CCMCL User Manual

National Institute of Standards & Technology

Description

CCMCL is a real time combinatorial coverage measurement tool. It is based off of an older tool – CCM, which had similar but limited functionality. CCMCL brings several new features to the user which will be discussed in this user manual.

Supported Operating Systems:

- Windows
- Mac OS
- Linux

Key Features:

- Measure the combinatorial coverage of static test case files
- Generate missing combinations from given test cases
- Generate random tests
- Measure combinatorial coverage in real time via various input modes
    - Standard Input
    - Output of external programs
    - TCP/IP
- Specify robust constraints
- Support for equivalence classes and groups via SET notation definitions
- Read ACTS configuration files via .txt or .xml

For any questions regarding the tool please contact:

Zachary Ratliff: ratliffzachary@yahoo.com
Rick Kuhn: kuhn@nist.gov
Dylan Yaga: dylan.yaga@nist.gov

GitHub Page: https://github.com/usnistgov/combinatorial-testing-tools

Classic Mode

Classic mode is the command line version of the original CCM tool. It provides all the same functionality as the GUI version, but with some extra functionality added in. By default, classic mode is enabled, and gives the user a lot of power in static analysis of test cases. In this section we will explain the various command line parameter options for classic mode of CCMCL.

Command Line Arguments Pertaining to Classic Mode:

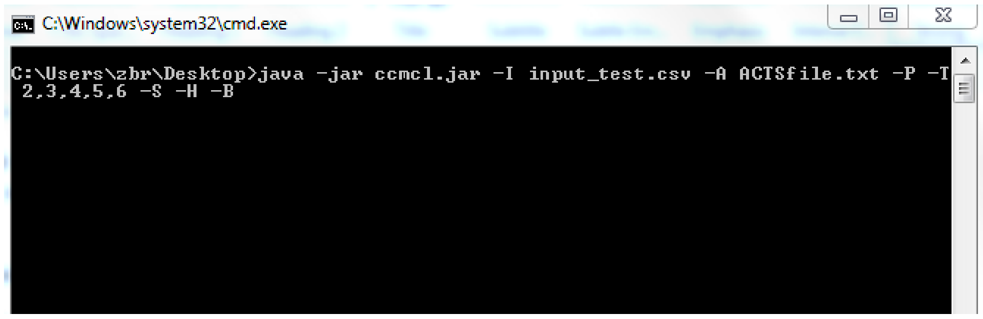
To see a list of all the possible command line arguments run:

java -jar ccmcl.jar --help

```
--inputfile (-I) : [path to test case file (.txt, .csv)]

--ACTSfile (-A): [path to .txt or .xml ACTS file]

--constraints (-C): [path to .txt file containing constraints]

--tway (-T): [2,3,4,5,6] *any order and any combination of these values.*

--generate-missing (-G): *generates missing combinations not in test file.*

                          *Must include "-m" and "-o" with this option.*

                          *Not available for real time mode (-R).*

--minimum-coverage (-m): *Minimum coverage for generating missing combinations*

--output-missing (-o): *output path for the missing combinations.*

--append-tests (-a): *appends original tests to missing combinations file.*

--parameter-names (-P): *parameter names are first line of test case file (-I)*

--parallel (-p): *Puts the program in parallel processing mode.*

--generate-random (-r): *Sets the program to generate a random set of inputs.*

                        *Must include "-n" and "-f" with this option. *

                        *Not available in real time mode (-R).*

--number-random (-n): *Amount of random inputs to generate.*

--output-random (-f): *Path to output the random test cases to.*

--stepchart (-S): *Generates a step chart displaying t-way coverage.*

--barchart (-B): *Generates a bar chart displaying t-way coverage.*

--heatmap (-H): *Generates a 2-way coverage heatmap.*

--display-progress (-d): *Displays progress of coverage measurement*
```

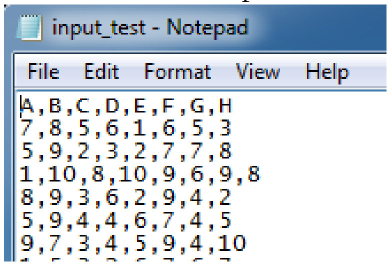
To test the current version CCMCL, download the ccmcl.jar file from GitHub. If you would like to try out the demo we created for you, be sure to also download the "Examples" folder. If you are on Windows, use the batch file to quickly get up and running, or try out the examples in the command line for yourself.

