Standard Performance Evaluation Corporation (SPEC)

Chauffeur™ Worklet Development Kit (WDK) User Guide 2.0.0

7001 Heritage Village Plaza, Suite 225
Gainesville, VA 20155,
USA

SPECpower Committee

Table of Contents

SVN Revision: 1132

SVN Date: 2017/03/09 04:28:34

1. Overview

1.1. Summary

When SPEC developed SPECpower_ssj2008, the first industry-standard benchmark for measuring energy efficiency of servers, they also created the SPEC Power and Performance Benchmark Methodology to describe best practices for future benchmarks and tools developed to measure power and performance of computer systems. As SPEC set out to create the Server Efficiency Rating Tool (SERT), they recognized that many of these best practices were independent of the actual workload being measured.

The Chauffeur framework was designed to simplify the development of workloads for measuring both performance and energy efficiency. Chauffeur contains functionality that is common to most workloads, enabling developers to focus on the actual business logic of the application, and take advantage of Chauffeur's capabilities for configuration, execution, data collection, validation, and reporting.

Chauffeur was initially designed to meet the requirements of the SERT. SPEC recognized that the framework would also be useful for research and development purposes and is now being made available as the Chauffeur Worklet Development Kit (WDK). This kit can be used to develop new workloads, or "worklets" in Chauffeur terminology. Researchers can also use the WDK to configure worklets to run in different ways in order to mimic the behavior of different types of applications. These features can be used in the development and assessment of new technologies such as power management capabilities.

1.2. What's New

There are a variety of enhancements and bug fixes in this release of the Chauffeur WDK, described in the subsections below. In general, worklets, listeners, and configuration files that were developed for Chauffeur WDK 1.x are compatible with Chauffeur WDK 2.0.0. Reporter stylesheets, validators, and extensions to Chauffeur may require updates for compatibility with Chauffeur WDK 2.0.0. In particular, the format of the results.xml file has changed slightly, with the results of the run encapsulated in a `<run-data>` element; reporter stylesheets and validators will need to be updated to expect this additional element.

The standard Chauffeur WDK reports include an Efficiency Score for each load level (measurement interval) but do not report overall scores for a worklet, workload, or suite. However, the `<summary>` section in the results.xml file does include summarized scores. The calculation of these scores has changed in Chauffeur WDK 2.0.0, so these values will not be comparable between Chauffeur WDK 2.0.0 and previous versions of the Chauffeur WDK. Worklet developers are encouraged not to rely on these summarized worklet, workload, or suite scores as the calculations may change again in the future; instead, if a composite score is needed, a custom reporter stylesheet can be used to calculate the score from the load level results.

The Chauffeur WDK 2.0.0 requires Java 7 or higher. Chauffeur WDK 1.x supported Java 6.

1.2.1. Platform Support

The platform support code in Chauffeur provides the platform-specific code needed for running Chauffeur on a variety of platforms.

Enhancements

- Add support for Ubuntu 14.04 LTS and Ubuntu 16.04 LTS [C545]
- Add support for AMD Opteron A1100 series of AArch64 microprocessors

Bug Fixes

- Linux: Handle cases where lscpu includes empty fields [C561]

1.2.2. Discovery

The Chauffeur Discovery scripts gather information about the System Under Test (SUT) and store it in the result output file.

Enhancements

- A new Groovy-based discovery mechanism was implemented for Chauffeur WDK 2.0.0. This change generally won't be noticed by end users, but offers several benefits to developers enhancing the discovery code. [C556]
- Discovery for Linux and AIX has been migrated to the new design. Other platforms continue to use the original discovery implementation. [C556, C564]

Bug Fixes

- In rare cases, discovery data may have been truncated [C557]
- Enhance JVM detection to distinguish Oracle HotSpot from OpenJDK [C559]
- AIX: the CpuCores values from discovery included extra whitespace [C541]
- AIX: the ActiveCores value from discovery was not used properly [C541]
- Linux: network speed reported incorrectly with 10Gbit adapters [C575]
- Linux: fix detection of SLES versions
- Linux: discovery support for SLES 12
- Linux: network discovery fails on device names not starting with 'e' [C581]
- Linux: use parted instead of fdisk where available
- Solaris: discovery was broken for versions of Perl later than 5.18 [S554]
- Windows: discovery may fail due to the PowerShell execution policy [S552]
- Miscellaneous fixes to improve the accuracy of discovery data

1.2.3. Reporter

The Reporter generates HTML, text, and CSV reports from a results.xml file.

