



# 21<sup>st</sup> Century Medical Scheduling America COMPETES Contest

## Instructions on Use of Testing Environment

The Virtual Machines provided to contestants have installed a pre-configured environment for the creation and execution of tests. The CMake ([www.cmake.org](http://www.cmake.org)) suite of tools is being utilized for this contest. This environment is using two main tools: CMake and CTest.

- CMake: An open source cross-platform build system. CMake can generate build components for a multitude of systems and build environments.
- CTest: An open source cross-platform test driver system. CTest integrates with the CMake configured system and is able to perform the tests that have been written for the system.

See <http://www.cmake.org/cmake/help/v2.8.10/cmake.html> for information on the CMake commands.

### Setup

The testing environment has already been configured in the three virtual machines that have been supplied to you as a contestant. However, as you modify the existing tests or add new tests you will have to rerun this configuration and generation process in order to get ready to run your new or modified tests.

CMake uses a set of files called `CMakeLists.txt` to read commands and build the testing environment. These files contain the full description of how to run each one of the tests. Once a `CMakeLists.txt` file is modified, the testing configuration must be updated in order to take such changes into account. This is done in two steps: (a) configure and (b) generate.

### Configure Step

This step can be performed at any time, by entering the command

```
$ cmake .
```

while in the Testing Binary directory (see Table 1 at the end of this document for the specific location of the directories). CMake will then reconfigure and regenerate the files and tests for

the Testing repository from the Testing source directory into the Testing binary directory (See Table 1 for details on the directories locations).

**Warning:**

All changes to the files should be done in the source directory specified in Table 1. Any changes made to the configured file located in the binary directory will be lost.

This command will need to be run at certain times such as when a change is made to a \*.in file, or a CMakeLists.txt file within the Testing source directory or when a new test file has been added. CMake will replace any variables in the input file referenced as \${VAR} or @VAR@ with their values as determined by CMake defined in CMakeLists.txt.

## Run tests

### Reproducibility Principle

Each one of the tests provided by the contestant must be reproducible and must be self-sufficient.

To enable this behavior, the test that demonstrates each one of the Use Cases must also provide a mechanism for restoring the VistA instance to the initial state as it was before the test. Such restoration mechanism could be, for example, a set of scripts that will take a clean database and configure it up to the point where the test can be run, or it could be a simple copy of the database files into an alternative directory. Whatever the restoration process is, it must be scripted in such a way that the evaluators can run it with a simple command. It is the contestant responsibility to provide clear instructions on this restoration process.

During the evaluation process, evaluators will run the tests of the Use Cases multiple times and in different orders. It is the contestant responsibility to ensure that the scripts it provides to run the tests are suitable for supporting that evaluation process.

### Running the Tests: Using CTest

The execution of the created tests is done using the CTest program. The tests are run from a command prompt, such as any of the following options:

- The *Terminal* in Linux,
- The *Git Bash Shell* in Windows
- The *Windows command prompt* in Windows.

From any of these options, changing the directory (cd) to the Testing Binary Directory of your test installation and typing “ctest” will run every test that has been created. Other ctest options allow or disallow specific tests or groups of tests to be run.

## CTest Basics

To see usage information for ctest along with the list of available options enter “ctest --help” or visit <http://www.cmake.org/cmake/help/v2.8.10/ctest.html> :

```
$ ctest --help
ctest version 2.8.10.1
Usage

    ctest [options]

Options
  -C <cfg>, --build-config <cfg>
                        = Choose configuration to test.
  -V,--verbose
                        = Enable verbose output from tests.
  -VV,--extra-verbose
                        = Enable more verbose output from tests.
  --debug
                        = Displaying more verbose internals of CTest.
  --output-on-failure
                        = Output anything outputted by the test program
                          if the test should fail. This option can
                          also be enabled by setting the environment
                          variable CTEST_OUTPUT_ON_FAILURE
  -F
                        = Enable failover.
<SNIP>
```

A very useful option is the ‘-N’, which will print out the names of the tests that would run, but not actually perform the tests.

```
$ ctest -N
Test project /home/contestant/OSEHRA/Dashboards/OSEHRA-Automated-Testing-build
Test #1: XINDEX_SAGG_Project
Test #2: XINDEX_Run_Time_Library
Test #3: XINDEX_Survey_Generator
Test #4: XINDEX_Police_and_Security
Test #5: XINDEX_Beneficiary_Travel
Test #6: XINDEX_Visual_Impairment_Service_Team
Test #7: XINDEX_NDBI
Test #8: XINDEX_Dental
<SNIP>
```

To run only a subset of the tests or alter the output that is shown on the screen, you can use one or more of those options to limit the tests. For example, the “-R” option allows you to specify a regular expression for the names of tests to be executed:

```
$ ctest -R XINDEX -V
```

That command will run the tests whose name matches the supplied regular expression '-R', in this case 'XINDEX', and with the '-V' (verbose) option it will display more information on the screen than usual.