We will demonstrate one of the examples here for you. Below you can see the command line arguments that were used to launch the ccmcl.jar executable.

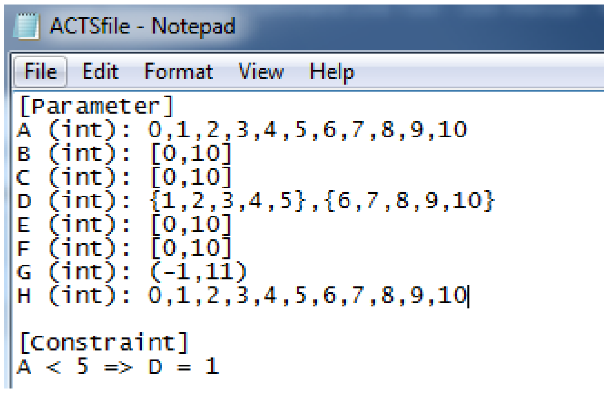


```
C:\Users\zbr\Desktop>java -jar ccmcl.jar -I input_test.csv -A ACTSfile.txt -P -T
 2,3,4,5,6 -S -H -B
```

Below is an explanation for each command line argument and what exactly it is doing:

1. **java -jar ccmcl.jar**
   - This launches the ccmcl.jar program. (Be sure you have Java installed.)
2. **-I input_test.csv**
   - This input parameter specifies that the test cases will come from a comma separated value file. An example of what these files will look like is shown below:
   - 

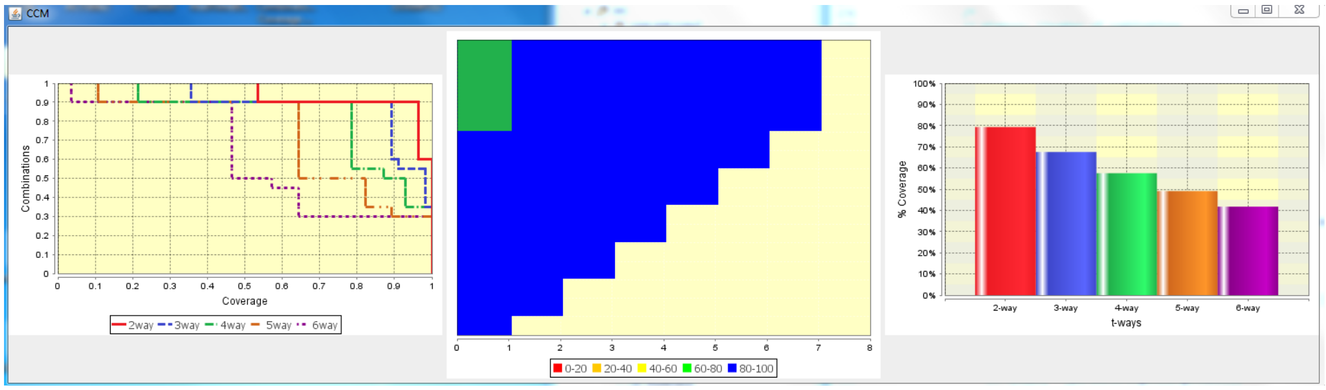
```
A,B,C,D,E,F,G,H
7,8,5,6,1,6,5,3
5,9,2,3,2,7,7,8
1,10,8,10,9,6,9,8
8,9,3,6,2,9,4,2
5,9,4,4,6,7,4,5
9,7,3,4,5,9,4,10
```

3. **-A ACTSfile.txt**
   - This input parameter specifies that an ACTS configuration file will used to define the input domain. ACTS files can be either .txt or .xml format. An example of the text version is shown below:
   - 

```
[Parameter]
A (int): 0,1,2,3,4,5,6,7,8,9,10
B (int): [0,10]
C (int): [0,10]
D (int): {1,2,3,4,5},{6,7,8,9,10}
E (int): [0,10]
F (int): [0,10]
G (int): (-1,11)
H (int): 0,1,2,3,4,5,6,7,8,9,10

[Constraint]
A < 5 => D = 1
```