Enhancements

- Added several new options to change the appearance of graphs [S597]:
  - 'show3d' enables a 3-D effect on bar graphs
  - 'category-margin' adjusts spacing between bars
  - 'hide-legend' option hides the legend
  - Y-axis 'location' changes which side of the graph where the axis appears
  - 'thickness' sets the line thickness in a line graph
- Added support for graph subtitles [S597]
- Allow charts and table titles to link to the glossary document [S599, S607]
- Use a common set of scales for graphs to make visual comparisons easier [C584, C590]

Bug Fixes

- Manual runs of the reporter may have failed due to an incorrect classpath [C542]
- Reporter exceptions when availability dates used an incorrect format [C548]
- CSV output didn't quote data with embedded commas or other problematic characters [C553]
- Change the implementation of the geomean function to be less susceptible to overflow [S599]
- "Total Memory available to OS" was incorrect for multi-node runs [S515]
- PTDaemon Version was not filled in properly [C544]
- Show more precision for efficiency scores less than 1.0 [C590]
- Avoid cropping range marker text and make the background semi-transparent [S603]
- Make error and warning message formats more consistent [C560, S612]
- Avoid copyright character in CSV reports [C587]

1.2.4. Result Output

The primary output of a Chauffeur WDK run is the results.xml file. As described above, the format of the results.xml file has changed slightly in this release. The run results (but not configuration information) are now encapsulated in a `<run-data>` element. In addition, the Client environment variables and Java properties are now included in the results.xml file only if they differ from the environment variables and Java properties from the Host.

Enhancements

- Update the RawFileWriter to generate XML using StAX [C582]
- Eliminate redundant Client info in results.xml [C423]
- Change worklet efficiency scores to use the geometric mean of interval efficiency [S591]
- Improve performance and memory usage of results signing [C527]
- The SuiteDescription can now include reference scores for normalizing the performance of each worklet [S610]

Bug Fixes

- Incorrect host id in results.xml [C563]
- Change geometric mean calculations to be less susceptible to overflow [S591]

1.2.5. Development/Build/Test

These features have no impact on users, but may affect developers.

Enhancements

- Improved documentation of RunListeners [C551]
- Make JUnit test failures cause the build to fail [C554]
- Automate collection of discovery data for testing

Bug Fixes

- Fix JavaDoc warnings [C576, C579]
- Use replacement rather than concatenation to generate validator messages
- Fix LinuxEnvironmentTest to read /proc/meminfo output regardless of JVM bitness [C555]

1.2.6. Miscellaneous

Enhancements

- Add support for handling time-limited trial versions of suites
- Log the full PTDaemon Identify string in results.xml
- Report an error when using mismatched versions of the Chauffeur WDK
- Update open source packages to recent versions
- Added a configurable timeout for connecting to Director [S558]
- Make Director connect to Hosts more quickly by running discovery in advance [C573]
- Experimental support for automatic client configuration matching [C571]
- Write log files to a separate logs directory
- A getIntervalDef method was added to the Interval interface [C589]
- A getCalibrated method was added to the IntervalDef class [C589]
- A preferred abbreviation was added to Size/SizeUnit output [S588]
- Run pre-run validation checks on the Hosts as well as Director [S577]
- The latest version of PTDaemon is included in the WDK

Bug Fixes

- Change the behavior of the client connection timeout to better support large numbers of clients with slow initialization times [S574]

- The FixedIterationsDirectorSequence was broken because it did not send an intervalResults command [C549]
- The DirectorCommunicator clientName has been updated to reference the host information [C574]
- An error is now generated if the Chauffeur configuration attempts to instantiate an abstract class [C558]

1.3. Using This Document

This User Guide is intended for both novice and experienced Chauffeur users. It provides instructions for installing the Chauffeur WDK and running initial tests with the sample worklets included in the kit. It also describes more advanced configuration, along with basic information for creating new worklets.

For the most basic Chauffeur hardware measurement setup, one of each of the following is required:

- System under Test (SUT): the actual system for which the measurements are being taken
- Controller (e.g. a server, desktop PC, or laptop): the system to which the power analyzer, temperature sensor, and SUT are connected
    - The Controller and SUT are connected to each other via an Ethernet connection.
    - The analyzer and temperature sensor are connected to the Controller via device specific means.
- Power analyzer and temperature sensor [optional]:
    - A power analyzer is necessary for measuring energy usage during the test, and a temperature sensor is needed for measuring the ambient temperature. These devices are not required for running performance-only tests. It is also possible to use simulated "dummy" devices for testing purposes.
    - The power analyzer is connected to the Controller and used to measure the power consumption of the SUT, whilst the temperature sensor is also connected to the controller and used to measure the ambient temperature in front of the SUT's main airflow inlet.
    - **Note:** Before connecting the power analyzer and the temperature sensor to the Controller and the SUT, please read the "Power and Temperature Measurement Setup Guide" in the Chauffeur doc directory carefully! The latest version of this document can be found on the SPEC web site:

      http://www.spec.org/power/docs/SPEC-Power_Measurement_Setup_Guide.pdf.

