Table of Contents

Production API

Introduction to the Production API

The Production API provides methods for production creation and also the possibility for generating workflow (XML) templates (editable via the WF Editor). The main purpose of this API is to condense the ~300 lines of old workflows into something more manageable for a wider group of people. Initially the API supports simulation and reconstruction type workflows only (e.g. stripping is yet to be added).

The building blocks of the API are the Gaudi Application Step containing GaudiApplication, LogChecker and BookkeepingReport modules and the Finalization Step.

For information there are currently two files in DIRAC containing the Production API:

The 2 distinct results of the finalization are explained below:

- LHCbDIRAC/Interfaces/API/Production.py - contains all the API methods;
- LHCbDIRAC/Interfaces/API/ProductionTemplates.py - to generate templates and provide examples.

Production.py inherits from LHCbJob.py so those who have a familiarity with the DIRAC API should be quite comfortable with the methods allowing some common settings to be applied at the production level if desired e.g.

```
setSystemConfig('slc4_ia32_gcc34')
setDestination('LCG.MySiteToTest.ch')
setBannedSites('LCG.CNAF.it')
setCPUTime('300000')
setLogLevel('debug')
...
```

One example of where the Production API can make life easier is by wrapping around these functions e.g. it may be common to ban all Tier-1 sites from a MC simulation production so there is a method to do just that:

```
banTier1s()
```

without remembering or typing the Tier-1 site names

Optimizations still to be performed:

- Stripping is yet to be fully implemented (pending something to test)
- Storing production parameters after publishing the workflow - plenty of scope for useful things to be available here: output data directories , ...
- Factoring out the options into a separate LHCbDIRAC/Core/Utilities/ProductionOptions.py utility for convenience (this will likely evolve faster than the API)
- Improved type checking is not in place purely due to time constraints.

A typical Gauss, Boole simulation workflow explained

This template is also available in LHCbDIRAC/Interfaces/API/ProductionTemplates.py and is included in full here before adding commentary line by line:

```
from LHCbDIRAC.Interfaces.API.Production import *
gaussBoole = Production()
gaussBoole.setProdType('MCSimulation')
gaussBoole.setWorkflowName('Production_GaussBoole')
```

```
gaussBoole.setWorkflowDescription('An example of Gauss + Boole, saving all outputs.')
gaussBoole.setBKParameters('test','2008','MC-Test-v1','Beam7TeV-VeloClosed-MagDown')
gaussBoole.setDBTags('sim-20090112','head-20090112')
gaussOpts = 'Gauss-2008.py;Beam7TeV-VeloClosed-MagDown.py;$DECFILESROOT/options/@{eventType}.opts
gaussBoole.addGaussStep('v36r2','Pythia','2',gaussOpts,eventType='57000001',extraPackages='AppCon
gaussBoole.addBooleStep('v17r2p1','digi','Boole-2008.py',extraPackages='AppConfig.v2r2')
gaussBoole.addFinalizationStep(sendBookkeeping=True,uploadData=True,uploadLogs=True,sendFailover=
gaussBoole.banTier1s()
gaussBoole.setWorkflowLib('v9r9')
gaussBoole.setFileMask('sim;digi')
gaussBoole.createWorkflow()
```

First the API must be imported:

```
from LHCbDIRAC.Interfaces.API.Production import *
```

The production object encapsulates the workflow and should be instantiated:

```
gaussBoole = Production()
```

The production type should always be specified, allowed production types include:

- MCSimulation
- DataReconstruction
- Merge
- MCStripping - yet to be implemented
- DataStripping - yet to be implemented

and protection is in place to ensure only these types can be specified.

```
gaussBoole.setProdType('MCSimulation')
```

The workflow name is the string given to the workflow XML file itself, this is what appears by default in the production monitoring page.

```
gaussBoole.setWorkflowName('Production_GaussBoole')
```

The workflow description can go into more detail about the workflow and (at least used to be) accessible from the production monitoring page.

```
gaussBoole.setWorkflowDescription('An example of Gauss + Boole, saving all outputs.')
```

The Production() object takes care of setting the BK processing pass for the production and each parameter will be explained in more detail below:

- configName - the BK configuration name e.g. 'MC'
- configVersion - the BK configuration version e.g. '2009'
- groupDescription - this corresponds to the processing pass index 'Group Description' field
- conditions - whether DataTaking or Simulation conditions both have a meaningful name

when publishing a production workflow to the production management system the create() method (see below) gives you the option to publish directly to the BK as well or to generate a script locally to perform the same action.

