Chauffeur: A Framework for Measuring Energy Efficiency of Servers – Part 2

Jeremy Arnold
arnoldje@us.ibm.com

IBM Corporation

International Conference on Performance Engineering 2014

Extending Chauffeur

Chauffeur was designed to be extensible:

- New worklets can be added without modification to Chauffeur itself
- Custom data collectors (listeners) can be used to collect additional data at runtime
- Report generation supports custom stylesheets to generate new reports with HTML/text or CSV output
- Runtime behavior can be modified through plugins

Agenda

1. Creating a New Worklet
   - Defining a Basic Worklet
   - Enhancing the Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Agenda

1. Creating a New Worklet
   - Defining a Basic Worklet
   - Enhancing the Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Worklet Basics

A worklet is primarily composed of a set of transactions.

Each transaction can accept some input, perform some action, and generate a result.

A User represents some user of the system (human or otherwise). Each time a transaction is executed, the User is passed to the transaction.

Steps to Create a Worklet

1. Define a new User, or choose an existing implementation
2. Create a UserFactory to initialize these Users
3. Implement a Transaction
4. Create a worklet definition file
5. Configure Chauffeur to run the worklet

Example

The `IsWord` transaction determines whether a word is included in a dictionary. The input of the transaction is a String: the word to test. The output is a Boolean: true if the input string is in the dictionary, or false otherwise.

Step 0: Create the dictionary

```
1  public class Dictionary {
2    private final String[] words;
3
4    public static Dictionary readDictionary(BufferedReader in)
5    throws IOException {
6      List<String> wordList = new ArrayList<String>();
7      String line;
8      while ((line = in.readLine()) != null) {
9        wordList.add(line);
10     }
11     return new Dictionary(wordList.toArray(new String[]));
12   }
13
14   public Dictionary(String[] words) {
15     this.words = words;
16   }
```

```
17    public boolean isWord(String word) {
18      // Please don't do this in a real application.
19      for (String dictWord: words) {
20        if (word.equals(dictWord)) {
21          return true;
22        }
23      }
24      return false;
25    }
26  }
```

Step 1: Create the User

```
1  public class WordsUser extends AbstractUser {
2    private final Dictionary dict;
3
4    public WordsUser(Worklet<?> worklet, int userId, Dictionary dict)
5    throws InvalidConfigurationException {
6      super(worklet, userId);
7      this.dict = dict;
8    }
9
10   public Dictionary getDictionary() {
11     return dict;
12   }
13 }
```

Step 2: Create the UserFactory

```
1  public class WordsUserFactory extends AbstractUserFactory<WordsUser> {
2    private static final String DICTIONARY_RESOURCE =
3            "org/spec/chauffeur/tutorial/words.dict";
4    private Dictionary dict;
5
6    public WordsUserFactory() throws IOException {
7      try (
8        InputStream in = this.getClass().getClassLoader().
9                getResourceAsStream(DICTIONARY_RESOURCE);
10       Reader inReader = new InputStreamReader(in, "UTF-8");
11       BufferedReader bufIn = new BufferedReader(inReader)
12     ) {
13      this.dict = Dictionary.readDictionary(bufIn);
14     }
15   }
16
17   @Override
18   public synchronized WordsUser createUser(
19           WorkletContext<WordsUser> workletContext, int userId)
20   throws InvalidConfigurationException, FatalWorkletException {
21     return new WordsUser(workletContext.getWorklet(), userId, dict);
22   }
23 }
```

Step 3: Implement the IsWord transaction

```
1  public class IsWord extends
2          Transaction<WordsUser, String, Boolean> {
3    private static final int MIN_LENGTH = 4;
4    private static final int MAX_LENGTH = 10;
5
6    @Override
7    public String generateInput(WordsUser user)
8    throws TransactionFailedException, FatalWorkletException {
9      Random rand = user.getRandomGenerator();
10     int range = MAX_LENGTH - MIN_LENGTH + 1;
11     int len = rand.nextInt(range) + MIN_LENGTH;
12
13     StringBuilder buf = new StringBuilder(len);
14     for (int i = 0; i < len; i++) {
15       buf.append((char)('a' + rand.nextInt(26)));
16     }
17     return buf.toString();
18   }
```

```
19    @Override
20    public Boolean process(WordsUser user, String word)
21    throws TransactionFailedException, FatalWorkletException {
22      return (user.getDictionary().isWord(word));
23    }
24
25    @Override
26    public boolean verifyResults(WordsUser user, String input,
27            Boolean results) {
28      return true;
29    }
30
31    @Override
32    public String getTransactionName() {
33      return "IsWord";
34    }
35  }
```