      This practical guide explains how to set up and run various power analyzers and temperature sensors with the SPEC PTDaemon. Additionally, it describes the configuration recommendations for obtaining accurate data.
    - **Note:** This release of the Chauffeur WDK is able to measure only the energy usage of servers running on AC power. The Chauffeur WDK can run on DC powered servers, but DC power measurement is not supported.

The Chauffeur WDK is composed of several elements including:

- The test harness (called Chauffeur): handles the logistical side of measuring and recording power data along with controlling the software installed on the SUT and controller system itself
- The Director: instructs the SUT to execute the workload
- The workload (a set of worklets, which are small, self-contained work items that may be run individually or collectively): exercises the SUT while the test harness collects the power and temperature data. The Chauffeur WDK includes a set of sample worklets, and allows for the development of new worklets.
- The SPEC Power & Temperature Daemon (PTDaemon): connects to the power analyzer and temperature sensor and gathers their readings while the workload executes
- The Reporter: gathers the environmental, power, and performance data after a run is complete and compiles it into an easy-to-read format.

All of these components are included in the Chauffeur WDK distribution.

The sample worklets included in the Chauffeur WDK are provided for demonstration purposes only, and are not suitable for measuring the energy efficiency of systems.

1.4. Configuration Requirements

The hardware and software requirements for running the Chauffeur WDK depend on the worklets and configuration. The following are recommended minimum requirements; it may be possible to run in smaller environments for some configurations.

- System Under Test (SUT) – one or more host systems, each with the following:
  - Hardware requirements
    - One or more CPUs. The Chauffeur WDK has been tested on recent x86_64, POWER, and SPARC processors. Other processors are likely to work as long as a compatible Java Virtual Machine is available.
    - At least 4 GB of RAM. Memory requirements tend to increase with larger numbers of logical processors, so additional memory may be required for many SUTs.
    - A minimum of 1 GB free storage space
    - At least one Network Interface Controller (NIC)
  - Software requirements
    - A single operating system per host system. Most testing of the Chauffeur WDK has been on Microsoft Windows Server 2008 R2 SP1, Windows Server 2012 R2, Red Hat Enterprise Linux 7.3 (or higher), AIX 7.1, and Solaris 11.
    - Limited testing has been performed using multiple guest operating systems under control of a hypervisor; these configurations are likely to be functional, but interactions among the guests may cause unexpected results.
    - A Java SE 7 (or higher) runtime environment. The Chauffeur WDK has been tested primarily with Java SE 8 runtime environments from Oracle and IBM. Other implementations are likely to work as long as they are fully compatible with the Java specifications.
  - In multi node test setups, it is recommended that all nodes have exactly the same hardware and software components. Configurations with heterogeneous nodes may work, but have not been extensively tested.
- Controller
  - Hardware requirements
    - A hardware configuration capable of running a supported OS and JVM combination.
    - At least one NIC
    - If a power analyzer and temperature sensor is to be used, the controller system must have external ports (such as RS232, USB, or GPIB) that are supported by PTDaemon for these devices. See the PTDaemon Accepted Devices List for details: http://www.spec.org/power/docs/SPECpower-Device_List.html
  - Software requirements
    - Any operating system that supports the required Java runtime. If a power analyzer or temperature sensor is to be used, PTDaemon requires x86 hardware running either Microsoft Windows or Linux.
    - A Java SE 7 (or higher) runtime environment.
    - Vendor specific drivers supporting special power analyzer and/or temperature sensor connectors, if required.
- Power analyzer requirements (only required for measuring power consumption)
  - At least one power analyzer from the list of accepted measurement devices http://www.spec.org/power/docs/SPECpower-Device_List.html
  - In order to ensure data is accurate, the power analyzer should have been calibrated within the past year.
- Temperature sensor requirements (required only for measuring temperature)

- At least one temperature sensor from the list of accepted measurement devices http://www.spec.org/power/docs/SPECpower-Device_List.html

The SPEC PTDaemon includes support for additional devices that are not on the list of accepted devices. Use of these devices is not recommended, as they had limited testing and generally do not meet the accuracy requirements set out by SPEC.

