Starterkit tutorial with new DaVinci

LHCb StarterKit-For-All Full Run 3 Edition

Abhijit Mathad & Davide Fazzini
on behalf of the DPA WP3

14-18 March 2022

How to run a DaVinci job?

- New application has been implemented with the aim to ensure the largest flexibility to the **user**
- Different ways to run a DaVinci job according to the use case
- Two different commands can be used:

1. ./run *davinci* [options] command [user_options]
   - new syntax useful to exploit the Click potential
   - allow a greater flexibility in passing argument via command line
   - different methods for running jobs according to the input data
2. ./run *gaudirun.py* [options] user_options.py:
   - standard method for running LHCb applications
   - user_options.py can be configured as a Moore-like option file
   - using the same strategy used for running DaVinci in Production

New method with Click

- Main function is written in DaVinciSys/script/davinci
- Based on a dedicated implementation of Gaudi and Click
- New default running command:

./run davinci [option_davinci] run-mc [option_command]

option_davinci

- **--export**:
  dump configuration [.opts]
- **--dry-run**:
  configure without run the job

option_command

- **--inputfiledb, (-i)**:
  key and DB with input files
- **--joboptfile, (-j)**:
  file with job option list
- **--extra_arg**:
  any DV option

- Helper can be invoked with **--help** after ”davinci” or ”run-mc”
- NB: removing run-mc subcommand in the future releases:

./run davinci [option_davinci] [option_command]

New method with Click (II)

- Main elements for configuring a DaVinci jobs are:
  - a set of input files

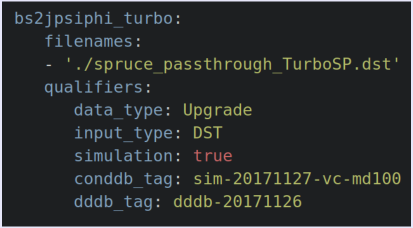
Input files

myDB.yaml

- Set with *--i* option
- Args:

  Key: *bs2jpsiphi_turbo*

  Location: *path/to/myDB.yaml*
- Input files location and related qualifiers are collected in a .yaml database



```
bs2jpsiphi_turbo:
  filenames:
  - './spruce_passthrough_TurboSP.dst'
  qualifiers:
    data_type: Upgrade
    input_type: DST
    simulation: true
    conddb_tag: sim-20171127-vc-md100
    dddb_tag: dddb-20171126
```

New method with Click (III)

- Main elements for configuring a DaVinci jobs are:
  - a set of input files
  - a list of option for running the job

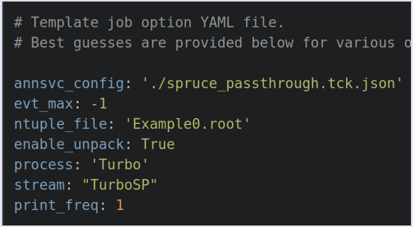
Job options

- Set with *--j* option
- Args:

  Location: *path/to/job_option.yaml*

- Job option can be collected in a dictionary in a dedicated .py or .yaml file

job_options.yaml



```yaml
# Template job option YAML file.
# Best guesses are provided below for various o

annsvc_config: './spruce_passthrough.tck.json'
evt_max: -1
ntuple_file: 'Example0.root'
enable_unpack: True
process: 'Turbo'
stream: "TurboSP"
print_freq: 1
```

- Full list of DV options available at options_default.py

New method with Click (IV)

- Main elements for configuring a DaVinci jobs are:
  - a set of input files
  - a list of option for running the job
  - a code containing the user algorithms to be run in the job

User algorithms

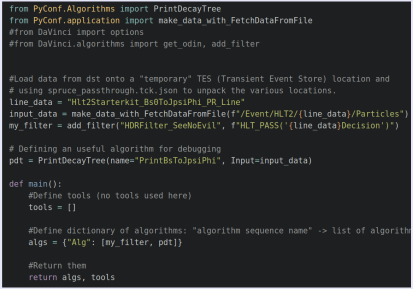
- Set with *--user_algorithms* option
- Args:

