same source file is compiled with different options. Without **--multiple**, only the last command is returned. **--libraries** is used to retrieve all the link commands including the ones creating libraries. **--abspath** is used to convert emitted paths to absolute paths.
Details	This is a converse of **putcommand**.The emitted output with **--abspath** can be used as an input to **putcommand**. This can be used in AutomationController.
Examples	**vcdb --db=any.db getallcmdlines --abspath** **vcdb --db=any.db --needabspath="-MF" getallcmdlines**

Get App Files

Command	**getappfiles**
Description	Used to retrieve a listing of all the files that make up a specific application.
Syntax	**vcdb [--app=<appname>] [--cfg=<cfgdir>] [--cmd-verb=<cmdlist>][cflag=<compileOnlyFlags>] [--db=<dbname>][--ext=<specialExts>] [--libext=<libraryExtensions>][--oext=<objExt> [--prefix-var="EnvironVar"] [--prefix=<prefix-path> [--tag=<tagname> getappfiles**
Usage	**--app**: Used to get appfiles for a specific app. By default, lists files of all the apps in the database. If **--app** is not used, files for all the apps are displayed. **--cflag**: Flags used here are used to select compile commands from which sources are gathered. **--cmd-verb**: Used to provide both compile and link commands. **--ext**: Used to specify special source file extensions. **--libext**: Used to specify special library extensions. **--oext**: Used to specify object extensions. (Ex:.obj) when **--app** is used.
Details	Object files of the app are noted using the **oext(C_OBJECT_EXT)** value.

	Associated source files are obtained using the: **cflag(C_COMPILE_CMD_FLAG)** to identify the compilation commands, **ext(VCAST_C_FILE_EXTENSIONS)** to identify any special source files, and the outputflag value (**C_OUTPUT_FLAG**) to identify the linker command.
Examples	**vcdb --app=myapp getappfiles** **vcdb --oext=".XYZ" --app=APPNAME getappfiles**

Get Apps

Command	**getapps**
Description	Used to list all the applications recorded in the database.
Syntax	**vcdb [--cflag=<compileOnlyFlags>][--cfg=<cfgDir>][--cmd-verb=<cmdList>] [--db=<dbname>][--oext=<objExt>][--outputflag=<optionList>][--prefix-var="EnvironVar"][--prefix=<prefixPath>][--tag=<tagname>] getapps**
Usage	**--cflag**: Used to specify compile only flags (Ex: **-c**) to recognize appropriate link commands. **--cmd-verb**: This option can be used to provide the linker command. **--oext**: Used to specify object extensions (Ex: **.obj,.o**) that are not to be treated as apps. **--outputflag**: Used to specify option for target destination (Ex: **-o**).
Details	Apps are identified as targets of link commands. Link commands are obtained via attributes or user options: **cmd-verb(C_COMPILE_CMD,C_LINK_CMD)** to recognize linker command, **cflag(C_COMPILE_CMD_FLAG)** to leave out compilation commands, **oext(C_OBJECT_EXT)** to leave out special execs, and **output_flag(C_OUTPUT_FLAG)** to recognize the targets. Outputs of all link commands are treated as apps.
Examples	**vcdb --cmd-verb="mycomp" --outputflag="-o" --cflag="-c" getapps**

Get Command Directory

Command	**getcmddir**
Description	Used to retrieve the build directory of a file.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] --file=<filename> getcmddir**
Usage	**--db**: Used to specify any custom database. The default is vcshell.db. **--file**: A required option to specify the file
Details	Can be used with other vcdb commands to get the command directory.

Examples	`vcdb --db=myown.db --file=myfile.cpp getcmddir`

Get Command

Command	`getcommand`
Description	Command to retrieve the full command of a specific file.
Syntax	`vcdb [--abspath][--cfg=<cfgdir>] [--cmd=<additions>][--cmddir] [--delimiter=<delimiter>] [--cmd-verb=<cmdList>] [--db=<dbname>][--needabspath=<optionList>][--omit=<omitList>][--prefix=<path>][--prefix-var="EnvironVar"][--tag=<tag-name>] --file=<filepath> getcommand`
Usage	`--cmd`: Used to add options to the command line. `cmd-name NEW-OPTIONS` can be used to add NEW-OPTIONS to the command line. `cmd-name` of the spec is ignored. `--cmddir`: When used, the directory where command is invoked is emitted. `--cmd-verb`: A set of verbs (Ex: `gcc,g++,cl`) to select a specific command. `--delimiter:` Specify any delimiter other than comma. `--omit`: Can be used to omit specific options from the output.