Figure 1 shows the system overview diagram of a Chauffeur test environment:

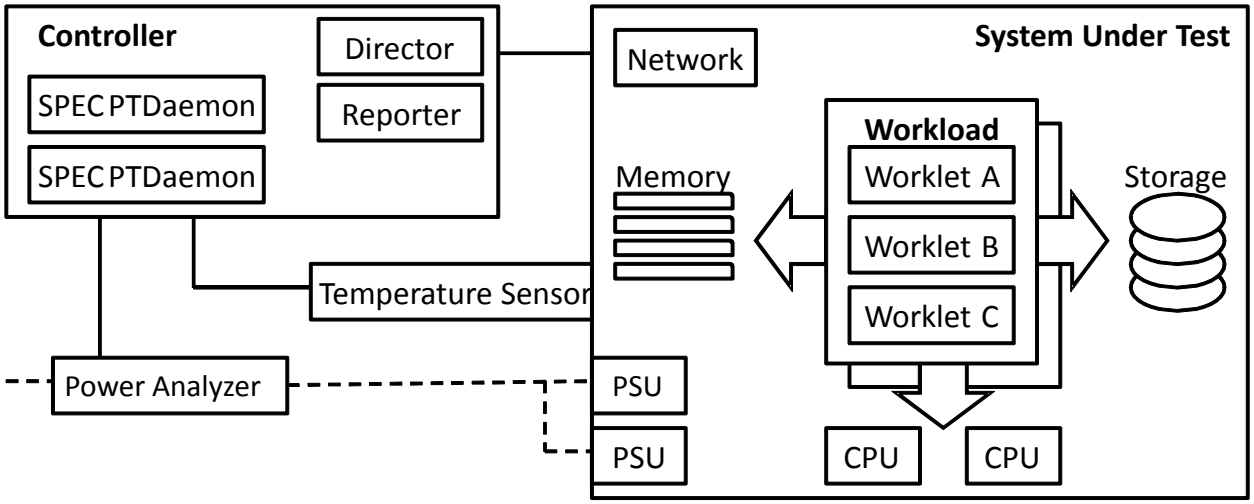

Figure 1 – Logical components of the Chauffeur test environment

1.5. Chauffeur WDK Media Content

The Chauffeur WDK is distributed in both zip and tar.gz formats. Both formats contain the same files. The zip archive is recommended for Microsoft Windows platforms. The tar.gz archive uses UNIX-style line-ending characters for text files, and has the executable bit set for executable files; this is recommended for all other platforms.

Extract the archive using the appropriate tools for your operating system. This will result in the following directory structure:

```
└──ChauffeurWDK-2.x.y
   ├── discovery
   ├── doc
   │   └── apidoc
   ├── lib
   ├── logs
   ├── PTDaemon
   ├── redistributable_sources
   ├── results
   └── src
```

This document can be found in the "doc" directory, while the various script and batch files discussed in the following section are present in the top-level ChauffeurWDK directory or the PTDaemon sub-directory.

The ChauffeurWDK files need to be extracted on the Controller system and each SUT.

2. Chauffeur Command Line Usage

2.1. Chauffeur Configuration and Start Procedure

This section provides details to enable a successful test with a simple server configuration. Further sections provide additional details of the configuration settings required to perform a successful test. Please note that in the following instructions there are typically two script file options: *.bat for the Windows OS and *.sh for UNIX and Linux OSes.

- If power and/or temperature will be measured, configure the power analyzer and/or temperature sensor. By default, the PTDaemon scripts are configured to use a "dummy" power analyzer and temperature sensor. These dummy devices support the PTDaemon command interface but produce simulated data. These dummy devices may be used if real devices are not available. To configure PTDaemon for actual devices, edit the power analyzer PTDaemon (`PTDaemon\runpower.bat` or `PTDaemon/runpower.sh`) and temperature sensor PTDaemon (`PTDaemon\runtemp.bat` or `PTDaemon/runtemp.sh`) script files on the controller system (the system that the devices are physically connected to). Ensure the proper communication ports and network ports are used.
    - For each additional power analyzer: Create a copy of `runpower.bat` or `runpower.sh` and ensure that the proper communication ports and network ports are set as described in the respective files.
- If power and/or temperature will be measured, start an instance of PTDaemon for each power analyzer (`runpower.bat/runpower.sh`) and temperature sensor (`runtemp.bat/runtemp.sh`). The scripts should be started on the controller system.
- If power or temperature will not be measured (with either real or dummy devices), edit the `listeners.xml` file (in the `ChauffeurWDK-2.x.y` directory on the controller system). For each PowerAnalyzerListener or TemperatureSensorListener in the file, set `<listener enabled="false">`.
- Optional: Edit the `config.xml` file (in the `ChauffeurWDK-2.x.y` directory on the controller system). The default file can be used without modification for a simple test run on most systems. The configuration can be customized by adjusting interval lengths, the number of intervals, the set of worklets to be run, etc.