The groupDescription field can be chosen anew or the existing names can be reused (check via ***dirac-bookkeeping-pass-index-list***). If not already present the new processing pass will be added automatically.

Examples for the conditions parameter include:

- DataTaking6137
- Beam7TeV-VeloClosed-MagDown

in both cases the specific tag must be predefined in the Bookkeeping (for Simulation Conditions the parameters are retrieved from there before publishing the production processing pass).

```
gaussBoole.setBKParameters('test','2008','MC-Test-v1','Beam7TeV-VeloClosed-MagDown')
```

All productions (or production requests) must be defined with database tags for the conditions and detector description. The setDBTags() method takes the following arguments:

- conditions e.g. sim-20090112
- detector e.g. head-20090112

```
gaussBoole.setDBTags('sim-20090112','head-20090112')
```

The options files to be used for a given Gaudi Application Step are specified as a string separated by a colon or as a python list ![]. Standard workflow parameters can be set in the names of the options files (more on this in the examples below).

```
gaussOpts = 'Gauss-2008.py;Beam7TeV-VeloClosed-MagDown.py;$DECFILESROOT/options/@{eventType}.opts
```

The Gauss Gaudi Application Step of the workflow can now be defined. The full interface is available in Production.py and there are further examples below.

```
gaussBoole.addGaussStep('v36r2','Pythia','2',gaussOpts,eventType='57000001',extraPackages='AppCon
```

The Boole Gaudi Application Step can now also be defined. Note that it is unnecessary to specify the eventType again this is taken by default from the previous step (but is possible to override this). The ordering of the addStep() methods also dictates the input / output chain e.g. the outputs of Gauss are automatically fed to Boole in this workflow.

```
gaussBoole.addBooleStep('v17r2p1','digi','Boole-2008.py',extraPackages='AppConfig.v2r2')
```

The Finalization step should be defined for all productions. The below can be set without arguments since the default is True for the standard modules but for debugging purposes individual modules can be enabled / disabled if desired. For local testing the default can also be set to True since nothing happens without the JOBID environment variable being set (see the below section on running XML workflows).

```
gaussBoole.addFinalizationStep(sendBookkeeping=True,uploadData=True,uploadLogs=True,sendFailover=
```

As explained above, this is a quick way to ban all the Tier-1 sites for simulation productions.

```
gaussBoole.banTier1s()
```

The version of the workflow library can be set using the below method, this automatically appends the input sandbox LFN for the production jobs.

```
gaussBoole.setWorkflowLib('v9r9')
```

The output data files with correct naming convention are automatically produced for each Gaudi Application Step. The output data file mask is the final authority on which file extensions are to be retained for the given production and can be specified as a semi-colon separated string or a python list ![].

```
gaussBoole.setFileMask('sim;digi')
```

At this point by adding another API command the workflow XML can be created for local testing.

Generating the workflow for a production

In addition to the create() method explained below it is possible to generate the workflow for a given production using the createWorkflow() method, for the above example adding:

```
gaussBoole.createWorkflow()
```

then executing the resulting script would create a workflow XML file. Using the below script for running locally we can now test whether the production workflow is sane.

Running any XML workflow locally

Once you have an XML workflow file this section explains how to run it locally. The only caveat to mention here is that it has only been tested in the DIRAC Env scripts environment (some configuration changes may be required for this to work in the SetupProject Dirac case e.g. DIRACROOT, DIRACPYTHON, defining local SEs / DIRAC site name).

The following simple script will allow to run any workflow XML file (user / production / sam job XML from an input sandbox) locally:

```
import sys
xmlfile = str(sys.argv[1])
from DIRAC.Interfaces.API.Dirac import Dirac
from LHCbDIRAC.Interfaces.API.LHCbJob import LHCbJob
d = Dirac()
j = LHCbJob(xmlfile)
print d.submit(j,mode='local')
```

and is available along with all workflow templates by executing the ProductionTemplates.py file mentioned above in a directory of your choosing.