Examples	`vcdb --file=myfile.cpp --abspath getcommand`
	
	`vcdb --file=xyz.c --cmd="command -DNEW=55 -DOLD=99" --omit="-MD,-MF=1,-o=1,-O=1" getcommand`

Get Defines

Command	`getdefines`
Description	Lists ALL the define flags used in the command line of a file.
Syntax	`vcdb [--version] [--db=<dbname>] [--tag=<tag-name>] [--cfg=dir] [--delimiter=<delimiter>] [--cmd-verb="cmd1, cmd2, ..."] [--defines=defines-list] [--abspaths] [--optionprefix] --file=<file> getdefines`
Usage	`--cfg`: Directory from where CCAST_.CFG is to be read `--cmddir`: Enables emission of the directory of build cmd invocation. `--cmd-verb`: A set of verbs (Ex: `gcc,g++,cl`) to select a specific command. `--defines`: Used to add new entries as well as to change existing entries. Default definitions provided as part of `--defines` are appended at the end. `--delimiter`: Specify any delimiter other than comma. `--optionprefix`: Used to include the prefix(-D) in the output. By default, the prefix itself is not emitted in the output. `--tag`: If `--tag` is not used and if there is only one record for the file, the corresponding value is returned. If there are multiple records with the same file, the `--tag` option is mandatory.
Details	Uses the C_DEFINE_FLAG of CCAST_.CFG to determine the prefix.

	Returns a string which can be inserted directly into a command (for example, `-Done -Dtwo -Dthree=3`).
Examples	`vcdb --file=xyz.c --defines=" -DNEW=55 -DOLD=99 " getdefines`

Get Destination Dir

Command	`getdestinationdir`
Description	In `vcutil parse` and `instrument` commands, the option `--destination_dir` is used to specify the directory to contain the output files. The command `getdestinationdir` emits the destination directory that was recently used for `vcutil parse --destination_dir=<custom artifacts dir>` or `vcutil instrument --destination_dir=<custom artifacts dir>`.
Syntax	`vcdb --app=<instrument | parse> [--db=<dbname>] [--tag=<tagname>] getdestinationdir`
Usage	`--app` is a required option. It takes one of two values: `parse` or `instrument`.
Details	
Examples	`vcdb --app=instrument getdestinationdir`

Get Filename

Command	`getfilename`
Description	Used to get the full filename of a specific file.
Syntax	`vcdb [--cfg=<cfgdir>] [--db=<dbname>] [--tag=<tag-name>] --file=<filepath> getfilename`
Usage	`--file` is a required option.
Details	Can be used in scripts for querying the database.
Examples	`vcdb --file=xyz.c getfilename`

Get List of Files

Command	`getfiles`
Description	Lists ALL of the recorded files on a database.
Syntax	`vcdb [--app=<appname>] [--cfg=<cfgdir>] [--cflag=<compileOnlyFlags>] [--cmd-verb=<cmdList>] [--db=<dbname>] [--oext=<objExt>] [--outputflag=<optionList>] [--prefix=<path>] [--prefix-var="EnvironVar"] [--tag=<tag-name>] getfiles`
Usage	`--app`: Used to provide an appname to get files for the app. `--ext`: Used to specify special source file extensions.

Examples	`vcdb --oext=".XYZ" --app=APPNAME getfiles`

Get Include Paths

Command	`getincludes`
Description	Lists all include paths used in the command line of a file.
Syntax	`vcdb [--abspath][--cfg=<cfgdir>][--cmd-verb=<cmdList>] [--delimiter=<delimiter>] [--db=<dbname>][--includes="includeList"][--optionprefix][--prefix=<path>] [--prefix-var="EnvironVar"][--tag=<tag-name>] --file=<filepath> getincludes`
Usage	`--file` is a required option. `--abspath`: This option converts emitted paths to absolute paths. `--cfg`: Directory from where CCAST_.CFG is to be read. `--cmd-verb`: A set of verbs (Ex:gcc,g++,cl) to select a specific command. `--includes`: String provided with this option is added to the command line includes and emitted. `--delimiter`: Specify any delimiter other than comma. `--optionprefix`: Can be used to include the prefix(Ex:-I) in the output.
Details	Uses the C_INCLUDE_FLAG of CCAST_.CFG to determine the prefix.
Examples	`vcdb --includes="-I/newpath/spec, -xyz abc" --file=xyz.c getincludes`

Get Link Command

Command	`getlinkcmd`
Description	Used to retrieve all the link commands used in a build.
Syntax	`vcdb [--abspath][--app=<appname>][--cfg=<cfgdir>][--cflag=<compileOnlyFlag][--cmd-verb=<cmdList>] [--db=<dbname>][--needabspath][--outputflag=<optionList>][--oext=<objExt>][--prefix=<path>][--libraries][--prefix var="EnvironVar"][--tag=<tag-name>] getlinkcmd`
Usage	`--cflag`: Flags used here are used to eliminate compile commands. `--cmd-verb`: A set of verbs (Ex: `gcc,ld,link`) to select specific commands. `--oext`: Used to specify object extensions (Ex: `.obj`). `--app`: When `--app` is used, link commands of the app are emitted.
Details	Uses the following user-options / attributes: `cmd-verb` (C_LINK_CMD) to identify link commands, `outputflag` (C_OUTPUT_FLAG) to identify the app and `cflag` (C_COMPILE_CMD_FLAG) to eliminate compile commands
Examples	`vcdb --abspath --app=a.exe getlinkcmd`