Step 4: Create the Worklet Definition

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <workletDefinition xmlns="http://spec.org/power_chauffeur">
3   <name>Tutorial Worklet</name>
4   <user-factory
5           className="org.spec.chauffeur.tutorial.WordsUserFactory"/>
6   <launch-customizer className="CpuLaunchCustomizer"/>
7
8   <default-scenario-mix-factory>wordsScenarioMix
9           </default-scenario-mix-factory>
```

```
10    <scenario-mix-factories>
11      <scenario-mix-factory name="wordsScenarioMix">
12        <scenarios>
13          <weighted-scenario>
14            <weight>1</weight>
15            <scenario-factory
16                    implClass="org.spec.chauffeur.RandomBatchScenario">
17              <batch-size>1</batch-size>
18              <macro-transaction-size>1</macro-transaction-size>
19              <transaction-mix-factory>
20                <weighted-transaction>
21                  <transaction-factory implClass="org.spec.
22                          chauffeur.tutorial.IsWord"/>
23                  <weight>1</weight>
24                </weighted-transaction>
25              </transaction-mix-factory>
26            </scenario-factory>
27          </weighted-scenario>
28        </scenarios>
29      </scenario-mix-factory>
30    </scenario-mix-factories>
31  </workletDefinition>
```

Some Definitions

Scenario A sequence of transactions. Chauffeur includes RandomBatchScenario and SequentialBatchScenario.

Weight The relative weight of a scenario or transaction. Higher numbers will cause this scenario/transaction to be executed more frequently.

Batch Size The number of transactions executed during each scenario.

Macro Transaction Size The number of transactions scheduled in one group (with no delay in between). Usually equal to batch size.

Step 5: Configure Chauffeur to run the worklet

Create a copy of an existing `config.xml` file. Update the workload and worklet name. Update the `<workletDefinition>` to reference your new worklet definition file, and add the code to the classpath.

```
1 <workletDefinition>
2   <location>org/spec/chauffeur/tutorial/tutorialWorklet.xml</location>
3   <classpath>
4     <entry>lib/tutorial.jar</entry>
5     <entry>../../Chauffeur/ChauffeurTutorial/src</entry>
6     <entry>../../Chauffeur/ChauffeurTutorial/build/classes</entry>
7   </classpath>
8 </workletDefinition>
```

Agenda

1. Creating a New Worklet
   - Defining a Basic Worklet
   - Enhancing the Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Enhancing the Worklet

At this point the new Worklet is runnable. But we can make the worklet more useful with some additional enhancements.

- Create a Worklet Version class
- Parameterize the worklet
- Add a second Transaction

Create a worklet Version class

The worklet version is printed at runtime and recorded in the results file. We typically update the version information from our Ant build scripts (see the ChauffeurTest worklet for an example).

Extend the `AbstractVersion` class (see next page). Then reference it from the worklet definition file:

```
<version className="org.spec.chauffeur.tutorial.Version"/>
```

```
1  public class Version extends AbstractVersion {
2    private static final String NAME = "@NAME@";
3    private static final String VERSION = "@VERSION@";
4    private static final String DATE = "@DATE@";
5    private static final String TIME = "@TIME@";
6    private static final String NUMBER = "@NUMBER@";
7
8    private static Version instance = new Version();
9
10   public Version() {
11     super("Tutorial", NAME, VERSION, DATE, TIME, NUMBER);
12   }
13
14   public Version(CommunicatorIn in, ClassLoader loader)
15   throws ChauffeurIOException {
16     super(in, loader);
17   }
18
19   public static Version getInstance() {
20     return instance;
21   }
22 }
```

Parameterize the worklet

Parameters can be passed to the worklet definition. Within the worklet definition, these parameters can be used to customize the user factory, transaction factory, or other objects.

When an element is added to the worklet definition XML file, Chauffeur translates the element name to a `set` method in that object. The value is converted to the appropriate type. For example, the XML fragment:

```
<user-factory
        className="org.spec.chauffeur.tutorial.WordsUserFactory">
  <dictionary>org/spec/chauffeur/tutorial/words.dict</dictionary>
</user-factory>
```

would result in a call to:

```
WordsUserFactory.setDictionary(String dict)
```

Add parameters to WordsUserFactory

```
1  public class WordsUserFactory extends AbstractUserFactory<WordsUser> {
2    // ...
3
4    public synchronized void setDictionary(String dictResource)
5    throws IOException {
6      try (
7        InputStream in = this.getClass().getClassLoader().
8                getResourceAsStream(dictResource);
9        Reader inReader = new InputStreamReader(in, "UTF-8");
10       BufferedReader bufIn = new BufferedReader(inReader)
11     ) {
12      this.dict = Dictionary.readDictionary(bufIn);
13     }
14   }
15 }
```

Add parameters to IsWord

IsWord.java:

```
1  private int minLength = 4;
2  private int maxLength = 10;
3
4  public void setMinLength(int minLength) {
5    this.minLength = minLength;
6  }
7
8  public void setMaxLength(int maxLength) {
9    this.maxLength = maxLength;
10 }
11
12 // Replace MIN_LENGTH with minLength and MAX_LENGTH with maxLength
```

tutorialWorklet.xml:

```
1 <transaction-factory implClass="org.spec.chauffeur.tutorial.IsWord">
2   <min-length>5</min-length>
3   <max-length>8</max-length>
4 </transaction-factory>
```

Adding parameters to the worklet definition

The previous charts made the parameters configurable from the worklet definition file. That is convenient for developers, but still doesn't make it easy for users to change the values. To make the parameters configurable from `config.xml`, first define them (with default values) in the worklet definition file:

```
1 <parameterDefs>
2   <minimum-word-length>4</minimum-word-length>
3   <maximum-word-length>10</maximum-word-length>
4 </parameterDefs>
```

To refer to the parameter, use the `param` attribute:

```
1 <transaction-factory implClass="org.spec.chauffeur.tutorial.IsWord">
2   <min-length param="minimum-word-length"/>
3   <max-length param="maximum-word-length"/>
4 </transaction-factory>
```

Now the parameters can be set from `config.xml`:

```
<workletDefinition>
  <location>org/spec/chauffeur/tutorial/tutorialWorklet.xml</location>
  <classpath>
    <entry>lib/tutorial.jar</entry>
  </classpath>
  <parameters>
    <parameter name="minimum-word-length">5</parameter>
    <parameter name="maximum-word-length">8</parameter>
  </parameters>
</workletDefinition>
```

Create a Second Transaction

Adding more transactions to an existing worklet is straightforward. Just create a new `Transaction` class and add the transaction information to the worklet definition file.
Let's add a second transaction that counts the number of words in the dictionary that contain a particular double letter (e.g. 'aa' or 'rr').

Double Letter: Extend the Dictionary

```
1  public class Dictionary {
2    ...
3    public int countSequence(CharSequence seq) {
4      int count = 0;
5      for (String dictWord: words) {
6        if (dictWord.contains(seq)) {
7          count++;
8        }
9      }
10     return count;
11   }
12 }
```

Double Letter: Create the Transaction

```
1  public class DoubleLetter extends
2          Transaction<WordsUser, Character, Integer> {
3    @Override
4    public Character generateInput(WordsUser user)
5    throws TransactionFailedException, FatalWorkletException {
6      Random rand = user.getRandomGenerator();
7      return Character.valueOf((char)('a' + rand.nextInt(26)));
8    }
9
10   @Override
11   public Boolean process(WordsUser user, Character letter)
12   throws TransactionFailedException, FatalWorkletException {
13     StringBuilder buf = new StringBuilder(2);
14     buf.append(letter).append(letter);
15     return user.getDictionary().countSequence(buf);
16   }
```

```
17    @Override
18    public boolean verifyResults(WordsUser user, Character input,
19            Integer results) {
20      // For a particular dictionary, we could precompute the
21      // correct answers for each letter, and verify them here.
22      return true;
23    }
24
25    @Override
26    public String getTransactionName() {
27      return "DoubleLetter";
28    }
29  }
```

Double Letter: Edit the worklet description