The only caveat to mention here is that the workflow library specified in the production is not currently downloaded (this can be done by hand with a ***dirac-dms-get-file*** or the local version is picked up). Also for testing workflows with input data files they must be accessible locally

For production jobs the finalization modules are disabled by default such that uploading output data / publishing BK records is not performed during local testing. Setting the JOBID environment variable to an integer will enable these modules if desired (but be especially careful in the Merging production case because input files would be removed).

The production create() method

create() will allow to publish the production to the production management system and to the BK. This currently relies on the conditions information being present in the worklow. Production parameters can also be added at this point (to be implemented). The create() method takes the following arguments and default values:

- publish=True
  - Set to True this will add the production to the production management system
  - Set to False this will not publish the production but allows printing of BK parameters /

generation of BK script with default production ID
- fileMask=''
  - Optional file mask (regular expression approach)
- bkQuery = { }
  - Optional dictionary containing the BK query parameters
- groupSize=1
  - Number of input files per job for data processing productions
- derivedProduction
  - Optionally the ID of the production from which to derive the input data
- bkScript=True
  - Set to True this will write a script that can be checked before publishing to the BK
  - Set to False this will automatically publish to the BK

The workflow XML is created regardless of the flags (with a default production ID of 12345 if publish=False).

It should be noted that as well as providing arguments to the create() it is also possible to use the following dedicated setting methods:

- setInputFileMask(fileMask)
  - Where fileMask is the reg expression to match input files
- setInputBKSelection(bkQuery)
  - Where bkQuery is the dictionary containing the BK query parameters
- setJobFileGroupSize(files)
  - Where files is an integer containing the number of files
- setAncestorProduction(prodID)
  - Where prodID is the ID of the production from which to derive the input data

When these methods are used their input overwrites what is supplied to create().

In addition to the explicit setting methods presented there are two more which can be useful:

- setProdGroup(groupString)
  - Where the groupString is a simple description of the production. examples are 'MC09', 'FEST'... This allows simple searching for production group on the production monitoring page.
- setProdPlugin(pluginName)
  - Where the pluginName is a string with the name of the plugin to be used when creating the jobs. For FEST reconstruction of the FULL stream this should be set to 'CCRC_RAW'.

Example 1 - From a typical simulation request email to events in the BK

In this case there was a request for a production with 'old' versions of the projects that don't support LHCbApp(). The standard options are printed by the Production.py API and it was sufficient to remove traces of the LHCbApp() module (relying on the defaults from the options). Note that the create() function here is used just to write a script for BK publishing, this allows to check that the processing pass is ok before making it visible (if it is ever to be made visible) and will only generate the workflow XML e.g. this can be tested locally before entering to the production system.

The production request for the above was an email and for information this is included below:

```
I am forwarding you this request for a production of 800K inclusive B events (event type 10000000
Gauss v35r1 and Boole v16r3.
```

```
This production will also need to use AppConfig v1r1 that is not picked up by default by Gauss v3

Only two steps have to be performed, you find them below but I restate them here for convenience
to correct the options for Gauss:

Step 1
   Gauss:                                v35r1
   EvtType                               10000000  ( Inclusive b)
   Number of events to process:          800,000
   Tag of DDDB and SIMCOND:              head-20081002
   Generator to be used:                 Pythia
   Configuration (as set in default):    Beam Energy:          7 TeV
                                         Velo:                 Closed
                                         Magnet polarity:      Negative
                                         Luminosity:           2x10^32
   Options for all jobs (in addition to $DECFILESROOT/options/10000000.opts)
       gaudirun.py  $GAUSSOPTS/Gauss-2008.py  $APPCONFIGROOT/options/Gauss/RICHmirrorMisalignment

Step 2
   Boole:                                v16r3
   Tag of DDDB and SIMCOND:              head-20081002
   Spillover:                            Off

   Options for all jobs: ( standard options )
       gaudirun.py  $BOOLEOPTS/Boole-2008.py
```