Get Obj Path

Command	**getobjpath**
Description	Retrieves the object file path of a given file. By default, **<current-dir>/basename.ext** is the object file path. The related options (-o, for example) are considered to get the user defined obj paths.
Syntax	**vcdb [--db=<dbname>][--tag=<tagname>] [--cmd-verb=<cmdverb>] [--oext=<obj-ext>] [--output_flag=<flag>] [--file=<fpath> | --filelist=<fpath>] getobjpath**
Usage	**--file**: Used to get object file path for a given file. **--cmd-verb**: Used to extract specific commands other than the verb in CCAST.CFG.
Details	C_OUTPUT_FLAG entry in CCAST_.CFG is used as needed. **cmd-verb**, **oext** and **output_flag** can be provided to the get the obj path without CCAST_.CFG. Note that existence of the object file is not verified. Vcdb only derives the standard obj path from the command line.
Examples	**vcdb --file=fast-dtoa.cc getobjpath** **vcdb --file=x.c --oext=".OBJ" --output_flag="-o" --cmd-verb=xcc getobjpath**

Get Option

Command	**getoption**
Description	Lists all the options corresponding to a set of flags provided in the **--flags** option.
Syntax	**vcdb [--abspath][--cfg=<cfgdir>][--cmd-verb=<cmdList>] [--db=<dbname>][--needabspath][--prefix=<path>][--prefix-var="EnvironVar"][--delimiter=<delimiter>][--tag=<tag-name>] --file=<filepath> --flags=<optionList> getoption**
Usage	**--cmd-verb**: A set of verbs (Ex: **ld, link**) to select specific commands. **--file** is a required option. **--delimiter**: Specify any delimiter other than comma. **--flags** is a required option.
Examples	**vcdb --file=xyz.c --flags="-I=1,-D=1,-g" getoption**

Get Paths

Command	**getpaths**
Description	Displays include paths in the corresponding command line with the type

	when used with **--file**.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] [--cfg=cfgdir] [--file=filepath] [--nocache] getpaths**
Usage	**--file**: When **--file** is not specified, all the unique include paths in the database are emitted along with the type. **--nocache**: When specified, includes are computed and not taken from cache data.
Details	The order of includes is the same as that in the build command. The list is alphabetically sorted.
Examples	**vcdb -file=xyz.c getpaths** **vcdb getpaths**

Get Path Type

Command	**getpathtype**
Description	Retrieves the include type for a path.
Usage	**vcdb [--db=<dbname>] [--tag=<tag-name>] getpathtype path**
Examples	**vcdb getpathtype /an/example/path**

Get Pre-processor Options

Command	**getppoptions**
Description	Derives the pre-processor option from the command line.
Syntax	**vcdb [--abspath][--appo=<additions>][--cfg=<cfgdir>] [--db=<dbname>][--delimiter=<delimiter>][--needabspath=<optionList>][--oppo=<omissions>][--ppcmd=<ppcmd>][--prefix=<path>][--prefix-var="EnvironVar"][--tag=<tag-name>] --file=<filepath> getppoptions**
Usage	**--file**: is a required option. **--abspath**: Converts emitted paths to absolute paths. **--appo**: Inserts new specific options. **--delimiter**: Specify any delimiter other than comma. **--ppcmd**: Used to specify the pre-processor command (Ex: **-E**) to replace the compile command. **--oppo**: Removes specific options.
Details	The method is to remove the compile option (Ex: **-c**), add pre-processor option (Ex: **-E**) and remove output option (**-o**).
Examples	**vcdb --ppcmd="-E" --oppo="-o=1" --file=xyz.c getppoptions**

Get Tags

Command	`gettags`
Description	Used to list the tags in the database. Tags define logical DBs within a physical db.
Syntax	`vcdb [--db=<dbname>] gettags`
Usage	Can be used with the `--prefix-var` option to port a database.
Details	SEARCH, LIB and TYPE are the types that can be set. Default type is SEARCH when the type is not specified.
Examples	`vcdb gettags` `vcdb --db=myown.db gettags`

Get Top Command

Command	`gettopcmd`
Description	Used to get the build command that is directly passed to the most recent vcshell command.
Syntax	`vcdb [--abspath] [--db=<dbname>] [--tag=<tagname>]gettopcmd`
Details	`--abspath` is required to get full path of the command

Examples	`vcdb gettopcmd`
	
	`vcdb --abspath gettopcmd`

Get Top Dir

Command	`gettopdir`
Description	Emits the full path of the directory in which vcshell was invoked.
Syntax	`vcdb [--db=<dbname>] [--tag=<tagname>] gettopdir`
Details	Unit index in a database starts from 1 and is unique to a given source file. This is automatically set by vcshell and cannot be altered.
Examples	`vcdb gettopdir`

Get Unit Index

Command	`getunitindex`
Description	Gets the index of a file in the database.
Syntax	`vcdb [--db=<dbname>] [--tag=<tag-name>] --file=<filename> getunitindex`
Usage	`--file` is a required option. `--db`: Optional argument. `--tag`: Optional argument.