  Py-module: *path/user_algs:main*
- Algorithm can be imported as output of a main function
- User algorithms can be passed:

  - by command line

  - in job_option.yaml with *user_algorithms* : *"path/user_algs:main"*

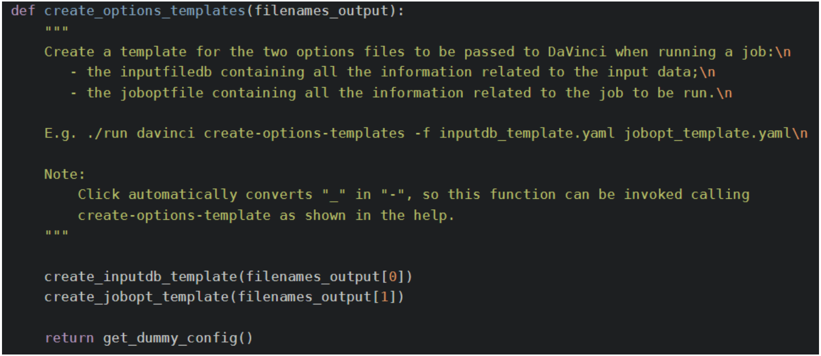
user_algs.py



```
from PyConf.Algorithms import PrintDecayTree
from PyConf.application import make_data_with_FetchDataFromFile
#from DaVinci import options
#from DaVinci.algorithms import get_odin, add_filter


#Load data from dst onto a "temporary" TES (Transient Event Store) location and
# using spruce_passthrough.tck.json to unpack the various locations.
line_data = "Hlt2Starterkit_Bs0ToJpsiPhi_PR_Line"
input_data = make_data_with_FetchDataFromFile(f"/Event/HLT2/{line_data}/Particles")
my_filter = add_filter("HDRFilter_SeeNoEvil", f"HLT_PASS('{line_data}Decision')")

# Defining an useful algorithm for debugging
pdt = PrintDecayTree(name="PrintBsToJpsiPhi", Input=input_data)

def main():
    #Define tools (no tools used here)
    tools = []

    #Define dictionary of algorithms: "algorithm sequence name" -> list of algorith
    algs = {"Alg": [my_filter, pdt]}

    #Return them
    return algs, tools
```

Templates for -*i* and -*j* argument

- A template for the .yaml for the input fileDB and job option files can be created using a dedicated method
- Method takes the template names as input, via -*f* argument



```
def create_options_templates(filenames_output):
    """
    Create a template for the two options files to be passed to DaVinci when running a job:\n
      - the inputfiledb containing all the information related to the input data;\n
      - the joboptfile containing all the information related to the job to be run.\n

    E.g. ./run davinci create-options-templates -f inputdb_template.yaml jobopt_template.yaml\n

    Note:
        Click automatically converts "_" in "-", so this function can be invoked calling
        create-options-template as shown in the help.
    """

    create_inputdb_template(filenames_output[0])
    create_jobopt_template(filenames_output[1])

    return get_dummy_config()
```

- Templates are created setting options to the default values

Standard LHCb method

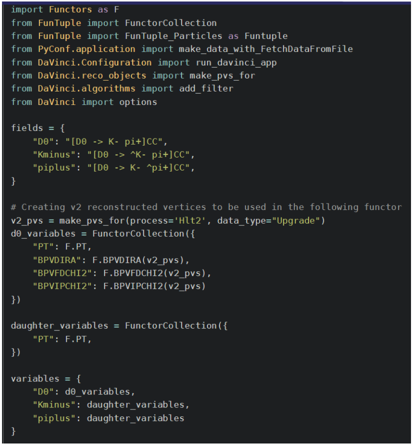
- DaVinci can still be run with the gaudirun.py command (not covered in this tutorial):

```
./run gaudirun.py options.py
```