Now let's take a look at the production API script for this case, as mentioned above the standard API options use LHCbApp() but using the standard options without the LHCbApp() settings was sufficient for the above:

```
(DIRAC3-production)[paterson@lxplus223 ~]$ cat gaussBooleBInclusive.py
from LHCbDIRAC.Interfaces.API.Production import *
gaussBoole = Production()
gaussBoole.setProdType('MCSimulation')
gaussBoole.setWorkflowName('GaussBoole_500evt_inclusiveB_1')
gaussBoole.setWorkflowDescription('Gauss + Boole, saving sim + digi, 500 events, 800k requested.'
gaussBoole.setBKParameters('MC','2008','MC08-v1','Beam7TeV-VeloClosed-MagDown')
gaussBoole.setDBTags('head-20081002','head-20081002')
gaussOpts = 'Gauss-2008.py;$APPCONFIGROOT/options/Gauss/RICHmirrorMisalignments.py;$DECFILESROOT/

opts = "MessageSvc().Format = '%u % F%18W%S%7W%R%T %0W%M';MessageSvc().timeFormat = '%Y-%m-%d %H:
opts += """OutputStream("GaussTape").Output = "DATAFILE='PFN:@{outputData}' TYP='POOL_ROOTTREE' O
opts += 'from Configurables import SimInit;'
opts += 'GaussSim = SimInit("GaussSim");'
opts += 'GaussSim.OutputLevel = 2;'
opts += 'HistogramPersistencySvc().OutputFile = "@{applicationName}_@{STEP_ID}_Hist.root"'

gaussBoole.addGaussStep('v35r1','Pythia','500',gaussOpts,eventType='10000000',extraPackages='AppC

opts2 = """OutputStream("DigiWriter").Output = "DATAFILE='PFN:@{outputData}' TYP='POOL_ROOTTREE'
opts2 += "MessageSvc().Format = '%u % F%18W%S%7W%R%T %0W%M';MessageSvc().timeFormat = '%Y-%m-%d %
opts2 += 'HistogramPersistencySvc().OutputFile = "@{applicationName}_@{STEP_ID}_Hist.root"'

gaussBoole.addBooleStep('v16r3','digi','Boole-2008.py',outputSE='CERN-RDST',overrideOpts=opts2)

gaussBoole.addFinalizationStep(sendBookkeeping=True,uploadData=True,uploadLogs=True,sendFailover=
gaussBoole.banTier1s()
gaussBoole.setWorkflowLib('v9r9')
gaussBoole.setFileMask('sim;digi')
gaussBoole.setProdPriority('5')
gaussBoole.create(publish=True,bkScript=True)
```

Notice that the workflow was created and tested locally using the above recipe before proceeding to this point. The create(publish=True,bkScript=True) means that executing the script will create a production and a BK script (but not publish to the BK yet).

```
(DIRAC3-production)[paterson@lxplus223 ~]$ python gaussBooleBInclusive.py
2009-04-08 10:34:43 UTC gaussBooleBInclusive.py  INFO: Setting default outputSE to Tier1-RDST
2009-04-08 10:34:43 UTC gaussBooleBInclusive.py  INFO: Setting default outputSE to Tier1-RDST
We need to write piece of code to replace existent DefinitionsPool.__setitem__()
For now we ignore it for the Gaudi_App_Step
2009-04-08 10:34:43 UTC gaussBooleBInclusive.py  INFO: Found simulation conditions for Beam7TeV-V
2009-04-08 10:34:44 UTC gaussBooleBInclusive.py  INFO: Production 4614 successfully created
2009-04-08 10:34:44 UTC gaussBooleBInclusive.py  INFO: Writing BK publish script...
```

In the local directory the automatically generated BK publishing script is written e.g.

```
(DIRAC3-production)[paterson@lxplus223 ~]$ cat insertBKPass4614.py
# Bookkeeping publishing script created on Wed Apr  8 12:34:44 2009 by
# by $Id: ProductionAPI.txt,v 1.6 2011/06/27 01:00:14 stephane_2eguillaume_2eposs_40cern_2ech Exp
from LHCbDIRAC.BookkeepingSystem.Client.BookkeepingClient import BookkeepingClient
bkClient = BookkeepingClient()
bkDict = {'Production': 4614,'Steps': {'Step1': {'ApplicationName': 'Boole', 'ApplicationVersion'
print bkClient.addProduction(bkDict)
```

scrolling to the end you can see that the simulation conditions were retrieved from the BK for the specified tag (Beam7TeV-VeloClosed-MagDown).