Details	Unit index in a database starts from 1 and is unique to a given source file. This is automatically set by **vcshell** and cannot be altered.
Examples	**vcdb --file=anyfilename.cpp getunitindex** **vcdb getunitindex --file=\arbitrary\win\path\fname.c**

List Prefixes

Command	**list-prefixes**
Description	Lists all the environment variables stored in the database.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] list-prefixes**
Usage	Can be used with the **--prefix-var** option to port a database.

Rebase

Command	**rebase**
Description	Used to port a database to a new directory.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] --from=<filepath> --to=<filepath> rebase**
Usage	The prefix operations **--prefix-var** and **--prefix** are not permitted on a rebased database.
Details	Used to port a database to a new directory. It changes the common directory of an app in the database. The new common directory is the common path in the two given paths. The worst-case common directory is '/'. It is expected that the specified **from** path exists in the database and the specified **to** path exists in the file system. Only full paths are supported in both **from** and **to** specifications. After a rebase, all emitted paths are written with the new root.
Examples	**vcdb --db=rebase.db --from=/path/of/file/in/db.c --to=/path/of/new/file/db.c rebase**

Set Aliases

Command	**setaliases**
Description	With **setaliases**, two compilers can be specified as equivalent. All the compiler commands that are associated with one compiler are then tagged to both compilers.
Syntax	**vcdb [--db=<dbname>] [--tag=<tagname>] --cfg=<cfgfile> aliases=<aliaslist> setaliases**
Usage	**<aliaslist>** here refers to a list of comma-separated aliases of the form: **new_compiler=cfg_compiler**. **cfg_compiler** is the compiler specified in CCAST_.CFG.
Details	All **vcdb** commands with either of the compilers in **cmd-verb** will produce

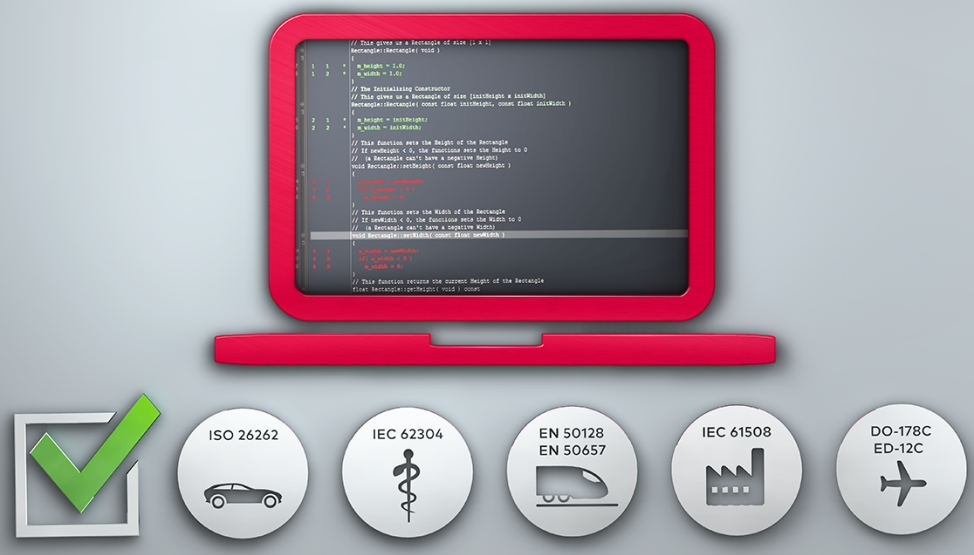
	the same result. This command is also available in **vcshell**.
Examples	**vcdb --aliases="cc=gcc,mycc=cfgcc" setaliases**

Set Path Type

Command	**setpathtype**
Description	Used to specify the include type for a path.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] [--r] setpathtype <path> [<SEARCH|LIB|TYPE>]**
Usage	**--r**: Associated option to specify application of type recursively to all the descendents of the specified path.
Details	SEARCH, LIB and TYPE are the types that can be set. Default type is SEARCH when the type is not specified.
Examples	**vcdb setpathtype /software/apps/Python-3.3.0 type**