```
1   <weighted-scenario>
2     <weight>1</weight>
3     <scenario-factory
4             implClass="org.spec.chuaffeur.RandomBatchScenario">
5       <batch-size>1</batch-size>
6       <macro-transaction-size>1</macro-transaction-size>
7       <transaction-mix-factory>
8         <weighted-transaction>
9           <transaction-factory implClass="org.spec.
10                  chauffeur.tutorial.IsWord"/>
11          <!-- 10 IsWord for every DoubleLetter -->
12          <weight>10</weight>
13        </weighted-transaction>
14        <weighted-transaction>
15          <transaction-factory implClass="org.spec.
16                  chauffeur.tutorial.DoubleLetter"/>
17          <weight>1</weight>
18        </weighted-transaction>
19      </transaction-mix-factory>
20    </scenario-factory>
21  </weighted-scenario>
```

Agenda

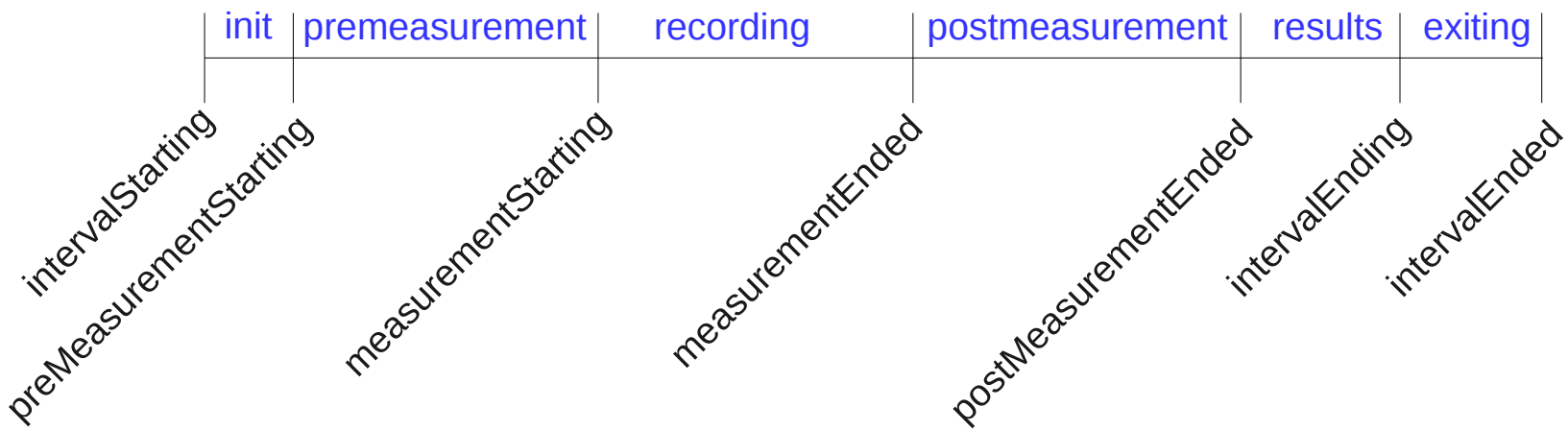
1. Creating a New Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Collecting Additional Data with Listeners

`RunListeners` can be registered with Chauffeur to receive notifications at various points in the run.

There are 4 separate notifications for most aspects of the run: starting, started, ending, and ended. These notifications are issued for each suite, workload, worklet, phase, and sequence.

There are a total of 7 notifications during each interval:



Included Chauffeur Listeners

The Chauffeur WDK includes serveral listeners:

- **PowerAnalyzerListener**: Collects power data from PTDaemon
- **TemperatureSensorListener**: Collects temperature data from PTDaemon
- **RawFileWriter**: Writes the `results.xml` file
- **TimingListener**: Used internally to record timestamps for various events
- **WriterRunListener**: Writes run status information to some location
- **ConsoleRunListener**: A `WriterRunListener` subclass that writes status information to the console
- **GarbageCollectorListener**: Captures GC metrics during each interval; can force GC cycles at various points
- **HostConfigurationListener**: Changes certain OS-specific parameters on the SUT that are necessary for some worklets

Metrics

Listeners can store data in `Metrics`. These metrics are stored in `results.xml`, and can be accessed by the reporter.

Each RunListener defines a `MetricsProvider` and creates a hierarchy of `MetricGroup` and `Metric`. These define the data types and structure of how the results will be stored. At runtime, the RunListener methods can access a `Metrics` object from the context, and use the `MetricData` to store the actual values.

Metrics should normally be stored in one of the "Ending" events, rather than "Ended". That way the values will be set before the metrics are recorded by the RawFileWriter.

Sample RunListener

As an example, we will write a listener for recording CPU utilization metrics on Linux. We could use this data to understand how CPU utilization is affected by power management as the throughput changes.

Note

CPU utilization is influenced by many factors, and is generally not comparable across hardware architectures or operating systems. Chauffeur load levels do NOT target a particular CPU utilization – they target some percentage of the calibrated throughput.

Creating the LinuxCpuUtilizationListener

```
1  public class LinuxCpuUtilizationListener extends RunListenerAdapter {
2    private static final MetricGroup GROUP_ROOT =
3            MetricGroup.createRootGroup("cpu");
4    private static final MetricGroup GROUP_MEASUREMENT =
5            GROUP_ROOT.createSubgroup("measurement");
6    private static final Metric<Double> METRIC_MEASUREMENT_CPU =
7            GROUP_MEASUREMENT.createMetric("utilization");
8
9    @SuppressWarnings("unchecked")
10   private static final Metric<Double>[] COMPONENT_UTILIZATION_METRICS
11           = (Metric<Double>[])new Metric<?>[] {
12     GROUP_MEASUREMENT.createMetric("user"),
13     GROUP_MEASUREMENT.createMetric("nice"),
14     GROUP_MEASUREMENT.createMetric("sys"),
15     GROUP_MEASUREMENT.createMetric("idle"),
16     GROUP_MEASUREMENT.createMetric("iowait"),
17     GROUP_MEASUREMENT.createMetric("irq"),
18     GROUP_MEASUREMENT.createMetric("softirq"),
19     GROUP_MEASUREMENT.createMetric("steal"),
20     GROUP_MEASUREMENT.createMetric("guest"),
21   };
```