After executing the above script you can see the result in the BK (e.g. taken today after we have produced the 800k requested):

```
(DIRAC3-production)[paterson@lxplus223 ~]$ dirac-bookkeeping-production-informations 4614
Production Info:
    Configuration Name: MC
    Configuration Version: 2008
    Event type: 10000000
Step0:Gauss-v35r1
      Option files: Gauss-2008.py;$APPCONFIGROOT/options/Gauss/RICHmirrorMisalignments.py;$DECFIL
      DDDb: head-20081002
      ConDDb: head-20081002
Step1:Boole-v16r3
      Option files: Boole-2008.py
      DDDb: head-20081002
      ConDDb: head-20081002
Number of jobs   3342
Total number of files: 6684
         SIM:1671
         DIGI:1671
         LOG:3342
Number of events [('SIM', 830043), ('DIGI', 831039)]
```

Now the production is fully created and standard command line tools can be used:

```
(DIRAC3-production)[paterson@lxplus223 ~]$ dirac-production-mc-extend 4614 100
Extended production 4614 by 100 jobs
(DIRAC3-production)[paterson@lxplus223 ~]$ dirac-production-set-automatic 4614
2009-04-08 10:37:21 UTC dirac-production-set-automatic.py/DiracProduction  INFO: Do you wish to c
Do you wish to change production 4614 with command "Automatic"?  [yes/no] : y
2009-04-08 10:37:22 UTC dirac-production-set-automatic.py/DiracProduction  INFO: Setting producti
```

in this case all events were produced in a day

Example 2 - Running a reconstruction (FEST in this case) production workflow from the template

Templates for both the express stream and full stream reconstruction are available in the ProductionTemplates.py file. The below example is for the express stream reconstruction:

```
from LHCbDIRAC.Interfaces.API.Production import *
expressStream = Production()
expressStream.setProdType('DataReconstruction')
expressStream.setWorkflowName('Production_EXPRESS_FEST')
expressStream.setWorkflowDescription('An example of the FEST EXPRESS stream production')
expressStream.setBKParameters('Fest','Fest','FEST-Reco-v1','DataTaking6153')
expressStream.setDBTags('head-20090112','head-20090112')

brunelOpts = '$APPCONFIGOPTS/Brunel/FEST-200903.py' #;$APPCONFIGOPTS/UseOracle.py'
brunelEventType = '91000000'
brunelData='LFN:/lhcb/data/2009/RAW/EXPRESS/FEST/FEST/44878/044878_0000000002.raw'
brunelSE='CERN-RDST'
brunelMaxEvts='100'
expressStream.addBrunelStep('v34r2','rdst',brunelOpts,extraPackages='AppConfig.v2r2',eventType=br


dvOpts = '$APPCONFIGOPTS/DaVinci/DVMonitorDst.py'
expressStream.addDaVinciStep('v22r1','dst',dvOpts,extraPackages='AppConfig.v2r2',histograms=True)

expressStream.addFinalizationStep(sendBookkeeping=True,uploadData=True,uploadLogs=True,sendFailov
expressStream.setWorkflowLib('v9r9')
expressStream.setFileMask('rdst;root')
expressStream.setProdPriority('9')
expressStream.create(publish=False)
```

Note that we use create(publish=False) here so executing the above script creates an example BK script for inspection with the default production ID e.g. no production was created:

```
(DIRAC3-production)[paterson@lxplus243 ~]$ cat insertBKPass12345.py
# Bookkeeping publishing script created on Sun Apr  5 17:15:17 2009 by
# by $Id: ProductionAPI.txt,v 1.6 2011/06/27 01:00:14 stephane_2eguillaume_2eposs_40cern_2ech Exp
from LHCbDIRAC.BookkeepingSystem.Client.BookkeepingClient import BookkeepingClient
bkClient = BookkeepingClient()
bkDict = {'Production': 12345, 'Steps': {'Step1': {'ApplicationName': 'DaVinci', 'ApplicationVers
print bkClient.addProduction(bkDict)
```