CTest is used as the driver for all of the types of tests found with the testing suite. XINDEX, MUnit, and the Scenario tests can all be run using the commands above.

### Warning:

CTest is a powerful automatic testing fixture with the ability to interact through the web with remote repositories and dashboards. For the purposes of this contest, your version of the environment has these network capabilities removed to enhance confidentiality during the contest. If you execute certain CTest commands such as those that use the '-D' option, you should expect error messages with respect to "git" and dashboard submissions. These network error messages can be safely ignored. The actual results of your testing scripts should not be affected.

The following are two examples of those error messages that you may see and you can safely ignore:

```
Update command failed: "/usr/bin/git" "fetch"
  Configure project

Drop site:http://
Submit failed, waiting 5 seconds...
Retry submission: Attempt 1 of 3
Submit failed, waiting 5 seconds...
Retry submission: Attempt 2 of 3
Submit failed, waiting 5 seconds...
Retry submission: Attempt 3 of 3
Error when uploading file:
/home/contestant/OSEHRA/Dashboards/OSEHRA-Automate
d-Testing-build/Testing/20121207-0400/Build.xml
Error message was: Couldn't resolve host ''
Problems when submitting via HTTP
```

## Run Example Scenario

For the Test 1 phase, contestants are required to implement eight use case scenarios and to provide automated tests for them. In order to provide guidance to contestants, the testing environment provides an example of a full implementation of one of those use cases. Also, for the convenience of the contestant, placeholders for the remaining seven use cases are provided as well.

These files have already been added as tests to the CMake configuration. To execute the test use CTest to call it by name (Scenario001) with the -R option, for example:

```
$ ctest -R CASE_Scenario001
```

## Add new tests

Creating new tests is a vital part of making a safe piece of software. To add a new test to a particular type in the current testing environment, follow these instructions.

As a suggestion, contestants could create local Git branches to host the modifications that they make to the local Git repositories in the Virtual Machines. This is not a requirement, only a suggestion for a potential workflow.

## XINDEX:

- If a new package is to be added to the environment, add a new folder in the VISTA-FOIA repository Packages directory (See Table 1 for details). The folder should have the same name as the package it is testing and have the same internal structure as the other folders. It should contain two folders: one labeled 'Routines' and one labeled 'Globals'. This folder will be found automatically by CMake and the XINDEX test will be created for it.
- If you are adding new routines to an existing package, place the routines into the 'Routines' folder within the VISTA\_FOIA repository package folder (See Table 1 for details) and CMake will automatically add the new routine to be checked in the XINDEX test for that package.

## MUnit:

- If the test to be added is in a new package or in a package that currently does not have MUnit tests, add a folder in the UnitTest/Vista-FOIA/Packages directory within the testing repository. The folder should have the same name as the package it is testing. The test routine should be placed in that folder. The next time CMake is run, it will create a test that will import that routine and run it as a test.

- If the test is being added to an existing package, place the test routine into the existing UnitTest/VistA-FOIA/Packages/ folder. The next time CMake is run, it will be added to the files imported by the test and will be run in the set of MUnit tests for that package.

### Use Case Scenario:

- If the scenario that is to be tested falls in the 2 - 8 range, there is no work to be done other than to write the script in the proper Python file. These files, which are found in the UseCases/ directory, have already been linked to a test in the CMakeLists.txt.

#### In Linux

```
/home/contestant/OSEHRA/Dashboards/OSEHRA-Automated-Testing/UseCases/
Scenario001.py
Scenario002.py
Scenario003.py
Scenario004.py
Scenario005.py
Scenario006.py
Scenario007.py
Scenario008.py
```

#### In Windows

```
C:\Users\contestant\OSEHRA\Dashboards\OSEHRA-Automated-Testing\UseCases
```

If more scenarios are being added, the Python file should be placed in the UseCases/ directory and a new line be added to the end of the CMakeLists.txt that is in that directory:

```
add_test(CASE_Scenario0## ${PYTHON_EXECUTABLE} "${CMAKE_CURRENT_BINARY_DIR}/Scenario0##.py"
```

- The ‘##’ symbol in the line above should be replaced by the number of the scenario script.

If modifications, such as the above, are made to the testing infrastructure, contestants must rerun CMake before attempting to run the tests that they have added or modified. This can be done by typing:

```
$ cmake .
```

from within the Testing binary directory (See Table 1 for details).