```
22  private MetricsProvider provider;
23
24  private long startValues[];
25  private long endValues[];
26
27  @Override
28  public void suiteStarted(SuiteEvent evt) {
29    provider = new MetricsProvider("cpu");
30  }
31
32  @Override
33  public void measurementStarting(IntervalEvent<?> evt)
34        throws ListenerException {
35    try {
36      startValues = readCpuStats();
37    } catch (IOException x) {
38      throw new ListenerException("Exception reading CPU stats", x);
39    }
40  }
```

```
41    @Override
42    public void measurementEnded(IntervalEvent<?> evt)
43            throws ListenerException {
44      try {
45        endValues = readCpuStats();
46      } catch (IOException x) {
47        throw new ListenerException("Exception reading CPU stats", x);
48      }
49
50      if (startValues == null && endValues == null) {
51        return;
52      }
53
54      long startTotal = 0;
55      for (int i = 0; i < startValues.length; i++) {
56        startTotal += startValues[i];
57      }
58
59      long endTotal = 0;
60      for (int i = 0; i < endValues.length; i++) {
61        endTotal += endValues[i];
62      }
```

```
63      // startValues[3] is the idle CPU
64      long startUsed = startTotal - startValues[3];
65      long endUsed = endTotal - endValues[3];
66      double util = (endUsed - startUsed) * 1.0 /
67              (endTotal - startTotal);
68
69      Metrics metrics = evt.getContext().getMetrics();
70      MetricData metricData = metrics.getMetricData(provider);
71      metricData.set(METRIC_MEASUREMENT_CPU, util);
72
73      for (int i = 0; i < startValues.length; i++) {
74        metricData.set(
75                (Metric<Double>)COMPONENT_UTILIZATION_METRICS[i],
76                (endValues[i] - startValues[i]) * 1.0 /
77                    (endTotal - startTotal));
78      }
79    }
```

```
80   private long[] readCpuStats() throws IOException {
81     String line;
82     try (FileReader fin = new FileReader("/proc/stat");
83          BufferedReader in = new BufferedReader(fin)) {
84       while ((line = in.readLine()) != null) {
85         String values[] = line.split(" +");
86         if (values.length < 2) {
87           // Ignore
88           continue;
89         }
90
91         String name = values[0];
92         if (name.equals("cpu")) {
93           long results[] = new long[values.length - 1];
94           for (int i = 1; i < values.length; i++) {
95             results[i-1] = Long.parseLong(values[i]);
96           }
97           return results;
98         } // else ignore
99       }
100      return null;
101    }
102  }
```

Configuring the New Listener

Compile the code and store it in a jar file. Copy the jar to the Chauffeur WDK `lib` directory on the controller and host(s). To use the code, add a new entry to `listeners.xml` on the controller system:

```
1 <listener>
2   <clients>*:[host]</clients>
3   <type>com.ibm.chauffeur.listeners.LinuxCpuUtilizationListener</type>
4   <classpath>
5     <path>lib/cpuUtil.jar</path>
6   </classpath>
7 </listener>
```

Listener Clients Definitions

The `<clients>` element specifies where the listener will run.

`<clients>`	JVM(s)
default	Director
`*:[director]`	Director
`*:[host]`	Every Host
`myHost1:[host]`	Only the host with id *myHost1*
`*:*`	Every Client
`*:0`	Client id 0 on every host
`myHost1`	Every client on host `myHost1`
`myHost1:*`	Every client on host `myHost1`
`myHost1:1`	Client id 1 on host *myHost1*
`myHost1:1,myHost2:3`	Client id 1 on host `myHost1` and client id 3 on host `myHost2`

Results from the Listener

With this listener in place, the `results.xml` contains the following section, alongside the other metrics for each interval:

```
<metrics>
  <subset id="arnoldje-w530">
    <provider id="cpu">
      <cpu>
        <measurement>
          <irq>0.000248</irq>
          <softirq>0.000497</softirq>
          <guest>0</guest>
          <iowait>0</iowait>
          <sys>0.024596</sys>
          <idle>0.256149</idle>
          <steal>0</steal>
          <user>0.718509</user>
          <utilization>0.743851</utilization>
          <nice>0</nice>
        </measurement>
      </cpu>
    </provider>
  </subset>
</metrics>
```

Agenda

1. Creating a New Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Running the Reporter

Chauffeur will automatically generate reports at the end of each run. You can also generate reports manually. This is useful for:

- Re-creating the report after correcting test environment data
- Generating non-standard reports
- Ensuring that results have not been modified

To run the reporter, use the `reporter.bat` or `reporter.sh` script in the Chauffeur WDK installation.

Sample usage:

```
./reporter.sh -r results/chauffeur-0000/results.xml
```

Creating CSV Output

The HTML reports are designed to be human readable, but they aren't very useful for importing data into other tools. The reporter can also generate reports in CSV format, which can easily be imported into spreadsheets, R, or other tools for further analysis.

CSV Report:

```
./reporter.sh -r results/chauffeur-0000/results.xml
-c
```

```
Workload,Worklet,Phase,Load Level,Performance,Transactions/sec,Input Generation Time (s),Successful Time
Chauffeur Test,Chauffeur Test,warmup,max,25914.868093,25912.270737,0.046,39.163,0,
Chauffeur Test,Chauffeur Test,calibration,max,33589.650046,33589.449882,0.243,471.311,0,
Chauffeur Test,Chauffeur Test,measurement,100%,33585.051324,33584.988952,0.228,463.142,0,
Chauffeur Test,Chauffeur Test,measurement,50%,16798.435943,16798.384113,0.198,270.662,0,
```

Report Stylesheets

The Reporter generates reports based on XSLT stylesheets. CSV and unformatted text reports use the XSLT output directly. HTML and formatted text reports are first converted to an XML representation of the report, then the XML is converted to HTML or plain text.

The stylesheets for the standard reports are included in the Chauffeur WDK in `src/Chauffeur/Reporter/src/java/org/spec/chauffeur/reporter/resources`. The main HTML reports are defined in `summary.xsl` and `details.xsl`, but these also share common code in `common.xsl`, `environment.xsl` and other included files. The standard CSV reports are defined in `csv-summary.xsl` and `csv-detail.xsl`.

csv-detail.xsl

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0"
3          xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4   <xsl:output method="text"/>
5   <xsl:template match="/">
6    <xsl:text>Workload,Worklet,Phase,Load Level,Performance,
7  Transactions/sec,Input Generation Time (s),Successful Time (s),
8  Sum Average Watts,Watts_1,Volts_1,Amps_1,Uncert_1,PF_1
9  </xsl:text>
10
11   <xsl:for-each select="suite/workload">
12    <xsl:variable name="workloadName" select="@name"/>
13    <xsl:for-each select="worklet">
14     <xsl:variable name="workletName" select="@name"/>
15     <xsl:for-each select="phase">
16      <xsl:variable name="phaseName" select="@name"/>
17      <xsl:for-each select="sequence/interval">
18       <xsl:value-of select="$workloadName"/><xsl:text>,</xsl:text>
19       <xsl:value-of select="$workletName"/><xsl:text>,</xsl:text>
20       <xsl:value-of select="$phaseName"/><xsl:text>,</xsl:text>
21       <xsl:value-of select="@name"/><xsl:text>,</xsl:text>
```

```
22        <xsl:value-of select="result/score"/><xsl:text>,</xsl:text>
23        <xsl:value-of select="result/transactionsPerSecond"/>
24        <xsl:text>,</xsl:text>
25        <xsl:value-of select="format-number(sum(result/transactions/transaction/
26 inputGenerateElapsedNs) div 1000000000, '###0.000')"/>
27        <xsl:text>,</xsl:text>
28        <xsl:value-of select="format-number(sum(result/transactions/transaction/
29 successElapsedNs) div 1000000000, '###0.000')"/>
30        <xsl:text>,</xsl:text>
31        <xsl:value-of select="sum(result/metrics/provider/power-analyzer/measurement/
32 watts/average)"/>
33        <xsl:text>,</xsl:text>
34        <xsl:for-each select="result/metrics/provider/power-analyzer">
35         <xsl:sort select="../@id"/>
36         <xsl:value-of select="measurement/watts/average"/><xsl:text>,</xsl:text>
37         <xsl:value-of select="measurement/volts/average"/><xsl:text>,</xsl:text>
38         <xsl:value-of select="measurement/amps/average"/><xsl:text>,</xsl:text>
39         <xsl:value-of select="measurement/uncertainty/average"/><xsl:text>,</xsl:text>
40         <xsl:value-of select="measurement/power-factor/average"/><xsl:text>,</xsl:text>
41        </xsl:for-each>
42        <xsl:text>
43 </xsl:text>
44      </xsl:for-each>
45     </xsl:for-each>
46    </xsl:for-each>
47   </xsl:for-each>
48  </xsl:template>
49 </xsl:stylesheet>
```

Modifying a Report

Generating a new report from scratch is somewhat complicated. Creating a modified copy of an existing report is simpler. Start by making a copy of one of the standard reports, then edit it to fit your needs. Look at one of your `results.xml` files to see the structure of the XML and find the fields you are looking for.

As an example, let's update the CSV report to include the overall CPU utilization for each interval. Start by making a copy of `csv-detail.xsl` called `csv-cpu.xsl`.