  To change the configuration of the client JVMs, find the `<client-configuration>` / `<clients>` section in `config.xml`. The number of clients (JVM instances) can be set using the `<count>` element. This can be set to an absolute number (e.g. `<count>4</count>`) or the count can be a JavaScript expression which will determine the client count at runtime based on the characteristics of the system. This expression can make use of the variables:

  - `logicalCores`
  - `physicalCores`
  - `numaNodes`
  - `physicalMemoryBytes`
  - `paramMap` (worklet parameters)

  For example: `<count>logicalCores / 2</count>`

The <option-set> section can be used to specify Java command-line parameters to be used when launching the clients. For example, with HotSpot the following configuration would use the HotSpot Server JVM:

```
<option-set>
    <parameter>-server</parameter>
</option-set>
```

Note: heap parameters (-Xms and -Xmx) will be set automatically by Chauffeur, and should not be included here. On Solaris, it may be necessary to use <parameter>-d64</parameter> to force the use of a 64-bit JVM.

- Optional: Edit the test-environment.xml file (in the ChauffeurWDK-2.x.y directory on the controller system), describing the hardware and software details of the test setup. The default values in this file need to be changed to match the real test environment.
    - For each additional power analyzer, add the following section to the test-environment.xml in the <MeasurementDevices> section and change the default values to match the real test environment.

```
<PowerAnalyzer>
  <PTDaemonHostname>localhost</PTDaemonHostname>
  <PTDaemonPort>8888</PTDaemonPort>
  <HardwareVendor>_Energy Minder, Inc.</HardwareVendor>
  <Model>_EM1000+ USB</Model>
  <SerialNumber>_ser001122</SerialNumber>
  <Connectivity>_USB2</Connectivity>
  <InputConnection>_Default</InputConnection>
  <CalibrationInstitute>_NIST</CalibrationInstitute>
  <AccreditedBy>_IQ2 Calibration Laboratory</AccreditedBy>
  <CalibrationLabel>_N-32768</CalibrationLabel>
  <DateOfCalibration>2010-01-01</DateOfCalibration>
  <SetupDescription>_Unknown</SetupDescription>
</PowerAnalyzer>
```

    - For multi-node environments: The node quantity needs to be set. An example for a 3 node environment:

```
<Node>
  <Quantity>3</Quantity>
```

- If the default port numbers in the PTDaemon script files have been changed, the listeners.xml file (in the ChauffeurWDK-2.x.y directory on the controller system) must be changed accordingly. Also any range settings for the power analyzer can be specified in the same section of the file.
    - For each additional power analyzer, add the following section to the listeners.xml and change the default values to match the real test environment (**Note:** Do not forget to change the port #).

```
<listener enabled="true">
  <type>PowerAnalyzerListener</type>
  <classpath>
    <path>lib/ptdaemonClientApi.jar</path>
  </classpath>
  <parameters>
    <parameter name="hostname">localhost</parameter>
```



```
    <parameter name="port">8888</parameter>
    <parameter name="voltage-range">
     <range-set>
      <default-range>none</default-range>
     </range-set>
    </parameter>
    <parameter name="current-range">
     <range-set>
      <default-range>none</default-range>
      <range level="87.5%">none</range>
      <range level="75%">none</range>
      <range level="62.5%">none</range>
      <range level="50%">none</range>
      <range level="37.5%">none</range>
      <range level="25%">none</range>
      <range level="12.5%">none</range>
     </range-set>
    </parameter>
   </parameters>
</listener>
```

- On the system under test, edit the Chauffeur host script file (`host.bat/host.sh`) for the appropriate system configuration. For single runs, the `KEEPALIVE` variable can be cleared; for multiple consecutive runs, it should remain at the default setting: `-keepalive`.

  Check the following specifications for proper values matching the test system configuration:

  ```
  JAVA = <for a non-default Java Virtual Machine (JVM), change the java command to
  the pathname of the java command you want to run>
  ```

  The `host.sh/.bat` script files both include the default definition:

  ```
  JAVA=java
  ```

- On the system under test, call the `host.bat/.sh` script file.

- On the controller system, edit the `director.bat/.sh` script files. First, specify the host IP address of the SUT by editing the following line:

  ```
  HOSTS=_hostname_or_IP_of_Host_systems_
  ```

  For multi-SUT environments, add each SUT hostname or IP address (separated by a comma) to the HOST argument. **Note:** Do not use a space behind the comma(s). An example for 3 SUTS:

  ```
  HOSTS=10.10.10.1,10.10.10.2,10.10.10.3
  ```

  Also set `JAVA` as you did on the SUT in `host.bat/.sh`

- On the controller system, call the `director.bat/.sh` script file to start the run.

- The duration of the whole test sequence depends on the number of test cases defined above. The result files for each Chauffeur WDK test run (`results.html` and `results.xml`) are stored in a separate subdirectory within `results`, with a name beginning with `chauffeur-`*`nnnn`* e.g. `chauffeur-0000`.