4. **-P**
    - This option simply specifies that in the test case file (input_test.csv), the first row is the parameter names. Include this option if this is the case.
5. **-T 2,3,4,5,6**
    - This option specifies the t-way levels you would like to measure. Notice only the values (2,3,4,5,6) are available. In this case, we selected to measure all of them.
6. **-S**
    - This tells the CCMCL program to display a stepchart of the measured combinatorial coverage.
7. **-B**
    - This tells the CCMCL program to display a barchart of the measured combinatorial coverage.
8. **-H**
    - This tells the CCMCL program to display a heatmap of the 2-way coverage measured. Notice, the heatmap option is only available for 2-way coverage.

Below is an example of what the output from this program might look like:



So let's break down what happened in this example:

1. CCMCL is started and given command line arguments telling it what to do
2. The input_test.csv file contains several test cases used to test a program
3. The ACTSfile.txt file contains information defining the input domain. Without this file, the program will enter "auto-detect" mode. We will talk more about auto-detect later. You may have also noticed way that the ACTS configuration file defines the input domain. We will also go more in depth on this later.
4. We told the program that we wanted to measure 2 through 6 way combinations of the test case file.
5. The program output several graphs showing us the various t-way coverages.

In the next section we will discuss the importance of the ACTS configuration files and how they are used.

ACTS Configuration Files

In this section we will take an in depth look at how CCMCL uses the ACTS configuration files. If you aren't familiar with ACTS, it is another combinatorial testing tool which generates t-way covering arrays that can be used in software testing. However, ACTS is not necessary for CCMCL to operate. We simply just followed the same syntax so that the files would be more universal and operate on a variety of tools.

There are 3 different header definitions within the ACTS configuration files that the user can choose to define. The first one is mandatory, but the other two are optional. Any other headers within the ACTS file will be ignored.

1. [Parameter] – list all parameter definitions underneath this header
2. [Constraint] – list all constraint definitions underneath this header
3. [Test Set] – list all the test sets you want underneath this header.

Below we will demonstrate how to define the various sections within the ACTS configuration file.

Parameters are defined in the following format:

parameter_name (data_type) : value1, value2, value3, …

The data types that are allowed are:

- enum (string / character values)
- int
- boolean

Constraints are defined in the following format:

(constraint1 condition) => (constraint2 condition)

The constraints can be defined in a variety of different ways for example:

employee && (time > 0800 && time < 17:00) => grant

Test cases are defined in the following format:

v1,v2,v3,v4,v5
v1,v2,v3,v4,v5

Where **v1** is a valid value for parameter 1, **v2** is a valid value for parameter 2, etc.

To further explain the configuration files, let's imagine a scenario in which we are trying to define the input domain of a desktop application. To start, we can say that the app is supposed to operate given the following environment:

- Operating System is either Android, iOS, or Windows.
- Permission has been granted by the user for the app to access internal files.
- An internet connection has been established.

We can also say that each of these environmental factors can lead to a wide array of other configurations so that we have configuration options such as:

- Browser type.
- The user is on Wi-Fi, cellular data (air card), or Ethernet.