Add the new column to the row of headings:

```
<xsl:text>Workload,Worklet,Phase,Load Level,Performance,
Transactions/sec,Input Generation Time (s),Successful Time (s),
Sum Average Watts,Watts_1,Volts_1,Amps_1,Uncert_1,PF_1,CPU Util
</xsl:text>
```

Now at the end of the `<xsl:for-each select="sequence/interval">` block, add the new value:

```
<xsl:value-of select="100.0 *
  sum(result/metrics/subset/provider/cpu/
      measurement/utilization) div
  count(result/metrics/subset/provider/cpu/
        measurement/utilization)"/>
```

Note that we take the average, since in a multi-host run the CPU utilization would be reported for each host. We could have also used a `<xsl:for-each>` to loop through each host and put each utilization in its own column.

Using Your New Report Stylesheet

To generate a report using your new stylesheet, use the `-x` parameter to specify the stylesheet.

CSV Report with custom stylesheet:

```
./reporter.sh -r results/chauffeur-0000/results.xml
-c -x path/to/csv-cpu.xsl -o myReport.csv
```

The process of modifying an HTML-formatted report is the same; only the XSLT is different.

Agenda

1. Creating a New Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Create a Suite Description

A `SuiteDescription` helps put "spit and polish" on a Suite of Chauffeur worklets. Simple worklets can just use the default `BasicSuiteDescription`. Creating your own `SuiteDescription` gives you the ability to:

- Specify a `Version` for the entire suite
- Indicate the results directory prefix
- Create a list of one or more reports that will be generated at the end of the run
- Make reports link to a particular glossary URL
- Define a set of `Validators` that are used to determine whether or not a result is valid

A simple SuiteDescription

```
1  public class Tutorial extends BasicSuiteDescription {
2    public Tutorial() { }
3
4    public Tutorial(CommunicatorIn in, ClassLoader loader)
5    throws ChauffeurIOException {
6      super(in, loader);
7    }
8
9    public void writeObject(CommunicatorOut out)
10   throws ChauffeurIOException {
11     super.writeObject(out);
12   }
13
14   @Override
15   public String getResultsDirectoryPrefix() {
16     return "tutorial-";
17   }
18
19   @Override
20   public IVersion getVersion() {
21     return Version.getInstance();
22   }
23 }
```

Changing Default Reports

The `SuiteDescription` includes a list of the default reports generated at the end of each run. For each report, you specify the type (TEXT, HTML, or PLAIN), the output file, the XSLT stylesheet, and the glossary URL. The custom reporter stylesheets are normally packaged in the same jar file as the `SuiteDescription` and loaded via the ClassLoader.

The glossary URL is a hyperlink to a document that describes each field in the report. The HTML reports allow the user to click on the field names to go directly to the corresponding documentation. The default URL includes the field descriptions for the standard Chauffeur WDK reports.

```
1  private static final String XSL_SUMMARY_RESOURCE =
2          "org/spec/chauffeur/tutorial/resources/summary.xsl";
3  private static final String CSV_XSL_DETAIL_RESOURCE =
4          "org/spec/chauffeur/tutorial/resources/csv-details.xsl";
5
6  @Override
7  public List<ReportSpecification> getReportSpecifications(
8          File outputDir, String outputFilenamePrefix) {
9      List<ReportSpecification> reportSpecs =
10             new ArrayList<ReportSpecification>();
11     try {
12         reportSpecs.add(new ReportSpecification(ReportFormat.HTML,
13                 outputDir, outputFilenamePrefix, null,
14                 XSL_SUMMARY_RESOURCE, getGlossaryUrl()));
15         reportSpecs.add(new ReportSpecification(ReportFormat.PLAIN,
16                 outputDir, outputFilenamePrefix + "-details", "csv",
17                 CSV_XSL_DETAIL_RESOURCE, getGlossaryUrl()));
18     } catch (FileNotFoundException x) {
19         // Error handling omitted
20     }
21     return reportSpecs;
22 }
23
24 @Override
25 public String getGlossaryUrl() {
26     return "http://example.com/wdk/my_glossary.html";
27 }
```

Defining Validators

The Chauffeur WDK includes support for automated validation of results. The WDK includes several default validators, but new ones can be added and existing ones removed.

Each `Validator` can process the `results.xml` and return a list of messages to indicate any issues. These messages can be at `INVALID`, `WARNING`, or `INFORMATION` levels. Validators can also perform checks at runtime to detect issues early in the run and allow the user to correct them.

For sample `Validator` implementations, see the default validators in `src/Chauffeur/ChauffeurCommon/src/java/org/spec/chauffeur/common/validators`.

