Table of Contents

JSONCollector (DEPRECATED) **1**
- Description 1
- File categories 1
- Get the code 1
- Useful Links 1
- Binary 1
- API: writing CSV (fast) and JSON (slow) files 2

JSONCollector (DEPRECATED)

Description

The JSONCollector is a tool that enables summing FFF monitoring files. The two main uses of the tool are:

1. Binary: loading json files and generating a single output file, representing the summed output
2. API: used by CMSSW processes to output monitoring files respecting a pre-defined structure of the monitorable fields

In order to improve efficiency, the json files are split into **legend** and **data** files. Legend files describe the structure of a data file, and are split into a few categories (e.g. monitoring files generated by the rubuilder, monitoring files generated by the filter processes, monitoring files describing data files output by the filtering processes, etc.). Data files are instances of the different monitored fields, respecting a file type described by the legend.

The JSONCollector implementation uses the *jsoncpp* library: http://jsoncpp.sourceforge.net/

File categories

There are 3 main categories of files the collector works with:

1. JSON data files: monitoring files in the *json* format that respect a legend (also referred to as *definition*).
2. CSV data files: monitoring files in the *csv* format that respect a legend. These files are used for fast monitoring in the FFF (monitoring information output and collected every second).
3. Legend files: describe the JSON and CSV data files, providing the operation to be performed on each field of the data files

For usage examples and monitoring file types in the FFF, please see: [link]

Get the code

Code is available on the cms-hlt git repo: **https://git.cern.ch/web/?p=cms-hlt.git**

Useful Links

* FFF Metafile Formats

Binary

The command line tool loads json or csv data files from a directory (optionally using a regex for the file name) and outputs the aggregated result according to the legend. Input files must have the same legend and be consistent.

Usage: **$jsoncollector [-json/-csv] [-d] [-r regex] [-o outfile] [-i indir1 indir2 ...infileN]**

where:

- **[-json/-csv]**: the type of input files

- **[-d]** (optional): output-for-display flag. By setting this flag, the collector outputs a json file that combines the summed data together with the legend information for this field. This is useful for display purposes, providing a self-describing json file. **This file cannot be aggregated again since it no longer respects the format for data files, but should be used only when a human-readable result is required.**
- **[-r regex]** (optional): regular expression for the names of files in the input space. The regular expression applies to the file name **stub** (which is /path/to/**myMonitoringFile**.jsn)
- **[-o outfile]**: full path of the output file
- **[-i indir1 indir2 ...infileN]**: list of input files and directories to get the files from

API: writing CSV (fast) and JSON (slow) files

```
#include "JSONCollector/interface/JsonMonitorable.h"
#include "JSONCollector/interface/FastMonitor.h"
#include "JSONCollector/interface/JSONSerializer.h"

#include <vector>

using namespace std;
using namespace jsoncollector;

class Monitored {

public:
        IntJ processedEvents;
        IntJ acceptedEvents;
        IntJ microstate;
        IntJ ministate;
        IntJ macrostate;
};

int main() {

        Monitored mon;

        // set names of the variables to be matched with JSON Definition
        mon.processedEvents.setName("Processed Events");
        mon.acceptedEvents.setName("Accepted Events");
        mon.microstate.setName("MicroState");
        mon.macrostate.setName("MiniState");
        mon.macrostate.setName("MacroState");

        // create a vector of all monitorable parameters to be passed to the monitor
        vector<JsonMonitorable*> monParams;
        monParams.push_back(&mon.processedEvents);
        monParams.push_back(&mon.acceptedEvents);
        monParams.push_back(&mon.macrostate);
        monParams.push_back(&mon.ministate);
        monParams.push_back(&mon.microstate);

        // create a FastMonitor using vector of monitorable parameters, a path to a JSON Definiti
        FastMonitor monitor (monParams,"/path/to/definition.jsd");

        // change the monitored parameters
        mon.processedEvents = 100;
        mon.acceptedEvents = 76;
        mon.microstate = 3;
        mon.macrostate = 1;

        // take a snapshot of the monitored values and output to a CSV file
        monitor.snap("/path/to/outputCSVFile.csv");

         // change the monitored parameters again
        mon.processedEvents = 200;
```

```
        mon.acceptedEvents = 150;
        mon.microstate = 9;
        mon.macrostate = 1;

        // same as before, updating the file
        monitor.snapshot("/path/to/outputCSVFile.csv");

        // do something else ...

        // output a slow monitoring file, summing all the information from the snapshots
        monitor.outputFullHistoDataPoint("/path/to/outputJSONFile.jsn");

return 0;
}
```

For the above code we need a definition file at the specified path. The output format will be given by the definition at */path/to/definition.jsd*.

```
{
   "legend" : [
      {
         "name" : "Processed Events",
         "operation" : "sum"
      },
      {
         "name" : "Accepted Events",
         "operation" : "sum"
      },
      {
         "name" : "Microstate",
         "operation" : "histo"
      },
      {
         "name" : "Ministate",
         "operation" : "histo"
      },
      {
         "name" : "Macrostate",
         "operation" : "histo"
      }
   ]
}
```

-- AndreiSpataru - 14-Aug-2013

This topic: Sandbox > JSONCollector
Topic revision: r19 - 2013-08-14 - AndreiSpataru