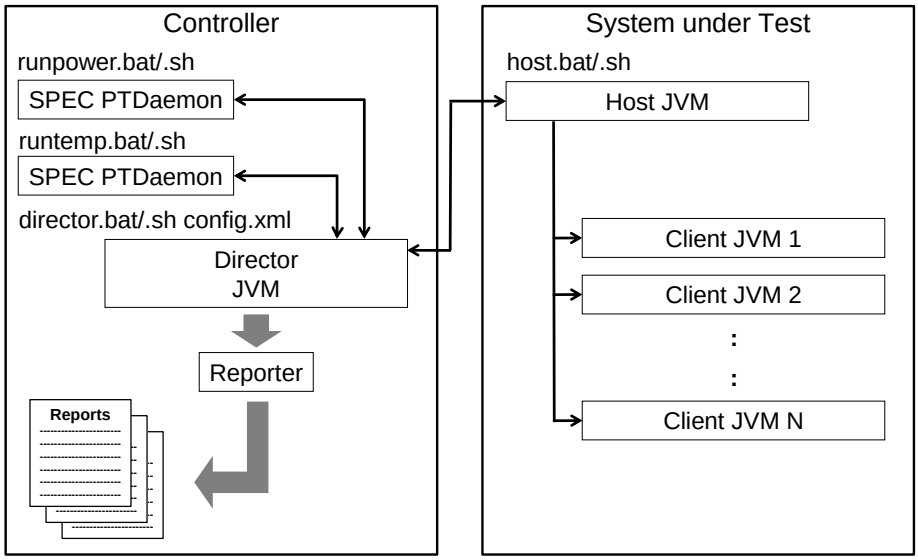

Figure 2 – Chauffeur Startup Procedure

2.2. Generate report files with the reporter scripts

In some cases the user might want to create a new .html, .csv, or .txt report from a Chauffeur xml output file, e.g., after editing user-modifiable fields (configuration information in the TestEnvironment section of `results.xml`).

The reporter can be run on the controller system or any other system where the Chauffeur WDK has been installed. You may need to edit the `reporter.bat/.sh` script to set the `JAVA` environment variable, as you did in the host and director scripts. The syntax of the reporter command is:

```
        reporter.bat [options] {for Windows-based system}
        reporter.sh  [options] {for Linux/UNIX-based system}

Required option:
-r <filename>    A results.xml file from a Chauffeur run
Other options are:
-a               Generate all report formats (HTML, CSV, and Text)
                 default: generate only HTML output

-c               Generate CSV report files

-o <outfile>     Output file for generated report
                 default: match the filename of the raw file, with a .txt,
                 .csv, or .html extension (to match the report type).  May
                 not be used with the -a option

-p               Direct XSL transform (must be used with -x)
                 default: generate a report from the transformed output

-s               Skip validation. Reports will automatically be marked invalid

-t               Plain ASCII text output
                 default: generate HTML output

-x <xslfile>     Use a non-default XSL file to generate the report

Example: reporter.bat –a –r results/chauffeur-0002/results.xml
```


3. Power Analyzer Range Settings

Most power analyzers can be set to multiple current ranges. In order to achieve accurate results, the range settings on the device must be set appropriately for the load being measured. Most devices also support an auto-ranging mode which will adjust the device automatically as the load changes. Use of this mode is not recommended for measurement because it takes time for the range to adjust (during which the measured data may not be accurate), and because SPEC PTDaemon cannot determine the accuracy of the data while the device is in auto-ranging mode.

The Chauffeur WDK allows you to configure separate ranges for each power analyzer and can adjust the ranges as needed for each measurement interval. These settings appear in the `listeners.xml` file on the controller system. Each PowerAnalyzerListener includes "`voltage-range`" and "`current-range`" parameters. The default configuration includes multiple current range settings for different load levels, but all are set to the same value: "`none`". This indicates that no change should be made to the settings on the device.

To configure the ranges manually, replace each of these entries with a value appropriate for your system under test. The range setting should generally be slightly higher than the maximum expected reading during that interval. Each model of power analyzer will support a specific set of ranges; however, the configuration can use any value, and SPEC PTDaemon will round the range settings up to the next supported range.

The range settings may need to be adjusted when making changes to the SUT, to the worklets being tested, or changes to the configuration of the operating system or Chauffeur. The validation performed at the end of the run will determine if power measurement accuracy thresholds are not being met. If your result includes warnings regarding the power analyzer uncertainty, adjusting the current ranges may resolve the issue.

3.1. Automating Range Settings

The Chauffeur WDK includes a process for automatically determining appropriate ranges for your configuration. To perform range setting automation:

1. Edit `listeners.xml` on the controller system and ensure that the analyzer is set to auto-ranging mode or make an approximation at the appropriate ranges. The default `listeners.xml` specifies "none" as the range for each worklet. This tells Chauffeur not to change settings at runtime, so it will use the range the device is already set to.