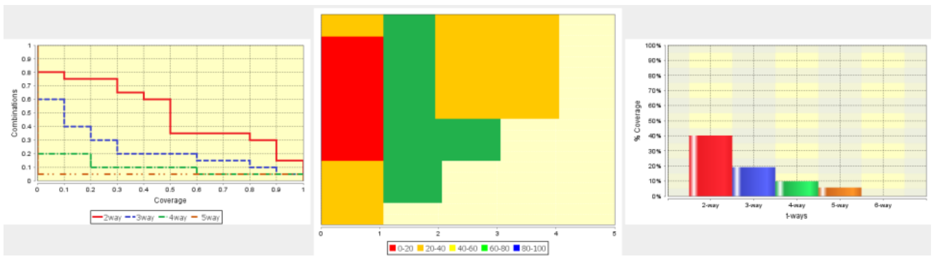
This is a somewhat small example, but should serve the purpose as to explaining how we can create our own ACTS format file. An example ACTS configuration file could look like:

```
[Parameter]
os (enum): windowsXP, windows7, windows8, redhat, debian, ubuntu, osx
permission (bool): true, false
internet (bool): true, false
internet_type (enum): wifi, cellular, ethernet, none
browser (enum): ie, firefox, safari

[Constraint]
os != "windowsXP" && os != "windows7" && os != "windows8" => browser != "ie"
os != "osx" => browser != "safari"
!internet => internet_type = "none"
internet => internet_type != "none"

[Test Set]
windowsXP,true,true,wifi,firefox
osx,true,true,ethernet,safari
debian,false,true,wifi,firefox
windows7,true,false,none,ie
windows7,false,true,ethernet,ie
```

A few things you should notice is how various data types are defined within constraints. For example, the **enum** data types values are represented between quotation marks when referencing them inside a constraint.

Running CCMCL on this ACTS configuration file with the following command line arguments:

```
java -jar ccmcl.jar -A desktop_app_ACTS.txt -T 2,3,4,5 -d -p -S -H -B
```



ACTS files can also be in .XML format. We could have defined the same exact input domain as the file on the previous page by using:

```
<?xml version="1.0" encoding="UTF-8"?>
<System name="desktop_app">
  <Parameters>
    <Parameter id="0" name="os" type="1">
      <values>
        <value>windowsXP</value>
        <value>windows7</value>
        <value>windows8</value>
        <value>redhat</value>
        <value>debian</value>
        <value>ubuntu</value>
        <value>osx</value>
      </values>
    </Parameter>
    <Parameter id="1" name="permission" type="2">
      <values>
        <value>true</value>
        <value>false</value>
      </values>
    </Parameter>
    <Parameter id="2" name="internet" type="2">
      <values>
        <value>true</value>
        <value>false</value>
      </values>
    </Parameter>
    <Parameter id="3" name="internet_type" type="1">
      <values>
        <value>wifi</value>
        <value>cellular</value>
        <value>ethernet</value>
        <value>none</value>
      </values>
    </Parameter>
    <Parameter id="4" name="browser" type="1">
      <values>
        <value>ie</value>
        <value>firefox</value>
        <value>safari</value>
      </values>
    </Parameter>
  </Parameters>
  <Constraints>
    <Constraint text="os != &quot;windowsXP&quot; &amp;&amp; os != &quot;windows7&quot; &amp;&amp; os != &quot;windows8&quot; => browser != &quot;ie&quot;"/>
    <Constraint text="os != &quot;osx&quot; => browser != &quot;safari&quot;"/>
    <Constraint text="!internet => internet_type = &quot;none&quot;"/>
    <Constraint text="internet => internet_type != &quot;none&quot;"/>
  </Constraints>
  <Testset doi="2">
    <Testcase TCNo="0">
      <value>0</value>
      <value>windowsXP</value>
      <value>true</value>
      <value>true</value>
      <value>wifi</value>
      <value>firefox</value>
    </Testcase>
    <Testcase TCNo="1">
      <value>1</value>
      <value>osx</value>
      <value>true</value>
      <value>true</value>
      <value>ethernet</value>
      <value>safari</value>
    </Testcase>
    <Testcase TCNo="2">
      <value>2</value>
      <value>debian</value>
      <value>false</value>
      <value>true</value>
      <value>wifi</value>
      <value>firefox</value>
    </Testcase>
    <Testcase TCNo="3">
      <value>3</value>
      <value>windows7</value>
      <value>true</value>
      <value>false</value>
      <value>none</value>
      <value>ie</value>
    </Testcase>
    <Testcase TCNo="4">
      <value>4</value>
      <value>windows7</value>
      <value>false</value>
      <value>true</value>
      <value>ethernet</value>
      <value>ie</value>
    </Testcase>
  </Testset>
</System>
```