as well as a workflow XML file suitable for testing (in this case named Production_EXPRESS_FEST.xml). If we run the resulting workflow locally as prescribed above we can test the workflow e.g.

```
(DIRAC3-production)[paterson@lxplus243 ~]$ python runLocal.py Production_EXPRESS_FEST.xml
```

since JOBID is not defined this does not do anything intrusive e.g. no data upload or BK records sent etc. However, this does allow to check that the options are working:

```
2009-04-05 15:16:25 UTC DstWriter            INFO Data source: EventDataSvc output: DATAFILE='PFN
2009-04-05 15:16:27 UTC EventSelector        INFO Stream:EventSelector.DataStreamTool_1 Def:DATAF
2009-04-05 15:16:27 UTC ApplicationMgr       INFO Application Manager Initialized successfully
2009-04-05 15:16:27 UTC ApplicationMgr       INFO Application Manager Started successfully
2009-04-05 15:16:27 UTC EventSelector.D...   INFO Compression:0 Checksum:1
2009-04-05 15:16:27 UTC EventPersistenc...   INFO Added successfully Conversion service:PoolRootK
2009-04-05 15:16:27 UTC EventPersistenc...   INFO Added successfully Conversion service:LHCb::Raw
2009-04-05 15:16:29 UTC BrunelInit           INFO Evt 257,  Run 44878,  Nr. in job = 1
2009-04-05 15:16:29 UTC ChronoStatSvc        INFO  Number of skipped events for MemStat-1
2009-04-05 15:16:31 UTC IODataManager        INFO Referring to dataset 00012345_00012345_1.rdst b
2009-04-05 15:16:31 UTC 00012345_000123...SUCCESS Root file version:51800
2009-04-05 15:16:34 UTC BrunelInit           INFO Evt 616,  Run 44878,  Nr. in job = 2
2009-04-05 15:16:34 UTC BrunelInit           INFO Evt 795,  Run 44878,  Nr. in job = 3

...

2009-04-05 15:17:18 UTC BrunelInit           INFO Evt 38195,  Run 44878,  Nr. in job = 94
2009-04-05 15:17:19 UTC BrunelInit           INFO Evt 38519,  Run 44878,  Nr. in job = 95
2009-04-05 15:17:19 UTC BrunelInit           INFO Evt 39016,  Run 44878,  Nr. in job = 96
```

```
2009-04-05 15:17:19 UTC BrunelInit           INFO Evt 40758,  Run 44878,  Nr. in job = 97
2009-04-05 15:17:20 UTC BrunelInit           INFO Evt 39173,  Run 44878,  Nr. in job = 98
2009-04-05 15:17:20 UTC BrunelInit           INFO Evt 41193,  Run 44878,  Nr. in job = 99
2009-04-05 15:17:21 UTC BrunelInit           INFO Evt 40540,  Run 44878,  Nr. in job = 100
2009-04-05 15:17:21 UTC ApplicationMgr       INFO Application Manager Stopped successfully
2009-04-05 15:17:21 UTC BrunelInit        SUCCESS ===============================================
2009-04-05 15:17:21 UTC BrunelInit        SUCCESS 100 events processed
2009-04-05 15:17:21 UTC BrunelInit        SUCCESS ===============================================
2009-04-05 15:17:21 UTC BrunelEventCount     INFO 100 events processed

...

2009-04-05 15:17:22 UTC *****Chrono*****     INFO ********************************************
2009-04-05 15:17:22 UTC *****Chrono*****     INFO  The Final CPU consumption ( Chrono ) Table (or
2009-04-05 15:17:22 UTC *****Chrono*****     INFO ********************************************
2009-04-05 15:17:22 UTC DetailedMateria...   INFO Time User   : Tot= 13.6  [s] Ave/Min/Max=  394(
2009-04-05 15:17:22 UTC ChronoStatSvc        INFO Time User   : Tot= 51.5  [s]
2009-04-05 15:17:22 UTC *****Chrono*****     INFO ********************************************
2009-04-05 15:17:22 UTC ChronoStatSvc.f...   INFO  Service finalized succesfully
2009-04-05 15:17:23 UTC ApplicationMgr       INFO Application Manager Finalized successfully
2009-04-05 15:17:23 UTC ApplicationMgr       INFO Application Manager Terminated successfully
```

