

# Integrating VectorCAST/Cover into a Make Environment

## Introduction

With a valid working Makefile already established, it is possible and very beneficial to incorporate VectorCAST/Cover into the `Make` environment to instrument code coverage. Your existing Makefile possesses the knowledge to create an executable program from your source files and libraries, so it is much more efficient to take advantage by also using it to include the instrumentation for code coverage of those files.

The integration of VectorCAST/Cover into your `Make` environment will allow for a *'make instrumented'* utility to automatically determine which source files in your program need to be recompiled for instrumentation, and invoke `clicast` commands to instrument coverage before recompiling. When the instrumented source files are recompiled and linked, a resultant executable program enables VectorCAST/Cover to provide post-execution analysis reports of the coverage achieved. This process is generic and can be performed with any compiler, platform, or version of `make`. For the purpose of further clarification, this document includes an example Makefile using GNU *'make'*.

## Integrating VectorCAST/Cover

Combining VectorCAST/Cover into your `Make` environment involves creating a cover environment, selecting an instrumentation option, adding program source files to the cover environment, instrumenting the files, and finally compiling and linking files to create the executable program.

There are two separate approaches that can be taken to combine VectorCAST/Cover into your `Make` environment. The first approach features a quicker and easier to implement method, while the second approach allows for more flexibility and less ambiguity. We will carefully describe the advantages to each approach and allow you to decide which is best suited for your environment.

### Approach 1: The VectorCAST "wrap\_compile"

The first approach, which will be made available with the release of VectorCAST version 4.2, introduces a `clicast` command that includes features to automate the steps involved in creating an instrumented coverage environment. The command used with this process is:

```
$(VECTORCAST_DIR)/clicast -e <environment-name> Cover Source  
wrap_compile <coverage_type> <compile-command>
```

The `$(VECTORCAST_DIR)` variable in the command refers to the `VECTORCAST_DIR` environment variable that needs to be set in order to invoke the `clicast` command. The `clicast` command passes in an environment name given to the Cover environment, a coverage type to be instrumented for, and a `compile` command associated with the specific compiler being used. The `wrap compile` implementation handles the automation

---

of the instrumentation process. It calls the clicast commands to create a new coverage environment if needed, sets the coverage option to *In\_place* instrumentation, adds the source files to the cover environment, and finally instruments for the specified coverage. The following are the commands, in succession, that are automated.

```
$(VECTORCAST_DIR)/clicast Cover Environment Create <environment-name>
$(VECTORCAST_DIR)/clicast -e <env> Cover Options In_place Y
$(VECTORCAST_DIR)/clicast -e <env> Cover Options Preprocessed_file Y
$(VECTORCAST_DIR)/clicast -e <env> Cover Source Add <unit.ext>
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument
Statement
```

### Creating a New Cover Environment

The first step and first command used towards coverage instrumentation is to create a Cover environment. The environment created will contain dependencies that are added at instrumentation time and are subsequently needed for the compiling and linking of the executable.

### Coverage Options

The coverage option selected in the second command, and the more feasible option, is to instrument the files *In\_place*. *In\_place* instrumentation is an improvement over having to copy all source and object files into the same directory for compilation and linking in order to create the instrumented executable program.

### Adding Source Files to Cover Environment

The fourth command that is automated by the wrap\_compile implementation is the addition of source files. As of the release of VectorCAST 4.1d, the command to add source files can be modified to add all or an arbitrary number of source files in a working directory to a cover environment. The wildcard (\*.ext) notation can be used to replace <unit.ext> in the command to add all source files in the directory. Multiple unit names can also be listed in the command to add a select number of source files.

Note: *In\_place* instrumentation will create backup (.bak) files of the un-instrumented source files. A un-instrument command, mentioned in the next section, removes the .bak files after it converts the instrumented source file back to their original state.

