OPBM

Office Productivity Benchmark

August 16, 2011

**How to Use OPBM’s Command Line Interface**

By Van Smith and Rick C. Hodgin

Cossatot Analytics Laboratories, LLC.

or “Cana Labs”

**Overview**

OPBM is a robust GUI application with a developer interface allowing full editing of entire benchmark suites. It also contains a simple skinned user interface allowing for streamlined access to Trial Runs, Official Runs and benchmark results viewing.

To facilitate the needs of automated script execution and post-script-execution termination, a command line interface has also been created. This allows OPBM to launch, execute a desired workload, and terminate immediately thereafter, in a self-contained, controlled manner, utilizing all features of the harness directly from the command line or input files.

**Command Line**

The syntax for executing OPBM is one of the following:

**1| java -jar c:\path\to\opbm.jar**

or to create a shortcut which has for its executable:

**2| c:\path\to\opbm.jar**

The first example will launch a Java debug window, useful for identifying errors or observing periodic notes and debug information generated at startup and runtime automatically by OPBM.

Command line options are in the form of single hyphen prefixing the option:

**Syntax: -option**

**@filename**

The following section outlines the available commands from the command line.

Currently, there are some command-line parameters defined:

**1| java -jar opbm.jar –trial**

**2| java -jar opbm.jar –official**

**3| java -jar opbm.jar –simple**

**4| java -jar opbm.jar –skin**

**5| java -jar opbm.jar –developer**

*Note: A -restart option is NOT provided because OPBM contains internal logic which will instruct it to restart a benchmark if one was in progress when OPBM last exited, regardless of whether it exited cleanly or not. To prevent OPBM from re-starting automatically, clear out all files in the c:\users\user\opbm\running\ directory.*

Options 1 and 2 will execute a full Trial Run, or a full Official Run, when fully implemented (today there are placeholder dialog boxes). Options 3 and 4 are identical, but allow for the choice of name to access the simple, skinned user interface. Option 5 brings up the developer window. Early users of Opbm will know the developer window as the original window that was provided with Opbm.

From the simple, skinned interface, an option is provided near the bottom called “Developer Interface” which allows switching to the developer window. And on the developer window’s main navigation pane (click “Home” to go to the main navigation pane if on a sub-menu), there is a new “Skin” option, which takes the user to the simple, skinned interface. In this way, the user can move back-and-forth between the interfaces for demonstration purposes, or when developing new scripts.

**Script Execution**

In terms of script execution, in this release only Atoms can be run (an Atom is the smallest workload currently supported for direct execution within OPBM, though worklets are known to OPBM as “data points” within a single Atom, visible in the Results Viewer). Atoms can be repeated in succession as needed by extending the command line syntax slightly.

Of the sample scripts provided by *Cana Labs*, one Atom has been called “Word Alice”. This script will launch Internet Explorer 9 and perform several operations on an Alice in Wonderland html web-book, copy-and-paste into Microsoft Word, perform several steps such as navigate through pages, change formatting, fonts, and will write the document to an Adobe PDF format, manipulate in Acrobat Reader, and then exit all launched apps.

**To execute this sample Atom automatically from the command line:**

1. Strip the spaces from the name, “Word Alice” becomes “WordAlice” (not case sensitive)
2. Prefix with the text “-atom:”, as in “-atom:wordalice”
3. If iterations are needed, use “-atom(5):wordalice” to iterate five times, for example.

The syntax for executing OPBM with a command line option is:

**java -jar opbm.jar –atom:wordalice**

Multiple Atoms can be specified on a single input line, as in:

**java -jar opbm.jar -atom:wordalice -atom:anotheratom**

To execute each script 10 times in iteration:

**java -jar opbm.jar -atom(10):wordalice -atom(10):anotheratom**

**Input Files**

To facilitate the execution of similar benchmarks on multiple machines, a file can be constructed which contains the sequence of benchmarks to run in the order desired. This file can be copied out to the SUTs through scripts, or accessed from a network location, and used by OPBM at startup to execute identical benchmarks across a host of machines.

The input file has the same syntax as the command line, though each entry must be stored on a separate line, as in:

***sample.txt*** *contains six lines (line numbers are for illustration only, do not use them):*

1| -atom:wordalice

2| -atom(100):gpuheavy

3| -atom(100):opencltests

4| -atom(100):publisherintensive

5| -atom(500):wordintensive

6| -atom(200):excelintensive

These sample.txt items demonstrate entries within the file. The line numbers are used for illustration here and SHOULD NOT be used in the actual sample.txt file (each line should begin with just the “-atom:wordalice” portion with nothing else preceding or following, for example).

In this example, these items relate to fictional OPBM Atoms, named internally:

0: Word Alice

1: GPU Heavy

2: OpenCL Tests

3: Publisher Intensive

4: Word Intensive

5: Excel Intensive

To use the input file, prefix its name with an @ (at sign) on the command line:

**java -jar opbm.jar @c:\path\to\sample.txt**

OPBM will interpret the input from the sample.txt file as though the user repeatedly typed entries on the command line, one after the other. They will be added directly in sequence as they appear on the command line.