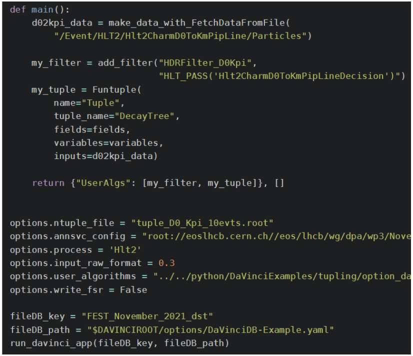
- option.py includes all the information needed by the user:
  - values of the DaVinci options
  - list of functors and branches for the FunTuple configuration
  - etc.
- Invoking *run_davinci_app(fileDB_key, fileDB_path)* at the end of the script

Example of DaVinci job running with gaudirun.py

- Example for running DaVinci with FunTuple on a hlt2 .dst
- Full code at option_davinci_tupling_from_hlt2_gaudirun.py



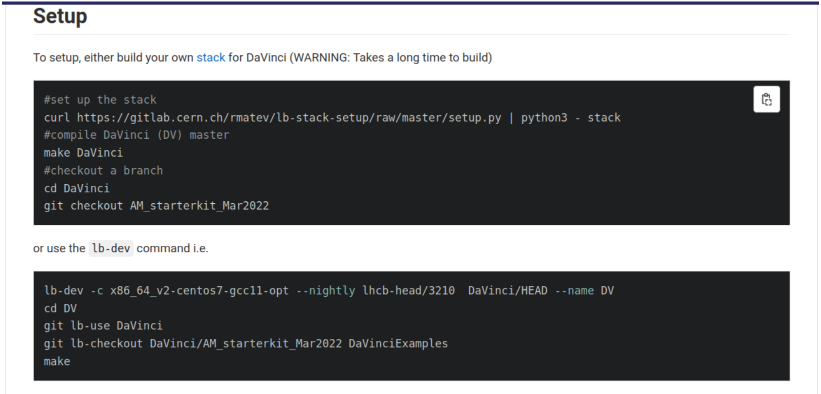
```
import Functors as F
from FunTuple import FunctorCollection
from FunTuple import FunTuple_Particles as Funtuple
from PyConf.application import make_data_with_FetchDataFromFile
from DaVinci.Configuration import run_davinci_app
from DaVinci.reco_objects import make_pvs_for
from DaVinci.algorithms import add_filter
from DaVinci import options

fields = {
    "D0": "[D0 -> K- pi+]CC",
    "Kminus": "[D0 -> ^K- pi+]CC",
    "piplus": "[D0 -> K- ^pi+]CC",
}

# Creating v2 reconstructed vertices to be used in the following functor
v2_pvs = make_pvs_for(process='Hlt2', data_type="Upgrade")
d0_variables = FunctorCollection({
    "PT": F.PT,
    "BPVDIRA": F.BPVDIRA(v2_pvs),
    "BPVFDCHI2": F.BPVFDCHI2(v2_pvs),
    "BPVIPCHI2": F.BPVIPCHI2(v2_pvs)
})

daughter_variables = FunctorCollection({
    "PT": F.PT,
})

variables = {
    "D0": d0_variables,
    "Kminus": daughter_variables,
    "piplus": daughter_variables
}
```



```
def main():
    d02kpi_data = make_data_with_FetchDataFromFile(
        "/Event/HLT2/Hlt2CharmD0ToKmPipLine/Particles")

    my_filter = add_filter("HDRFilter_D0Kpi",
                           "HLT_PASS('Hlt2CharmD0ToKmPipLineDecision')")
    my_tuple = Funtuple(
        name="Tuple",
        tuple_name="DecayTree",
        fields=fields,
        variables=variables,
        inputs=d02kpi_data)

    return {"UserAlgs": [my_filter, my_tuple]}, []


options.ntuple_file = "tuple_D0_Kpi_10evts.root"
options.annsvc_config = "root://eoslhcb.cern.ch//eos/lhcb/wg/dpa/wp3/Nove
options.process = 'Hlt2'
options.input_raw_format = 0.3
options.user_algorithms = "../../python/DaVinciExamples/tupling/option_da
options.write_fsr = False

fileDB_key = "FEST_November_2021_dst"
fileDB_path = "$DAVINCIROOT/options/DaVinciDB-Example.yaml"
run_davinci_app(fileDB_key, fileDB_path)
```

- Exercise: convert the tutorial scripts from *davinci* to *gaudirun.py*

Live coding session: setup your environment



Setup

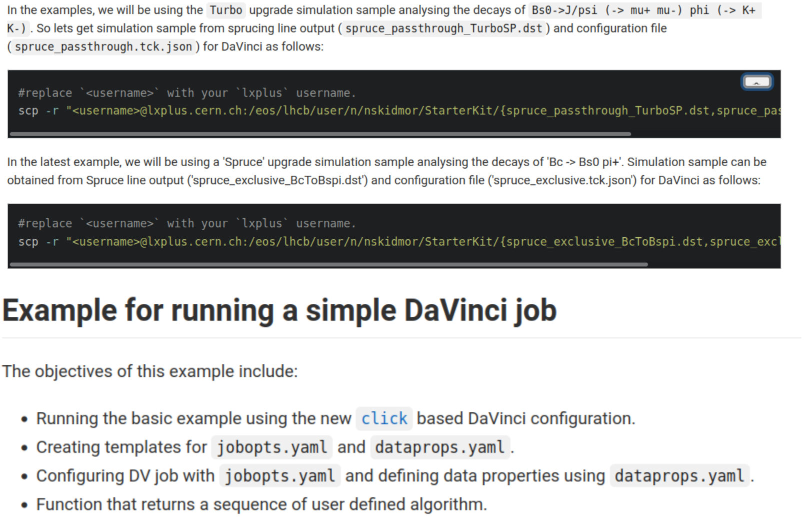
To setup, either build your own stack for DaVinci (WARNING: Takes a long time to build)

```
#set up the stack
curl https://gitlab.cern.ch/rmatev/lb-stack-setup/raw/master/setup.py | python3 - stack
#compile DaVinci (DV) master
make DaVinci
#checkout a branch
cd DaVinci
git checkout AM_starterkit_Mar2022
```

or use the `lb-dev` command i.e.

```
lb-dev -c x86_64_v2-centos7-gcc11-opt --nightly lhcb-head/3210  DaVinci/HEAD --name DV
cd DV
git lb-use DaVinci
git lb-checkout DaVinci/AM_starterkit_Mar2022 DaVinciExamples
make
```

Live coding session: download your input files



In the examples, we will be using the `Turbo` upgrade simulation sample analysing the decays of `Bs0->J/psi (-> mu+ mu-) phi (-> K+ K-)`. So lets get simulation sample from sprucing line output (`spruce_passthrough_TurboSP.dst`) and configuration file (`spruce_passthrough.tck.json`) for DaVinci as follows:

```
#replace `<username>` with your `lxplus` username.
scp -r "<username>@lxplus.cern.ch:/eos/lhcb/user/n/nskidmor/StarterKit/{spruce_passthrough_TurboSP.dst,spruce_pas
```

In the latest example, we will be using a 'Spruce' upgrade simulation sample analysing the decays of 'Bc -> Bs0 pi+'. Simulation sample can be obtained from Spruce line output ('spruce_exclusive_BcToBspi.dst') and configuration file ('spruce_exclusive.tck.json') for DaVinci as follows:

```
#replace `<username>` with your `lxplus` username.
scp -r "<username>@lxplus.cern.ch:/eos/lhcb/user/n/nskidmor/StarterKit/{spruce_exclusive_BcToBspi.dst,spruce_excl
```

Example for running a simple DaVinci job

The objectives of this example include:

- Running the basic example using the new `click` based DaVinci configuration.
- Creating templates for `jobopts.yaml` and `dataprops.yaml`.
- Configuring DV job with `jobopts.yaml` and defining data properties using `dataprops.yaml`.
- Function that returns a sequence of user defined algorithm.

- Full Example: link