Show Path Types

Command	**showpathtypes**
Description	Lists all paths for which type has been set.
Syntax	**vcdb [--db=<dbname>] [--tag=<tag-name>] showpathtypes**
Details	A '*' in the output indicates that the path has been set with the **--r** option.
Examples	**vcdb showpathtypes**



vcutil Command Line Reference

vcutil Overview

The **vcutil** utility allows you to perform common actions to files following the creation of the vcshell database. The utility is used to manipulate a vcshell database and to get and set attributes from CCAST_.CFG.

vcshell populates the vcshell database during the build process and then **vcutil** is used to populate or change specific values in the database. For example, **vcutil** can execute specific commands with the build commands of specific files.

vcutil Command Syntax

vcutil is a utility to get / set options in CCAST_.CFG. It is also used to process files in a database in different ways. **`instrument`** and **`codegen`** are examples of **vcutil** commands. **vcutil** gathers the required set of files or build data from a **vcshell**-created database.

The syntax for **`vcutil`** commands is given below. Please note that **`vcutil`** uses attributes *only* from the CCAST_.CFG file which is in the current directory (or in the path specified by **`--cfg`**); it does *not* use CCAST_.CFG files in VECTORCAST_DIR or **`home`** directories.

```
vcutil [--l<lang>] get_option <ATTRIB-NAME>

OR

vcutil [--l<lang>] set_option <ATTRIB-NAME> <ATTRIB-VALUES>

OR

vcutil [--cfg=dir][--db=<dbname>][--tag=<tagname>]
   [--flags=flags][--needabspath=<option-list>][--vcppopt]
   [--vcdebug][--jobs=num][--version]
   -file=<fpath> | --filelist=<fpath> | --all
   | codegen | instrument | parse

OR

vcutil --cmd=<script> [--db=<dbname>] [--jobs=num] [--tag=<tagname>]
  [--cfg=cfgfile] [--flags=<options> | --vcppopt ]
  --file=<fpath> | --filelist=<fpath> | --allfiles
  run

OR

vcutil --cmd=<script> [--db=<dbname>][--tag=<tagname>][--cfg=cfgfile]
  --app=<appname> | --applist=<applist> | --allapps run
```

Some examples are:

`vcutil -lc get_option C_COMPILE_CMD`

`vcutil -lc set_option C_EDG_FLAGS "--arm --no_wchar_t_word"`

`vcutil --cmd="python myscript.py" --db=x.db --allfiles run`

vcutil Options

[--all]	An option to specify all the files (from **vcdb getfiles**) in vcutil commands. **Instrument** and **parse** are examples.
[--allapps]	Used in **<vcutil run>** command to execute **<user-cmd>** on the link commands of ALL the apps in the database. Example: **vcutil --cmd="python x.py" --allapps run**
[--allfiles]	Used in **<vcutil run>** command to execute **<user-cmd>** for each file in the database.
[--app]	Used in **<vcutil run>** command to execute **<user-cmd>** on the link command of the specified app. This is also used in **getdestinationdir**, where **--app** refers to a specific **vcutil** command. Example: **vcutil --cmd="python x.py" --app=unique.app run**
[--applist]	Used to specify a list of apps for **<vcutil run>** command. The list is defined in a file and the filename is provided. The apps in the list may be specified: - with absolute paths - with relative paths - with just the appname, which must be unique in the vcshell database. Each appname should be specified in a separate line. Example: **vcutil --cmd=”scr.py” --applist=list.txt run**
[--cfg]	Directory containing the CCAST_.CFG file. The default is the current directory. Example: **vcutil --cfg=/software/apps/firefox_dir/ --all parse**
[--cmd]	Used to specify the command to be executed with the **run** command. The path of the command is to be provided. If provided in bare form, the command should be in path. The entire command with parameters, if any, is to be given. Example:

	vcutil --cmd="/some/path/ex.sh" --allfiles run
[--coverage_type]	Coverage type specifies the type of code coverage required in **instrument** command. It takes one of the following values: - statement - branch - basis_paths - mc/dc - none - probe_point - statement+mc/dc - statement+branch Default initial value: **statement**. When not specified, uses the type specified in an earlier command. Example: **vcutil --file=<myfile> --destination_dir=/sub/path/dir --coverage_type=probe_point instrument**
[--db]	The reference to the corresponding vcshell db that is to be used. The default name is **vcshell.db**.
[--destination_dir]	Specifies the directory to create instrument / parse output. This is optional and when not specified, the specification of an earlier command is used and if not specified the first time, the default values are: **vc-inst / vcparse** respectively. A directory is associated with a specific coverage type. That is, all the processed files in a specific directory have the same coverage type. Any related conflicts in the command line throw up an error. Once specified, the directory association is recorded and used in subsequent invocations as required. **getdestinationdir** is a related command that is used to get the destination directory for a particular **coverage_type**. Example: **vcutil --flags=\"-f=1,-m=1\" --coverage_type=statement --destination_dir=/sub/path/instrument**
[--dirnames]	Used to specify a list of directories. Each directory is recursively traversed and all files in the directory that also exist in the database are processed.