Defining the List of Validators

The `SuiteDescription` provides a list of `Validators` to be used for this suite of worklets. If your `SuiteDescription` doesn't override this method, the default set of validators will be used:

```
1 @Override
2 public Validators getValidators() {
3     return new MyValidators();
4 }
```

Table: Default Validators

TransactionFailuresValidator	TargetThroughputValidator
ThroughputVariationValidator	ResultModificationValidator
PowerAnalyzerDeviceValidator	PowerErrorReadingsValidator
PowerUncertaintyReadingsValidator	VoltageValidator
TemperatureSensorDeviceValidator	PtdVersionValidator
TemperatureErrorReadingsValidator	CompleteRunValidator
MinimumTemperatureValidator	

```java
public class MyValidators extends Validators {
    @Override
    protected List<Validator> getResultsValidators() {
        List<Validator> validators = new ArrayList<Validator>(
                super.getResultsValidators());
        validators.add(new MyNewValidator());
        return validators;
    }

    @Override
    protected Collection<Validator> getDirectorValidators() {
        return super.getDirectorValidators();
    }

    @Override
    protected Collection<Validator> getHostValidators() {
        return super.getHostValidators();
    }

    @Override
    protected Collection<Validator> getClientValidators() {
        return super.getClientValidators();
    }
}
```

Plugging in New Behavior

Most aspects of the Chauffeur runtime behavior can be changed by plugging in new implementations. We've already seen this when we replaced the `AverageThroughputCalibrator` with a `ConstantValueCalibrator`, and when we used a `ThroughputPercentageSeries` instead of `GraduatedMeasurementSeries`.

You can also create your own implementation of Chauffeur interfaces to plug in your own behavior. This requires a good understanding of the Chauffeur code, but we'll try a simple example.

We'll create a new IntervalSeries which uses random throughput percentages during each interval. This is somewhat similar to the `ThroughputPercentageSeries`, so we can use that as a base.

Defining The RandomPercentageSeries

```
1  public class RandomPercentageSeries<U extends User>
2          extends AbstractIntervalSeries<U> {
3      private long seed = System.nanoTime();
4
5      public RandomPercentageSeries() {
6      }
7
8      public RandomPercentageSeries(CommunicatorIn in,
9              ClassLoader loader) throws ChauffeurIOException {
10         super(in, loader);
11         this.seed = in.readLong();
12     }
13
14     @Override
15     public void writeObject(CommunicatorOut out)
16             throws ChauffeurIOException {
17         super.writeObject(out);
18         out.writeLong(seed);
19     }
20
21     public void setSeed(long seed) {
22         this.seed = seed;
23     }
```

```
24      @Override
25      public Iterable<IntervalDef> getIntervals(
26              WorkletContext<U> context, CalibrationResult calibrated)
27              throws InvalidConfigurationException {
28          Random random = context.getWorklet().createRandomGenerator();
29          random.setSeed(seed);
30          int numIntervals = getIntervalCount();
31          NumberFormat percentFormat =
32                  NumberFormat.getPercentInstance();
33          percentFormat.setMaximumFractionDigits(0);
34          List<IntervalDef> intervals =
35                  new ArrayList<IntervalDef>(numIntervals);
36          for (int i = 0; i < numIntervals; i++) {
37              double percentage = random.nextInt(101) / 100.0;
38              DelayDistributionGenerator delayDist =
39                      ExponentialDelayDistribution.
40                      createDistributionGenerator(percentage);
41              intervals.add(new IntervalDef(
42                      percentFormat.format(percentage),
43                      getScenarioMixFactoryId(context),
44                      delayDist,
45                      calibrated,
46                      getLaunchDefinition()));
47          }
48          return intervals;
49      }
50  }
```

Configuring the RandomPercentageSeries

Update the measurement phase configuration in `config.xml`. Note that we have to specify the full classname since this class isn't in the standard Chauffeur package:

```
1  <measurement-phase>
2      <sequence>
3          <interval-series className="org.spec.chauffeur.
4                      tutorial.RandomPercentageSeries">
5              <interval-count>4</interval-count>
6              <seed>123456</seed>
7          </interval-series>
8
9          <interval-length ref="testIntervalLength"/>
10     </sequence>
11 </measurement-phase>
```

Agenda

1. Creating a New Worklet
2. Creating New Listeners
3. Generating Custom Reports
4. Plugging In New Behavior
5. Conclusion

Conclusion

In this session we described how to create a new worklet, listeners, and reports. We also discussed how to customize the behavior of Chauffeur.

Chauffeur is a flexible tool for implementing performance and energy-efficiency workloads. I've demonstrated several ways that Chauffeur can be customized, but I'm most excited to find out how people use it in ways that I didn't expect. I look forward to seeing your research using Chauffeur at next year's ICPE.

Acknowledgements

Thanks to the current and former members of the SPEC Power committee who assisted with the development of Chauffeur and SERT:

- John Beckett
- Hansfried Block
- Greg Darnell
- Ashok Emani
- Karl Huppler
- Klaus-Dieter Lange
- Shreeharsha G. Neelakantachar
- Sanjay Sharma
- Van Smith
- Cloyce Spradling
- Nathan Totura
- Mike Tricker
- Jeff Underhill
- Karin Wulf

Questions