### General Test Additions:

- If adding a test that does not fall into one of the above categories, the test can be added via the CMakeLists.txt in either the Testing Source directory or one of the subdirectories using the command 'add\_test':

```
add_test( ${TESTNAME} ${EXECUTABLE} [POSSIBLE ARGUMENTS] )
```

See [http://www.cmake.org/cmake/help/v2.8.10/cmake.html#command:add\\_test](http://www.cmake.org/cmake/help/v2.8.10/cmake.html#command:add_test) for more information.

## Table 1. Important Directories

### *Windows*

#### Binary Directory:

```
C:\Users\contestant\OSEHRA\Dashboards\OSEHRA-Automated-Testing-build
```

#### Source Directory:

```
C:\Users\contestant\OSEHRA\Dashboards\OSEHRA-Automated-Testing
```

#### VistA-FOIA Directory:

```
C:\Users\contestant\OSEHRA\Dashboards\VistA-FOIA
```

### *Linux*

#### Binary Directory:

```
/home/contestant/OSEHRA/Dashboards/OSEHRA-Automated-Testing-build
```

#### Source Directory:

```
/home/contestant/OSEHRA/Dashboards/OSEHRA-Automated-Testing
```

#### VistA-FOIA Directory:

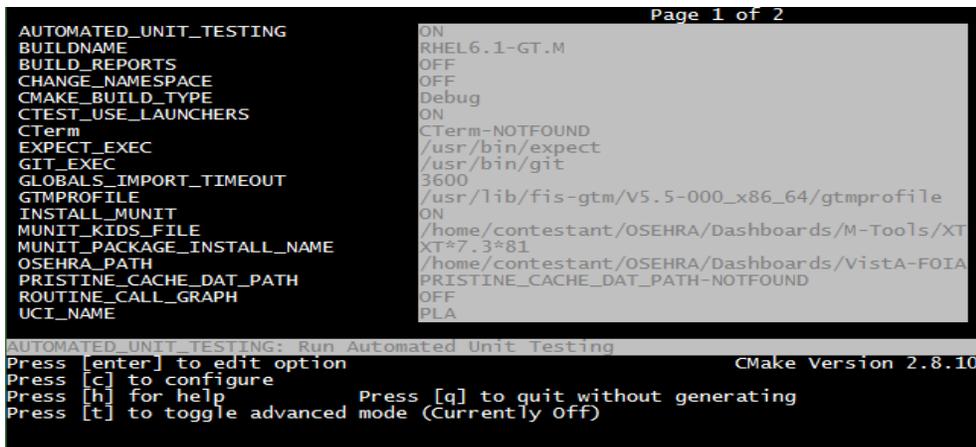
```
/home/contestant/OSEHRA/Dashboards/VistA-FOIA
```

## CMake GUIs

In addition to the CMake command line tools (cmake, ctest), the CMake system also provides Graphical User Interfaces (GUIs).

### *In Linux*

In the Linux platform CMake offers a GUI interface based on the Curses library, called “ccmake”. The following screenshot illustrates how this application would look when run from the command line:



```
Page 1 of 2
AUTOMATED_UNIT_TESTING ON
BUILDNAME RHEL6.1-GT.M
BUILD_REPORTS OFF
CHANGE_NAMESPACE OFF
CMAKE_BUILD_TYPE Debug
CTEST_USE_LAUNCHERS ON
CTerm CTerm-NOTFOUND
EXPECT_EXEC /usr/bin/expect
GIT_EXEC /usr/bin/git
GLOBALS_IMPORT_TIMEOUT 3600
GTMPROFILE /usr/lib/fis-gtm/V5.5-000_x86_64/gtmprofile
INSTALL_MUNIT ON
MUNIT_KIDS_FILE /home/contestant/OSEHRA/Dashboards/M-Tools/XT
MUNIT_PACKAGE_INSTALL_NAME XT*7.3*81
OSEHRA_PATH /home/contestant/OSEHRA/Dashboards/VistA-FOIA
PRISTINE_CACHE_DAT_PATH PRISTINE_CACHE_DAT_PATH-NOTFOUND
ROUTINE_CALL_GRAPH OFF
UCI_NAME PLA

AUTOMATED_UNIT_TESTING: Run Automated Unit Testing
Press [enter] to edit option
Press [c] to configure
Press [h] for help
Press [t] to toggle advanced mode (Currently Off)
CMake Version 2.8.10
Press [q] to quit without generating
```

### *In Windows*

In the Windows platform CMake offers the “cmake-gui” application. The following screenshot illustrates how this application would look:

