HPS MC Project Documentation

The hps-mc project provides a set of compiled tools and Python scripts for running MC jobs.

Installation

Prerequisites

The following prerequisites are required for running hps-mc:

- Maven (3.0 or greater)
- gcc (4.8 or greater)
- CMake (3.0 or greater)
- Python (2.7 or greater)
- SLIC
- ROOT (optional)

Bundled Dependencies

These dependencies are compiled and installed from source code within the project:

- egs5 event generator
- MadGraph 4 and MadGraph 5
- StdHep library and tools (based on version 5.6.1)

Installed Dependencies

The installation procedure will automatically download and install the following dependencies:

- LCIO (2.7.2)
- hps-java (HEAD of master)
- hps-fieldmaps

Building the Project

Start by checking out the project from github:

```
git clone https://github.com/jeffersonlab/hps-mc
```

Now, create a build dir and run CMake:

```
cd hps-mc; mkdir build; cd build
cmake -DCMAKE_INSTALL_PREFIX=../install ..
```

This will configure the project to be installed within a directory called *install.*

Next, run make to build the project:

```
make -j4 install
```

Job Environment

A number of environment variables are required for hps-mc to function properly.

These need to be setup by sourcing this script before attempting to run any jobs:

```
. hps-mc/install/bin/hps-mc-env.sh
```

The *HPSJAVA_JAR* variable will by default point to the copy built during the installation. You can override this by setting it to point to a copy of the jar you want to use instead e.g.

```
export HPSJAVA_JAR=~/.m2/repository/org/hps/hps-distribution/4.0/hps-distribution-4.0-bin.jar
```

The SLIC application binary needs to be present in the environment (the SLIC environment is not managed directly by hps-mc), and you can check for this using:

```
which slic
```

If this application is not found then run the *slic-env.sh* to set it up before executing any hps-mc jobs.

ROOT is only currently used by one job script (*tuple_job.py*). You need to set it up in your environment using the supplied *thisroot.sh* script for this job script to work.

Running Job Scripts

Running individual job scripts requires providing a JSON file with required parameters.

Running a job script looks like:

```
python job.py params.json
```

As an example, to generate WAB (Wide Angle Bremsstrahlung) events using MadGraph, these parameters could be used:

```
{
    "job_num": 1,
    "nevents": 10000,
    "seed": 1234,
    "output_dir": "output",
    "run_params": "1pt05",
    "output_files": {
        "wab_events.lhe.gz": "wab_events_1234.lhe.gz"
    }
}
```

The **job_num** specifies the job ID which is useful when running on a batch system.

The **nevents** parameter specifies how many events to generate.

The **seed** gives the random number seed for the job.

The **output_dir** is the destination directory for output files (it can be an absolute or relative path).

The **run_params** says what beam parameters to use.

The **output_files** section contains a list of source and destination files. The file on the left side is copied to a file name listed on the right side. (Here the file *wab_events.lhe.gz* will be copied to *wab_events_1234.lhe.gz* in the output directory.)

The names of the output file sources are dependent on the specific job script. The destination can be any valid file name.

If you saved these job parameters as the file *job.json* you can run the WAB job as follows:

```
python hps-mc/python/jobs/wab_job.py job.json
```

This should create the file *output/wab_events_1234.lhe.gz* in the current directory.

You will probably want to run jobs locally in a scratch directory, as they will tend to write out many files!

Input and Output Files

The job parameters may specify input files if the script uses them and optionally output file locations.

```
{
    "input_files": {
        "events1.stdhep": "/path/to/events1.stdhep",
        "events2.stdhep": "/path/to/events2.stdhep",
    }
    "output_files": {
        "events1.slcio": "my_events1.slcio",
        "events2.slcio": "my_events2.slcio"
    },
    "output_dir": "/path/to
/outdir"
}
```

In the above toy example, the files listed in *input_files* will be copied from the absolute path on the right to the file name on the left (format is *"destination": "source"*).

The output files will be copied from the left hand path in the local scratch dir to the file name on the right (format is *"source": "destination"*).

All output files will be copied to the directory listed under *"output_dir"*, which can be an absolute or relative path.

To keep the names of the output files created by the job, simply list the same file name for the output file entries.

```
{
    "output_files": {
        "events1.slcio": "events1.slcio",
        "events2.slcio": "events2.slcio"
    }
}
```

This will make the job copy the file *events1.slcio* to a file with the same name in the output directory.

Creating Job Workflows

In order to run jobs on a batch system such as LSF or Auger, the job parameters need to be expanded into a workflow, which is a JSON file containing parameters for all the individual jobs.

For instance, assume you have the following parameters in the file *job.json*:

```
{
    "nevents": 10000,
    "seed": 123456,
    "run_params": "1pt05",
    "detector": "HPS-EngRun2015-Nominal-v5-0-fieldmap",
    "run": 5772,
    "readout_steering": "/org/hps/steering/readout/EngineeringRun2015TrigPairs1_Pass2.lcsim",
    "recon_steering": "/org/hps/steering/recon/EngineeringRun2015FullReconMC.lcsim",
    "output_dir": "output",
    "output_files": {
        "tritrig.slcio": "tritrig_0001.slcio"
    }
}
```

This can be expanded into a workflow using the following command:

```
hps-mc-workflow -n 1000 -r 1234 -w tritrig hps-mc/python/jobs/tritrig_job.py job.json
```

Now you should see a local file called *tritrig.json* which contains information for running 1000 jobs of this type.

The input files for a workflow may be supplied in one of two ways.

A file glob will supply multiple input files to the workflow, one per job.

```
"input_files" : {
    "beam.stdhep": "/not/a/real/path/beam*.stdhep"
}
```

Mutiple files can be supplied based on the following syntax.

```
"input_files" : {
    "beam.stdhep": {
        "/not/a/real/path/beam*.stdhep": 10
    }
}
```

This will expand into JSON parameters that include 10 files per job in the workflow.

Running Jobs on the Batch System

Automatically submitting jobs to the batch system requires that you have created a workflow from your job parameters (covered in last section).

The following commands use the script *hps-mc-bsub* to submit jobs to LSF (e.g. SLAC environment).

The command *hps-mc-jsub* should be used instead when submitting at JLab.

To submit all jobs in a workflow, execute a command similar to the following:

```
 hps-mc-bsub -l $PWD/logs ./tritrig.json
```

You can also submit only certain job ids using a syntax like this to list specific job IDs:

```
hps-mc-bsub -l $PWD/logs ./tritrig.json 1000 2000 [etc.]
```

Finally, it is possible to submit a range of job IDs:

```
hps-mc-bsub -l $PWD/logs -r 0:99 ./tritrig.json
```

This will submit all the jobs IDs from 0 to 99 in the workflow.

Project Structure

The main project has the following directory structure:

Directory	**Contains**	**Notes**
data	data files	has *run_params.json* with beam parameters
generators	event generators	
generators/egs5	egs5 event generator	
generators/madgraph4	MadGraph4 generator	
generators/madgraph5	MadGraph5 generator	
python	Python scripts	
python/hpsmc	Python framework scripts	
python/jobs	Python job scripts	
python/test	Python test scripts	
scripts	scripts (bash, csh, XML, etc.)	Miscellaneous helper scripts and other scripts processed by CMake
scripts/mc_scripts	Auger based scripts	Backup of JLab Auger MC production scripts (not used by hps-mc)
scripts/run_params	scripts for printing run params	Backup of JLab scripts (not used by hps-mc)

scripts/MadGraph	scripts for printing information from LHE files	Backup of JLab scripts (not used by hps-mc)

Additionally, the following directory structure is installed to *CMAKE_INSTALL_PREFIX*.

Directory	Contains	Notes
bin	executables and scripts	
lib	program libraries	
lib/python	python framework and scripts	
share	project data	
share/detectors	detector description files (LCDD)	used when running SLIC
share/fieldmaps	full B-field maps	used when running SLIC

The *bin* dir contains a large number of scripts and binaries that are created during the build process.

File	Description	Notes
egs5_*	egs5 event generation executables	
stdhep_*	StdHep tools	
hps-mc-env.sh	Bash setup script	
hps-mc-env.csh	CSH setup script	
lcio_dumpevent	utility for dumping LCIO event data	
hps-mc-bsub	wrapper for submitting LSF jobs	
hps-mc-jsub	wrapper for submitting Auger jobs	
hps-mc-workflow	wrapper for creating job workflows	

Job Scripts

The project comes with a number of pre-written scripts in the *python/jobs* dir for running typical HPS MC jobs.

Python script	Description	Notes
ap_job.py	Generate A-primes using MadGraph4	
beam_job.py	Generate beam backgrounds using egs5	
dst_job.py	Create ROOT DST output from recon LCIO files	
egs5_beam_v3_job.py	Generate beam backgrounds using v3 of egs5 generator	
egs5_beam_v5_job.py	Generate beam backgrounds using v5 of egs5 generator	
lcio_concat_job.py	Concatenate a number of LCIO files together into a single output file	
lcio_count_job.py	Count the number of events in an LCIO file and throw an error if there are not enough	
tritrig_job.py	Generate trident events with trigger cuts	
tuple_job.py	Create ROOT tuple output from one or more input LCIO recon files	
wab_beam_job.py	Create wab-beam events from WAB and beam inputs and run in SLIC	
wab_beam_tri_job.py	Create wab-beam-tri events from wab-beam and tritrig inputs and run in SLIC	
wab_job.py	Generate WAB events in MadGraph4	

Job Scripts

All job scripts follow a specific structure.

First, necessary dependencies are imported.

```
from hpsmc.job import Job
from hpsmc.generators import MG4
```

Next the job is created and the parameters are fetched into a local variable.

```
job = Job(name="AP job")
job.initialize()
params = job.params
```

One or more components should be added to the job, for instance an event generator to create some LHE files.

```
# generate A-prime events using Madgraph 4
ap = MG4(name="ap",
        run_card="run_card_"+params.run_params+".dat",
        params={"APMASS": params.apmass},
        outputs=[filename],
        nevents=params.nevents)
```

Finally, the components should be added to the job and the job should be run.

```
job.components = [ap]
job.run()
```

The specific way that input and output files are used depends on the job script.

Typically, input files are read in without alteration, and (some) scripts can process multiple inputs while some cannot (depending on the particularities of the tools being used).

Output files written to the local "scratch" directory may be based on the name of input files or in some cases will be particular to a given script.

You must know the names of the output files in order to include them as output in your JSON parameters.