A combination of command-line options and command-line input files can thusly be specified, allowing for a concatenation of both directly specified entries on the command line, as well as input coming from a file. There are no limits to how many entries can be specified in this way.

**This syntax appears as these three examples:**

**java -jar opbm.jar -atom:wordalice @sample.txt**

**java -jar opbm.jar @sample.txt -atom(100):wordalice**

**java -jar opbm.jar -atom:wordalice @sample.txt -atom(100):wordalice**

As you can see, the command line interface is very flexible and powerful. Future versions will extend these features to executing all known items to OPBM:

**Atoms** - Smallest workload

**Molecules** - Groups of similar Atoms into defined workloads

**Scenarios** - Groups of Molecules into related scenarios

**Suites** - Entire benchmark suites

Future versions will also allow these additional command line options to be grouped, in any combination, so they can also be specified alongside input files:

**java -jar opbm.jar -molecule:name -scenario:name -suite:name -atom:name @file**

**Output Files and Processing**

As of the first June 27, 2011 release, continued through the August 22, 2011 release, OPBM internally recognizes four types of line items written to stdout or stderr. These are reported by the scripts executing, and are used to update the heads-up-display, and to capture some timing data for debugging:

**timing** Conveys timing information to OPBM. Must have the form “Workload description,timing,percentage”. Example: “Launch Microsoft Word,1.5733983892,89.8329821602”. Appears in blue.

**debug** Conveys debug information (shows up in the debug portion of the heads-up-display, which are the bottom 4 lines).

**status** Conveys status information (shows up in the status portion of the heads-up-display, which are the two lines above the middle gray portion).

**error** Conveys error information, which is usually terminal. In future versions, if the keyword “terminate” is found on the error string, OPBM will automatically terminate the current benchmark test and continue with any more that are scheduled. Appears in red.

OPBM records and accumulates everything generated from executing scripts in a single XML file called **results\_Mmm\_dd\_yyy\_at\_hh\_mm.xml**. This file is written out to the a relative location bsed on the current user, such as **C:\Users\username\Documents\opbm\results\xml\**.

In addition, OPBM automatically sums up run data from *timing* entries encountered while processing stdout or stderr, and creates CSV files in the *timing* format. These files appear in the **results\xml** directory stated above, with CSV files being deposited in **..\csv\** relative to the xml directory.

The scripts themselves may also write content directly, though this is not required. A directory has been established for script output as well:

**C:\Users\username\Documents\opbm\scriptOutput\**

OPBM records each timing line as it was generated. If multiple instances of the same test were run, then OPBM automatically averages the results. OPBM automatically appends a “Total” line to the reported timing lines as well.

**Note:** At present, iteration data is stored and processed separately, meaning each execution by iteration produces its own separate set of summation values, which are recorded as “executable\_name(0001of0010).csv” in the **opbm\results\csv\** directory.

This has been done purposefully because OPBM contains internal flow control logic which allows single tests to be repeated via OPBM’s Flow Control directives. It was determined that command-line iterations are desirable for generating volumes of separate data points to be analyzed.

Some of OPBM’s flow control directives are common programming features, such as for..next, do..while, repeat..until, while..whileEnd, etc. To record single iterative tests as an averaged output, either repeat the test within the sequence of operations explicitly, or use flow control directives to loop the script within OPBM itself.

**Note:** As of the August 16, 2011, only the **if..elseif..else..endif** Flow Control directives do not function. Everything else (except user prompts and rebooting) have been validated and are functional.

**Future Releases**

OPBM is still under development. There are bound to be unknown bugs encountered. *Cana Labs* will soon implement a bug/issue tracking feature on a GitHub repository that various key people will have access to. Bugs will be able to filed and tracked as encountered, with the authors regularly fixing those bugs, some as zero-day fixes.

Until the issue tracker is setup, please email all bugs with pertinent information (screenshots, descriptions of sequence of events leading to failure) to:

Van Smith, [van@canalabs.com](mailto:van@canalabs.com)

Rick C. Hodgin, [rick@canalabs.com](mailto:rick@canalabs.com)

**Service Releases**

Service releases will be made for critical bugs. Otherwise, bug-fix releases will be scheduled at regular intervals.

Please contact for any GUI or function use questions:

Rick C. Hodgin

[rick@canalabs.com](mailto:rick@canalabs.com)

317-879-6374 (cell phone)

**Sample output.xml File**

This file was taken from a real run of the Dual Alice script, iterated two times. It produced this output.xml file, which contains a history of everything generated by the script.