    - If multiple power analyzers are in use, make sure to create a listener section for each device in `listeners.xml`. This process is described in section 2.1.

2. Perform a run using the same steps described in section 2.1.
3. At the end of the run, the name of the results file will be printed:

    e.g. `results/chauffeur-0000/results.xml`

4. Use this filename and run the following command on the controller system:

```
reporter.bat –s –p -r results/chauffeur-0000/results.xml -x
org/spec/chauffeur/reporter/resources/rangeSettings.xsl -o
rangeSettings.txt
-or-
./reporter.sh –s –p -r results/chauffeur-0000/results.xml -x
org/spec/chauffeur/reporter/resources/rangeSettings.xsl -o
rangeSettings.txt
```

This will generate a `rangeSettings.txt` in the same directory as `results.xml`. This file will contain an XML listener specification for each power analyzer used in the range setting run. Open the file in a text editor and follow the included instructions, which will be to copy some text from that file to the `listeners.xml`.

4. Developing with the Chauffeur WDK

Source code for the Chauffeur WDK is provided mainly as an aid for understanding how to implement new worklets or extend Chauffeur functionality. It may be helpful to import the Chauffeur source code into your development environment while developing code based on Chauffeur. For certain types of extensions, modifications to Chauffeur itself may be necessary. The Chauffeur code can be recompiled within your IDE. Ant scripts are also provided.

4.1. Developing with Eclipse

The Chauffeur code should be straightforward to import into any Java development environment. As an example, this section describes the steps for importing the code into Eclipse, using Eclipse 4.6.2 (Neon.2). Some familiarity with Eclipse is assumed.

1. Create a new Eclipse workspace. An existing workspace can also be used.
2. From the Java Perspective, select **File > New > Project…** and choose **General > Project**. Name the project `ChauffeurLibraries`. Press **Finish**.
3. Right-click on the new ChauffeurLibraries project and choose **Import…** Select **General > File System**. Browse for the directory `ChauffeurWDK-2.x.y/src/Chauffeur/ChauffeurLibraries` and press **OK**. Check the box next to `ChauffeurLibraries` and press **Finish**.
4. Select **File > New > Java Project**. Name the new project `PTDaemonClientApi`. Select "Create separate folders for source and class files". Click "Configure default…", change the Source folder name to `src/java`, and press **OK**. Press **Next**. Click on "Create new source folder" and add the folder `src/test`. Press **Finish** (to accept the new source folder), and then press **Finish** again (to finish creating the new project).
5. Right-click on the newly created project and choose **Import…** Select **General > File System**. Browse for the directory `ChauffeurWDK-2.x.y/src/pwrd/PTDaemonClientApi` and press **OK**. Check the box next to PTDaemonClientApi and press **Finish**.
6. Repeat the two previous steps to create and import additional Java projects as listed below. Note that it is not necessary to reconfigure the default source folder, but the `src/test` folder will have to be added for each project:

Project Name	Import Location
ChauffeurCommon	`ChauffeurWDK-2.x.y/src/Chauffeur/ChauffeurCommon`
Reporter	`ChauffeurWDK-2.x.y/src/Chauffeur/Reporter`
Chauffeur	`ChauffeurWDK-2.x.y/src/Chauffeur/Chauffeur`
ChauffeurTest	`ChauffeurWDK-2.x.y/src/Chauffeur/ChauffeurTest`

7. Right-click on the `Chauffeur` project and select **Properties**. Choose **Java Build Path**. In the **Projects** tab, press **Add…** and select the projects `ChauffeurCommon`, `PTDaemonClientApi`, and `Reporter`, and press **OK**. In the **Libraries** tab, press **Add JARs…** and add `ChauffeurLibraries/junit/junit-x.y.jar`, `ChauffeurLibraries/xml-security-x_y_z/xmlsec-x.y.z.`, `ChauffeurLibraries/commons-codec-x.y/commons-codec-x.y.jar`, and `ChauffeurLibraries/saxonHE9-x-y-zJ/saxon9he.jar` Press **OK** to add the jars, then press **OK** again to exit the Java Build Path dialog.