---

## Instrument Coverage

The final step before compiling and linking the executable is to select the type of coverage to instrument. The following are the five command options to instrument coverage:

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument  
Statement
```

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument Branch
```

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument Mcdc
```

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument LEVELB
```

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument LEVELA
```

The command to un-instrument coverage on a select source file or multiple source files is:

```
$(VECTORCAST_DIR)/clicast -e <env> [-u <unit>] Cover Instrument NONE
```

The [-u <unit>] is optional with each command.

## Preprocessing

Along with the process of combining compile commands for instrumentation, the `wrap_compile` implementation also parses the unit specific compilation arguments and features the ability to preprocess the source files before they are instrumented. These features, which are not available with the second approach, are ideal when coverage instrumentation is being performed on sources files that contain macros and need to be expanded.

## Requirements

When integrating VectorCAST/Cover into a make environment using the “`wrap_compile`” command, VectorCAST makes certain assumptions and requires certain information to be added to the existing Makefile.

The first requirement is that the “`wrap compile`” command replaces the `compile` command used in the existing Makefile. The `compile` command is passed to the “`wrap compile`” command and therefore can replace the existing `compile` commands in the Makefile. The only situation where the “`wrap compile`” command does not replace the existing is when compiling and linking the dependency files located in the Cover environment directory. Please see the *Compile and Link* section for more details.

---

The second requirement is that the working directory contains a configuration file with information pertaining to the template of the compiler being used. If a configuration file does not already exist in the working directory, then the template will have to be set using the following clicast command:

```
$(VECTORCAST_DIR)/clicast -lc TEMPLATE <template_name>
```

Entering the command in a Unix or DOS shell and replacing the `<template_name>` with `all` will provide a list of all available templates. If source files that are being compiled reside in more than one directory location, then the configuration file will need to be located in one of three possible directories:

- The VectorCAST installation directory
- The user's home directory
- The current working directory

The reason being is that VectorCAST looks for a configuration file in those directories when launched and the options are read in the order specified, with each file taking precedence over the previous.

The third requirement is that the source files being instrumented for coverage have not been previously compiled and no object file exists in the working directory. This requirement is not assumed using the second approach and is mainly due to the *In\_place* instrumentation option that the “wrap compile” command implements. If an object file exists then the files will not be compiled via the “wrap compile” command. To ensure that no such object files exist, run a make ‘clean’ in the Makefile directory.

The final requirement is that a full path be given to the Cover environment in the Makefile, so that it can be used from any directory in your project.

### Approach 2:

The second approach to integrating VectorCAST/Cover into a make environment is the more flexible and more transparent approach, although it will involve more modifications to your current build process. This approach differs from the previous one in that all VectorCAST clicast commands will need to be invoked in order to perform the necessary steps needed to create a coverage environment. The clicast commands in the first four steps, which are implicit in the previous approach due to the automation performed by the `wrap_compile`, will all need to be called in your Makefile when using the second approach. This allows for more flexibility and a more readable Makefile. An example of the flexibility would be if the *In\_place* coverage option is not the preferred option for your particular `make` environment. The option is automated and cannot be changed using the “wrap compile” approach.

---

The second approach differs from the “wrap\_compile” approach in that there is no assumption made that a ‘make clean’ be run on the working directory to ensure that the source files being added to the Cover environment have not been previously compiled. This approach simply re-instruments and re-adds the source files every time, requiring a re-compile each time. The second approach also differs in that it does not need the unit specific compilation arguments since instrumentation is applied to the source without preprocessing first.

## Compile and Link

After adding the source files to the cover environment and performing coverage instrumentation, the source files need to be compiled and linked to create the instrumented executable program. Along with those source files will also be the compilation and linking of the appropriate `c_cover_io.cpp` (or `c_cover_io.c`) file, which is located in the cover environment directory. The compile and link process is identical in both approaches, except that a Makefile using the “wrap compile” approach will have to make an adjustment to the compile command. As mentioned above, the compile commands will need to be replaced by the `wrap_compile` command, except when compiling and linking in the dependency file. The reason being is that the dependency file will not need to be pre-processed before it is compiled and linked.

## Sample Makefile

The following sample Makefile and two subsequent approaches, developed using GNU ‘make’, use the tutorial example to demonstrate how VectorCAST/Cover can be integrated into a `make` environment. The first approach demonstrates the easier to implement integration using the “wrap compile” command. The second approach then shows the VectorCAST/Cover integration when calling all clicast commands needed to create a coverage environment. In the existing Makefile, ‘*make*’ simply creates a tutorial executable after compiling and linking the source and object files. When ‘*make instrumented*’ is performed, each approach adds the source files to a cover environment, instruments for statement coverage, compiles and finally links to create an instrumented executable program.