*Copy and paste into a text file:*

<?xml version="1.0" encoding="utf-8"?>

<opbm>

<rawdata>

<run>

<atom name="IE Google V8">

<abstract name="Google V8" sourcename="execute">

<start date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791255"></start>

<command>..\autoIT\ie\googlev8\ieGoogleV8.exe</command>

<errors></errors>

<finish date="Tue Aug 16 16:40:19 CDT 2011" millisecond="1313530819362" result="success"></finish>

<process date="Tue Aug 16 16:40:19 CDT 2011" millisecond="1313530819362">Terminated with code 0</process>

<outputs>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791470">debug,Starting up IE Google V8</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791470">debug,InitializeGlobalVariables()</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791471">debug,Plugin C:\cana\java\autoIt\ie\googlev8\..\..\common\opbm\dll\opbm.dll opened properly</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791471">debug,Detected Windows 64-bit Operating System</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791471">debug,InitializeIEScript()</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791670">debug,LaunchIE()</output>

<output date="Tue Aug 16 16:39:51 CDT 2011" millisecond="1313530791670">debug,Attempting to launch C:\Program Files (x86)\Internet Explorer\iexplore.exe</output>

<output date="Tue Aug 16 16:39:52 CDT 2011" millisecond="1313530792742">timing,Launch Internet Explorer, 1.06979634420796, 57.0201985922494</output>

<output date="Tue Aug 16 16:39:52 CDT 2011" millisecond="1313530792841">debug,TypeGoogleV8URL()</output>

<output date="Tue Aug 16 16:39:52 CDT 2011" millisecond="1313530792841">debug,Open</output>

<output date="Tue Aug 16 16:39:54 CDT 2011" millisecond="1313530794916">timing,Type URL to Google V8 benchmark, 2.07112948090135, 100.428293797199</output>

<output date="Tue Aug 16 16:39:54 CDT 2011" millisecond="1313530794916">debug,RunGoogleV8()</output>

<output date="Tue Aug 16 16:40:12 CDT 2011" millisecond="1313530812950">timing,Run Google V8, 18.0049044784975, 76.4236212218342</output>

<output date="Tue Aug 16 16:40:13 CDT 2011" millisecond="1313530813050">debug,CloseIE()</output>

<output date="Tue Aug 16 16:40:13 CDT 2011" millisecond="1313530813153">timing,Close Internet Explorer, 0.102918426474398, 106.880763501922</output>

<output date="Tue Aug 16 16:40:13 CDT 2011" millisecond="1313530813153">debug,FinalizeScript()</output>

<output date="Tue Aug 16 16:40:13 CDT 2011" millisecond="1313530813154">status,TimerFinish: Total Runtime,21.6842601245333</output>

<output date="Tue Aug 16 16:40:13 CDT 2011" millisecond="1313530813155"></output>

</outputs>

</abstract>

</atom>

<atom\_failures name="IE Google V8"></atom\_failures>

<results name="IE Google V8.Google V8">

<timing>Launch Internet Explorer, 1.06979634420796, 57.0201985922494</timing>

<timing>Type URL to Google V8 benchmark, 2.07112948090135, 100.428293797199</timing>

<timing>Run Google V8, 18.0049044784975, 76.4236212218342</timing>

<timing>Close Internet Explorer, 0.102918426474398, 106.880763501922</timing>

<timing>Total,21.24874873008121,82.69952068226023</timing>

</results>

</run>

</rawdata>

<resultsdata>

<result datetime="Aug 16, 2011 at 04:39PM" name="OPBM Benchmark" shortname="unnamed" tags="" tested="yes" status="success" score="0">

<suite name="Placeholder Suite" shortname="phS" tags="" tested="yes" status="success" score="0">

<scenario name="Placeholder Scenario" shortname="phS" tags="" tested="yes" status="success" score="0">

<molecule name="Placeholder Atom" shortname="phA" tags="" tested="yes" status="success" score="0">

<atom score="0" name="Google V8" status="success">

<worklet name="Launch Internet Explorer" shortname="LIE" timing="1.06979634420796" score="57.0201985922494" tested="yes" status="success"></worklet>

<worklet name="Type URL to Google V8 benchmark" shortname="TURLtGV8b" timing="2.07112948090135" score="100.428293797199" tested="yes" status="success"></worklet>

<worklet name="Run Google V8" shortname="RGV8" timing="18.0049044784975" score="76.4236212218342" tested="yes" status="success"></worklet>

<worklet name="Close Internet Explorer" shortname="CIE" timing="0.102918426474398" score="106.880763501922" tested="yes" status="success"></worklet>

</atom>

</molecule>

</scenario>

</suite>

</result>

</resultsdata>

</opbm>

**Sample CSV Output Files**

IEGoogleV8.csv:

***Launch Internet Explorer****,* ***1.06979634420796****,* ***57.0201985922494***

*Type URL to Google V8 benchmark, 2.07112948090135, 100.428293797199*

*Run Google V8, 18.0049044784975, 76.4236212218342*

*Close Internet Explorer, 0.102918426474398, 106.880763501922*

*Total Runtime,21.6842601245333*

**Legend:**

**Name of Worklet**

**Time (in seconds) to execute Worklet**

**Relative Percentage to Reference Machine**

**Note:** In this example, OPBM is not using final timing values. The percentages shown may not be accurate. All scripts are provided for reference only, and will be further developed in the future.