The previous page demonstrates what an extensible markup language format of the ACTS files look like. You should note that some ACTS files may contain various xml tags that aren't listed in this example. These will be ignored, as only the ones listed here are required. There are a few rules that one should keep in mind when using ACTS .xml files:

1. The first <value> in a test-case represents what test-case number it is. This is required as it follows the original ACTS format.
2. Constraints are defined within the "text" attribute, and escape sequences are necessary i.e. &amp; or &quot;
3. Parameter types represent the data type of the parameter. There are 5 different data types in xml format:

   0 = int

   1 = enum

   2 = boolean

   3 = range class

   4 = group class

The range class and group classes represent data types that can be defined as equivalence classes. Defining these values follow the same format as SET notation. For example, we could have defined our Operating System parameter as follows:

os (enum): {windowsXP, windows7, windows8},{redhat, debian, ubutnu},{osx}

This would represent a **group class** – where values in each group are considered identical e.g. all Windows operating systems should be treated the same.

Let's say we wanted to use a **range class** to represent a parameter called *age*. We would use SET notation to define the parameter as follows:

age (int): [0,12],[13,17],[18,64],[65,*)

This would set up 4 different range classes as follows:

***Class 0*: Ages from 0 to 12**

***Class 1*: Ages from 13 to 17**

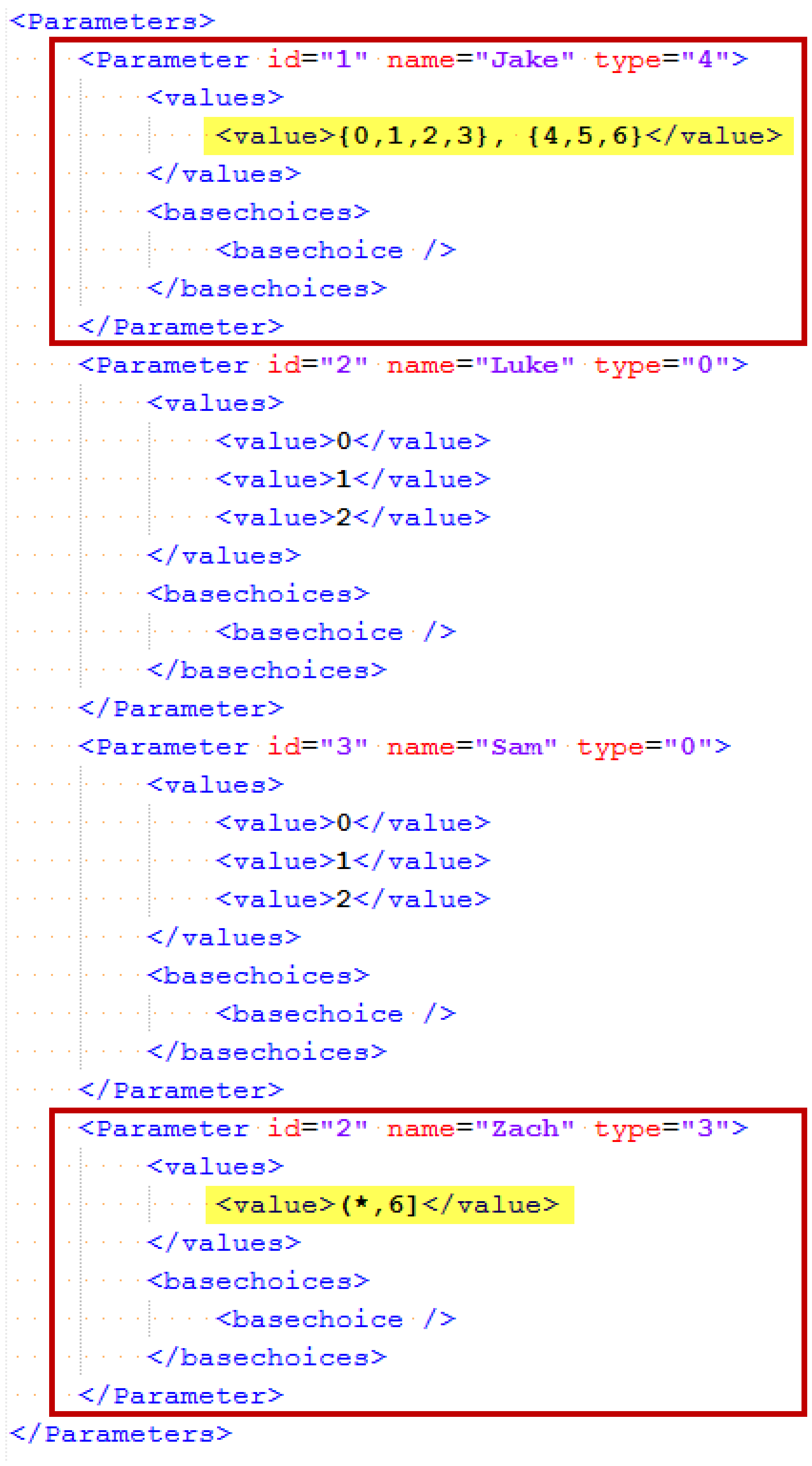
***Class 2*: Ages from 18 to 64**

***Class 3*: Ages 65 and older**

It is important to keep in mind the index of each class when defining group and range classes. Each range or group specified goes from 0 to the index of the last group. This is important for when we are defining constraints. For example, if we use the groups listed above for the **os** parameter, our previous constraint example restricting certain browsers to certain operating systems would instead look like:

os != 0 => browser != "ie"

Defining these groups in a .xml file would look as follows:



```xml
<Parameters>
    <Parameter id="1" name="Jake" type="4">
        <values>
            <value>{0,1,2,3}, {4,5,6}</value>
        </values>
        <basechoices>
            <basechoice />
        </basechoices>
    </Parameter>
    <Parameter id="2" name="Luke" type="0">
        <values>
            <value>0</value>
            <value>1</value>
            <value>2</value>
        </values>
        <basechoices>