8. Repeat the previous step for the other projects listed below, adding the Projects and Libraries shown:

Project Name	Build Projects	Build Libraries
ChauffeurCommon	<none>	ChauffeurLibraries/junit/junit-*x.y*.jar ChauffeurLibraries/saxonHE9-x-y-zJ/saxon9he.jar ChauffeurLibraries/slf4j-x.y.z/slf4j-api-x.y.z.jar ChauffeurLibraries/slf4j-x.y.z/slf4j-jdk14-x.y.z.jar
ChauffeurTest	Chauffeur ChauffeurCommon Reporter	<none>
Reporter	ChauffeurCommon	ChauffeurLibraries/jfreechart/jcommon-*x.y.z*.jar ChauffeurLibraries/jfreechart/jfreechart-*x.y.z*.jar ChauffeurLibraries/saxonHE9-x-y-zJ/saxon9he.jar ChauffeurLibraries/slf4j-x.y.z/slf4j-api-x.y.z.jar ChauffeurLibraries/slf4j-x.y.z/slf4j-jdk14-x.y.z.jar

9. At this point, all of the code should compile without errors. There may be warnings for some classes, but these can be ignored.

4.2. Debugging Chauffeur Worklets in Eclipse

When developing new worklets, it is helpful to be able to debug them within the IDE. To run Chauffeur inside Eclipse you will need to create Run Configurations for Director and the host.

1. Start Eclipse, and switch to the Java perspective.
2. Expand `Chauffeur/src/java/org.spec.chauffeur.host`. Right-click on `Host.java` and select **Run As > Java Application**. On some platforms, a warning message may be generated regarding missing helper libraries. Press the stop button (red square in the Console view).
3. Expand `Chauffeur/src/java/org.spec.chauffeur.director`. Right-click on `Director.java` and select **Run As > Java Application**. The Director will fail with a usage message.
4. Choose **Run > Run Configurations…** Select the Host configuration. On the **Arguments** tab, enter the **VM arguments**:

   ```
   -Djava.library.path=${resource_loc:/Chauffeur/lib}
   ```

   Press **Apply**.
5. Still in the Run Configurations dialog, select the Director configuration. On the **Arguments** tab, enter the **VM arguments**:

   ```
   -Djava.library.path=${resource_loc:/Chauffeur/lib}
   ```

   and the **Program arguments**:

   ```
   -hosts localhost ${resource_loc:/ChauffeurTest/config.xml}
   ```

   The second parameter is the location of the Chauffeur configuration file you want to run – in this case, `config.xml` is a resource in the `ChauffeurTest` project. An absolute pathname can be selected also.

   Press **Apply**.

6. If your configuration file (e.g. `config.xml`) specifies a suite `<description>` class, add the project containing that class to the classpath for both the Host and the Director configurations. For example, the `ChauffeurTest/config.xml` listed above includes:

   ```
   <description className="org.spec.chauffeur.test.ChauffeurTest"/>
   ```

   In this case, select the Director Run Configuration, choose the **Classpath** tab, select "User Entries", press **Add Projects…** and add the ChauffeurTest project. Press **Apply**. Repeat this step for the Host configuration.

7. If your configuration file (e.g. `config.xml`) includes `<classpath>` entries, add an additional `<entry>` to reference the build path of the project containing these classes. This new entry should be relative to the `Chauffeur/Chauffeur` directory, since this is the working directory used by the Director and Host JVMs. This way the run can use the current classes built by Eclipse, without having to first rebuild jar files. For example, the `config.xml` in the ChauffeurTest sample project already includes `lib/chauffeurTest.jar` in its classpath entry; the line in bold below should be added to enable the worklet to find the files from the Eclipse workspace:

   ```
   <classpath>
       <entry>lib/chauffeurTest.jar</entry>
       <entry>../../Chauffeur/ChauffeurTest/build/classes</entry>
   </classpath>
   ```

8. If your listener configuration file (e.g. `listeners.xml`) uses a `PowerAnalyzerListener` or `TemperatureSensorListener`, you will need to either disable them (by adding `enabled="false"` to the listener definition), or launch PTDaemon instances. This can be done outside of Eclipse using the regular runpower and runtemp scripts. Or you can define an **External Tool** within Eclipse to launch PTDaemon.
9. At this point, you can launch the Director and Host JVMs using the Run Configurations dialog or the Run button in the toolbar. The two JVMs can be started in either order.
10. The Director and Host JVMs can also be launched with the Eclipse debugger. But when debugging worklet code, it is usually more interesting to run the Client JVM(s) in the debugger. To do this, edit the `config.xml`. Find the `<launch-definition>` for your worklet, and add:

    ```
    <debug-base-port>8000</debug-base-port>
    ```

    You can select any port that is available on your system. Choose **Run > Debug Configurations…** Right-click on **Remote Java Application** and choose **New**. Name the new configuration "Client", and enter the project "Chauffeur". In the **Connection Properties**, ensure that the **Port** matches the `debug-base-port` set above.

    If you do a run and the Host launches a new client, you can start the Client Debug Configuration to attach to the running client with the debugger. If there are multiple clients, they will use successive ports (8001, 8002, etc).

5. Known Issues

See the SERT Chauffeur WDK web page for information about any issues identified after release.

6. Trademark and Copyright Notice