	Examples: **vcutil --file=baseName.c parse** **vcutil --cmd="scr.py" --file=x.c run**
[--file]	Specifies the file which is to be processed. Either the basename or the complete file path must be given. If multiple files of the same name exist in the database, full path must be given. Examples: **vcutil -cmd=″scr.py″ -filelist=list.txt run** **vcutil --filelist=<flist.txt> instrument | parse**
[--filelist]	Used to specify a list of files for a given operation. The list is defined in a file and the filename is provided. The files in list may be specified: - with absolute paths - with relative paths - just filenames, which must be unique in the vcshell database. Each filename should be specified in a separate line. The first entry in the filelist is used to ensure the file contains valid file entries. If the file existence check fails for the first entry, the filelist is not processed. Examples: **vcutil --filelist=/some/valid/path/filelist.xxx parse** **vcutil --cmd=″scr.py″ --filelist=list.txt run** **vcutil --filelist=<flist.txt> instrument | parse**
[--flags]	Used to specify special command line options. **--flags** are used when there are special options in the command line that are essential for compilation. When this option is followed by a parameter in the command line, **=1** must be specified. To specify a set of options, such as: **-Werror, -Wall,** the addition of a flag **-W=1** is sufficient. Also, if multiple flags match a specific option, the longest match is considered. For example, if flags contain the options **-c=1** and **-cpuarm** and if there is an option **-cpuarm** then it is matched. Example:

	vcutil --flags="-f=1,-W=1,-include=1" --all instrument
[--include_env_var]	Used to specify environment variables that hold the include directories. For example, - CPATH, C_INCLUDE_PATH, OBJC_ INCLUDE_PATH, etc., that are used by GNU tool chains. The include directories specified by these environment variables will be appended in the vcutil interface files. Example: **vcutil parse -all -include_env_var=CPATH**
[--incremental]	Used to process all unprocessed or erroneous files as compared to **--all**, where all the files are processed. Collects all files that are either: - not processed, or - for which errors have been reported, or - for which no content was found. For a fresh database, **--all** and **-incremental** are equivalent. This is the default option if none of (**--all,--file,--filelist,--dirnames,--incremental**) is specified. Example: **vcutil --flags="-D=1" --destination_dir=<dirname> --incremental instrument**
[--in_place]	Used in the **<vcutil instrument>** command to specify that parallel instrumentation will occur in-place, that is, in the source directory. The instrumented file replaces the original file and the original file is renamed with a **.vcast.bak** extension. The **destination.dir** directory takes precedence in creating the instrumented file if both of the options are specified. Example: **vcutil --all --in_place instrument**
[--instrument]	Specifies the type of code coverage required in **parse** command. It supports coverage types used in Industry modes and takes one of the following values: - statement - branch - basis_paths - none - mc/dc - statement+mc/dc - statement+branch Default initial value: **none**

	When not specified, uses the type specified in an earlier command. Examples: **vcutil --file=<myfile> --destination_ dir=/sub/path/dir --instrument=probe_point parse** **vcutil --flags=\"-f=1,-m=1\" --instrument=statement --destination_ dir=/sub/path/ parse**
[--jobs]	Used to suggest the maximum number of parallel instances to be executed. This can be used in any of the **vcutil** commands such as **instrument, parse** or **run**. The number of instances is the minimum of the number of licenses and the number specified. With the specification of **--jobs=MAX**, the number of instances is the minimum of the number of cores and available licenses. Without specification of **--jobs** too, **vcutil** creates parallel instances. The default is four for **instrument** commands and is based on the number of available cores for **parse**. Example: **vcutil --cmd="myscript.sh args" --jobs=MAX --allfiles run**
[--needabspath]	To specify the set of options for which absolute path emission is required. Suffix of **=1** is assumed here and need not be specified. This is required as the tool cannot determine parameters of specific options to be file paths automatically. specified. Example: **vcutil --file=fileX.c --needabspath="-optionX" instrument**
[--no_build]	Used in the **<vcutil instrument>** command to specify that the Cover environment be created by default in directory **vc-inst** or as specified by **--destination_dir**, but the instrumented sources are not added to the environment. Use the command **clicast -e <cover env> cover environment build** to add the sources later. Example: **vcutil --all --no_build instrument** **clicast -e vc-inst cover environment build**
[--omit_flags]	Used to recreate the unit options from the complete command line. The compile only flag (-c in gnu), the output flag (-o in gnu) and flags specified are removed from the command to create unit options.