            <basechoice />
        </basechoices>
    </Parameter>
    <Parameter id="3" name="Sam" type="0">
        <values>
            <value>0</value>
            <value>1</value>
            <value>2</value>
        </values>
        <basechoices>
            <basechoice />
        </basechoices>
    </Parameter>
    <Parameter id="2" name="Zach" type="3">
        <values>
            <value>(*,6]</value>
        </values>
        <basechoices>
            <basechoice />
        </basechoices>
    </Parameter>
</Parameters>
```

Note: *The entire range and group class should be defined in one <value> tag.*

Real Time Mode

Real time mode is the newest addition of functionality to the Combinatorial Coverage Measurement project. The ability to measure combinatorial coverage can be applied to several areas of interest in various industries. Some of its potential uses are:

- Product Readiness
- IV&V Performance Measurement
- Analysis of current test suite implementations
-

Command Line Arguments Pertaining to Real-Time Mode:

To see a list of all the possible command line arguments run:

java -jar ccmcl.jar --help

```
--realtime (-R) : *Sets the tool in Real Time Measurement Mode*

                  *Must specify input type (-k),(-e), or (-t)*

                  *Must specify ACTS configuration file (-A)*

-k : *Specifies real time mode to accept keyboard (stdin) input*

-e : [path to executable ,program argument 1,program argument 2,etc.]

NOTE: The (-e) option specifies real time mode to accept input from the standard
output (console output) of another program.

-t : [port] *Specifies real time mode to accept TCP input on the port*

--thread-max (-tm): [max number of threads] *Default 500 threads*

NOTE: Setting a high thread rate can cause problems in certain situations. It is
recommended to either throttle data coming into the program or use the recommended
(default settings)... Unless you know what you are doing...

--log (-L): [log file path] *Logs incoming data and measurement information*

NOTE: --log creates a report holding invalid combinations, total coverage, new tests,
etc. that occur while measuring in real time mode.
```

To put CCMCL in Real Time Mode, specify the **-R** switch, along with the switch pertaining to the method of input e.g. **-k** to accept input from the keyboard. It is also recommended to specify a log file. The log file contains information about incoming data that the tool has received. An example of how to run the Real Time Mode of the tool is as follows:

java -jar ccmcl.jar -A acts.xml -R -e gen.exe -B -T 2 -tm 1500 –- log.txt