```
##### EXISTING MAKEFILE #####
#--- Existing Makefile in this example doesn't contain any Includes

INCLUDES =
CC = g++

source_files = manager.cpp database.cpp manager_driver.cpp
obj_files = $(patsubst %.cpp, %.o, $(source_files))
```

```
tutorial.exe : $(obj_files) $(source_files)
    $(CC) -o tutorial.exe $(obj_files)

%.o : %.cpp
    $(CC) -c $< $(INCLUDES) -o $@

all : tutorial.exe

clean :
    rm -f tutorial.exe $(obj_files)
```

### **Approach 1:**

```
##### INTEGRATING VectorCAST/COVER #####

#--- Set the Cover Environment variable name.

VCAST_COVER_ENV = CoverENV

#--- The Instrumented utility target below adds the Cover environment
#--- directory as an Include. The wrap_compile command then
#--- substitutes the compile command used in the existing Makefile.
#--- The object file dependency needed for linking the instrumented
#--- executable is then added to the existing set of object files.
#--- Lastly, the instrumented target lists each prerequisite target
#--- needed for the integration process. It sets the compiler
#--- template, compiles the source files, creates the object file
#--- dependency, and creates the instrumented executable.

instrumented : INCLUDES += -I $(VCAST_COVER_ENV)
instrumented : CC = $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) \
    Cover Source wrap_compile Statement g++
instrumented : obj_files += c_cover_io.o
instrumented : VCAST_CCAST_CFG $(obj_files) c_cover_io.o all

#--- The c_cover_io.o target below creates the object file needed to
#--- link the instrumented executable. The target needs to include
#--- vcast_c_options.h as a prerequisite in the case that the coverage
#--- type is changed. The c_cover_io.cpp file will need to be
#--- recompiled to reflect the change to the coverage type.

c_cover_io.o : $(VCAST_COVER_ENV)/c_cover_io.cpp \
    $(VCAST_COVER_ENV)/vcast_c_options.h
    g++ -c $< $(INCLUDES) -o $@

#----- Set the Compiler Template -----#

VCAST_CCAST_CFG :
    $(VECTORCAST_DIR)/clicast -lc template GNU_CPP_34
```

```
##### UN-INSTRUMENT COVERAGE #####

uninstrumented : obj_files += c_cover_io.o
uninstrumented :
    $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) Cover Instrument None
    rm -f tutorial.exe TESTINSS.DAT $(obj_files)
    rm -r $(VCAST_COVER_ENV)*

.PHONY : clean instrumented uninstrumented
```

### **Approach 2:**

```
##### INTEGRATING VectorCAST/COVER #####

#--- Set the Cover Environment variable name.

VCAST_COVER_ENV = CoverENV

#--- The Instrumented utility target below adds the Cover environment
#--- directory as an Include. The object file dependency needed for
#--- linking the instrumented executable is then added to the existing
#--- set of object files. Lastly, the instrumented target lists each
#--- prerequisite target needed for the integration process.

instrumented : INCLUDES += -I $(VCAST_COVER_ENV)
instrumented : obj_files += c_cover_io.o
instrumented : $(VCAST_COVER_ENV) vcast_set_option add_source inst \
    c_cover_io.o all

#--- The c_cover_io.o target below compiles the dependency source file
#--- in the cover environment directory to create the object file
#--- needed for linking the instrumented executable.

c_cover_io.o : $(VCAST_COVER_ENV)/c_cover_io.cpp
    g++ -c $< $(INCLUDES) -o $@

#----- Create New Cover Environment -----#

$(VCAST_COVER_ENV) : $(VECTORCAST_DIR)
    $(VECTORCAST_DIR)/clicast Cover Environment Create $(VCAST_COVER_ENV)

#----- Coverage Option -----#

vcast_set_option : $(VECTORCAST_DIR) $(VCAST_COVER_ENV)
    $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) Cover Options
    In_place Y

#----- Add Source Files to Cover Environment -----#
```

---

```
add_source : $(VECTORCAST_DIR) $(VCAST_COVER_ENV)
    $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) Cover Source Add
$(source_files)

#----- Instrument Coverage -----#

inst: $(VECTORCAST_DIR) $(VCAST_COVER_ENV)
    $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) Cover Instrument
Statement

##### UN-INSTRUMENT COVERAGE #####

uninstrumented : obj_files += c_cover_io.o

uninstrumented :
    $(VECTORCAST_DIR)/clicast -e $(VCAST_COVER_ENV) Cover Instrument None
    rm -f tutorial.exe $(obj_files)

#----- Phony Target -----#

.PHONY : clean instrumented uninstrumented vcast_set_option inst \
    add_source
```