	This may be useful in reconstructing command lines to resolve Preprocess or Parse errors. Example: **vcutil --omit_flags=-W=1,-fsyntax-only --file=/some/example.c instrument**
[--probe_points]	Used in instrument command to insert debug code at user specified locations. A file containing the points at which the code has to be inserted must be specified. Example: **vcutil --file=<myfile> --destination_dir=/path/ --probe_points=ppoints.pp --coverage_type=branch instrument**
[--tag]	Text string allowing the user to specify a unique identifier for the data collected.
[--vcdebug]	An option to generate vctools debug info. Debug info is emitted in **stdout**.
[--vcppopt]	Used to recreate the preprocessor options from the complete command line. The compile only flag (**-c** in gnu) and the output flag (**-o** in gnu) are removed from the compilation command and the pre-processor command (**-E** in gnu) is added to recreate the command. Example: **vcutil --vcppopt --file=/some/example.c instrument**
[--version]	Provides the **vcutil** version.

vcutil Commands

The **vcutil** commands are discussed below.

Get Option

Command	**get_option**
Description	Used to retrieve a CCAST_.CFG attribute value.
Syntax	**vcutil get_option <ATTRIB-NAME>**
Usage	The attribute name is required.
Details	- Fetches multiple line values for attributes: **LIBRARY_INCLUDE_DIR**, **TYPE_HANDLED_SOURCE_DIR** and **TESTABLE_SOURCE_DIR**. - Returns the value of the first occurrence by default. - Returns default values if an attribute is found in CCAST_.CFG

	- Expands any environment variables found in CCAST_.CFG
Examples	`vcutil get_option C_COMPILE_CMD`

Instrument

Command	`instrument`
Description	Used to instrument a given set of files and creates a Cover environment in the default directory (`vc-inst`) or in the directory specified by `--destination-dir`. By default, the instrumented sources are added to the Cover environment. The instrumented sources reside in the `<destination_dir>/vc-inst` directory. Instrumentation of files can happen in parallel, and the number of parallel jobs is influenced by the number of licenses, the number of input files and the specification in `--jobs` option.

Syntax	`vcutil [--cfg=dir][--db=<dbname>][--tag=<tagname>`
	`[[--flags=flags]|[--vcppopt]|[--omit_flags=flags]]`
	`[--file=<fpath>|--filelist=<fpath>`
	`|--all|--dirnames=<dir-list>|--incremental]`
	`[--vcdebug][--jobs=num][--coverage_type=<ctype>]`
	`[--probe_points=<fpath>]`
	`[--destination_dir=<dirname>][--in_place][--no_build]`
	`instrument`

Usage	Specific options: `--flags` can be used to specify special command line options. `--omit_flags`: all the options from compile command except the flags specified and flags like compile only and output are used as unit options. `--vcppopt` the pre-process command is used as an input. `--no_build`: specify that the Cover environment be created, but the instrumented sources are not added.
Details	The coverage type used is remembered, so that subsequent commands relating to that destination directory will use the same coverage type type when `coverage_type` is not specified on the command line.

Examples	`vcutil --file=<myfile> --destination_dir=/sub/path/dir --probe_points=ppoints.pp --coverage_type=branch instrument`
	
	`vcutil --flags="-f=1,-m=1" --incremental --jobs=4 --destination_dir=/sub/path/dir instrument`

Parse

Command	`parse`
Description	Used to parse the specified files. Build parameters are extracted from the database and the output is available in the user provided directory. The database itself is not changed. Instrumentation can happen in parallel by the

	specification of the **-jobs** option.
Syntax	**vcutil [--cfg=dir][--db=<dbname>][--tag=<tagname>** **[[--flags=flags]|[--vcppopt]|[--omit_flags=flags]]** **[--instrument=<coverage_type>]** **[--file=<fpath>|--filelist=<fpath>** **|--all|--dirnames=<dir-list>|--incremental]** **[--vcdebug][--jobs=num]** **[--destination_dir=<dirname>] parse**
Usage	Specific options: **--flags** can be used to specify special command line options. **--omit_flags**: all the options from compile command except the flags specified and flags like **compile only** and **output** are used as unit options. **--vcppopt** the pre-process command is used as an input.
Details	The coverage type used is remembered, so that subsequent commands relating to that destination directory will use the same coverage type type when **coverage_type** is not specified on the command line.
Examples	**vcutil --file=<myfile> --destination_dir=/sub/path/ parse** **vcutil --flags="-f=1,-m=1" --all --jobs=4 --destination_dir=/sub/path/ parse**

Post

Command	**post**
Description	Post helps to easily dispatch vctools ouput. In the event of any interactions with the VectorCAST support team, either the **vcshell.<pid>.log** file or the files created by using **--vcdebug** and redirecting the **stdout** ouput files can be dispatched using this command.
Syntax	**vcutil post <filelist>**
Usage	Filelist can be bare filenames or filenames with full path. The specified file should be available.
Details	These files are transferred to a VectorCAST internal site and are processed by the VectorCAST support team.
Examples	**Vcutil post vcshell.log vcutil.log**

Run

Command	**run**
Description	Used to execute any user command with build parameters as inputs.
Syntax	**vcutil --cmd=<script> [--db=<dbname>] [--tag=<tagname>] [--cfg=cfgfile] [--file=<fpath>	--filelist=<fpath>	--allfiles] [--flags=<options>	--vcppopt	--omit_flags=flags] run**

	OR **vcutil --cmd=<script> [--db=<dbname>] [--tag=<tagname>] [--cfg=cfgfile] [--app=<appname> | --applist=<applist> | --allapps] run**
Usage	Specific options: **--flags** can be used to specify special command line options. **--omit_flags**: all the options from compile command except the flags specified and flags like **compile only** and **output** are used as unit options **--vcppopt** the pre-process command is used as an input.
Details	**-vcutil --cmd=<user-cmd> --file=<fpath> run** is the basic syntax. Here, **<user-cmd>** is executed with the build parameters of **<fpath>**. - **<user-cmd>** should be either in PATH or an absolute path. - The input to **<user-cmd>** is a file with: CmdLine, current directory, fileName (or FileNotPresent), UnitOptions (or emptyline), AppName (or AppNotPresent) in SEPARATE lines. - Unit options by default have the options in C_DEFINE_FLAG and C_INCLUDE_FLAG. **--flags** or **--vcppopt**, when provided, are used in addition.
Examples	**vcutil --cmd="python myscript.py" --db=x.db --allfiles run** **vcutil --cmd="./xyz.sh "PARAM-STR"" --applist=xy.txt run**

RunMulti

Command	**runmulti**
Description	This command has been deprecated.

Set Option

Command	**set_option**
Description	Used to set a CCAST_.CFG attribute value.
Syntax	**vcutil set_option <ATTRIB-NAME>**
Usage	The attribute name and the value are required.
Details	- Writes the attribute and the value in the CCAST_.CFG file. - An existing entry is overwritten.
Examples	**vcutil -lc set_option C_EDG_FLAGS "-w --c --arm --no_wchar_t_word"** **vcutil -lc set_option LIBRARY_INCLUDE_DIR /vcast/include/dir "-w -gcc"**

Index

create vcshell database using putcommand 7
create vcshell database using script or makefile 7
limitations
vcshell 18
make substitutions to a vcshell database 8
putcommand 7-8, 17
options 17
tool components 5
vcdb
commands 27
dumpcommands 27
dumpverbs 27
getallcmdlines 28
getappfiles 28
getapps 29
getcmddir 29
getcommand 30
getdefines 30
getfilename 31
getfiles 31
getincludes 32
getlinkcmd 32
getobjpath 33
getoption 33
getpaths 33
getpathtype 34
getppoptions 34
gettags 35
gettopcmd 35
gettopdir 35
getunitindex 35
list-prefixes 36
rebase 36
setaliases 36
setpathtype 37
showpathtypes 37
options 20
abspath 20
aliases 20
app 20
appo 21
cfg 21
cflag 21
clicast 21
cmd 21
cmddir 21
cmd-verb 21
db 22
defines 22
delimiter 22
ext 22
file 22
flags 22-23
include env var 23
includes 23
libext 23
libraries 24
needabspath 24
nocache 24
oext 24
omit 24

oppo 24
optionprefix 25
outputflag 25
ppcmd 25
prefix 25
prefix-var 26
strd 26
stri 26
tag 26
vcdebug 26
vcpathstyle 26
version 27
overview 20
syntax 20
vcshell
commands
put 17
examples
build inside eclipse 17
echo 17
linux make 16
standalone compilation 16
standalone shell 16
windows devenv 17
windows msbuild 16
limitations 18
options 11
aux-cmd 12
cmd-verb 12
db 12
echo 13
expand 13
ext 13
inputcmds 14
out 14
tag 14
vcaliases 15
vcbg 15
vcdebug 15
vcenv 15
version 16
overview 11
syntax 11
vcutil
commands 45
get_option 45
instrument 46
parse 46
post 47
run 47
runmulti (deprecated) 48
set_option 48
options 40
all 40
allapps 40
allfiles 40
app 40
applist 40
cfg 40
cmd 40
coverage_type 41
db 41
destination_dir 41
dirname 41

file 42
filelist 42
flags 42
in place 43
include env var 43
incremental 43
instrument 43
jobs 44
needabspath 44
nobuild 44
omit_flags 44
probe_points 45
tag 45
vcdebug 45
vcppopt 45
version 45
overview 39
syntax 39
VectorCAST utilities
components 5
overview 5
work flow 5
work flow
create vcshell.db using putcommand 7
create vcshell.db using script or makefile 7
make substitutions to a vcshell database 8
using utilities 5