also that things like the bookkeeping records created during the Gaudi Application steps look reasonable:

```
(DIRAC3-production)[paterson@lxplus243 ~]$ cat bookkeeping_00012345_00012345_2.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Job SYSTEM "book.dtd">
<Job ConfigName="Fest" ConfigVersion="Fest" Date="2009-04-05" Time="17:17">
  <TypedParameter Name="Production" Value="00012345" Type="Info"/>
  <TypedParameter Name="DiracJobId" Value="00012345" Type="Info"/>
  <TypedParameter Name="Name" Value="00012345_00012345_2" Type="Info"/>
  <TypedParameter Name="JobStart" Value="2009-04-05 17:17" Type="Info"/>
  <TypedParameter Name="JobEnd" Value="2009-04-05 17:17" Type="Info"/>
  <TypedParameter Name="Location" Value="LCG.CERN.ch" Type="Info"/>
  <TypedParameter Name="WorkerNode" Value="lxplus221.cern.ch" Type="Info"/>
  <TypedParameter Name="ProgramName" Value="DaVinci" Type="Info"/>
  <TypedParameter Name="ProgramVersion" Value="v22r1" Type="Info"/>
  <TypedParameter Name="DiracVersion" Value="v4r10p0" Type="Info"/>
  <TypedParameter Name="FirstEventNumber" Value="1" Type="Info"/>
  <TypedParameter Name="StatisticsRequested" Value="-1" Type="Info"/>
  <TypedParameter Name="NumberOfEvents" Value="100" Type="Info"/>
  <InputFile    Name="/lhcb/data/2009/RDST/00012345/0001/00012345_00012345_1.rdst"/>
  <OutputFile   Name="/lhcb/data/2009/DST/00012345/0001/00012345_00012345_2.dst" TypeName="DST" T
    <Parameter  Name="EventTypeId"     Value="91000000"/>
    <Parameter  Name="EventStat"       Value="100"/>
    <Parameter  Name="FileSize"        Value="5572023"/>
    <Parameter  Name="MD5Sum"        Value="1e8e380d4657bc541e3b0fac5fbdd931"/>
    <Parameter  Name="Guid"        Value="D87AD5EA-F421-DE11-863C-000423D94CB8"/>
  </OutputFile>
  <OutputFile   Name="/lhcb/data/2009/HIST/00012345/0001/DaVinci_00012345_00012345_2_Hist.root" T
    <Parameter  Name="EventTypeId"     Value="91000000"/>
    <Parameter  Name="EventStat"       Value="100"/>
    <Parameter  Name="FileSize"        Value="4156"/>
    <Parameter  Name="MD5Sum"        Value="684852670cf120f25954058fac0784e3"/>
    <Parameter  Name="Guid"        Value="68485267-0CF1-20F2-5954-058FAC0784E3"/>
  </OutputFile>
  <OutputFile   Name="/lhcb/data/2009/LOG/00012345/0001/00012345/DaVinci_00012345_00012345_2.log"
    <Replica Name="http://lhcb-logs.cern.ch/storage/lhcb/data/2009/LOG/00012345/0001/00012345/DaV
    <Parameter  Name="MD5Sum"        Value="9576851cffb042325eba51caa458ec69"/>
    <Parameter  Name="Guid"        Value="9576851C-FFB0-4232-5EBA-51CAA458EC69"/>
  </OutputFile>
</Job>
```

as well as the POOL XML catalog for the produced files:

```
(DIRAC3-production)[paterson@lxplus243 ~]$ cat pool_xml_catalog.xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!-- Edited By PoolXMLCatalog.py -->
<!DOCTYPE POOLFILECATALOG SYSTEM "InMemory">
<POOLFILECATALOG>


  <File ID="a39514a4-0808-11de-9206-00188b8565aa">
     <physical>
       <pfn filetype="ROOT_All" name="root:castor://castorlhcb.cern.ch:9002/?svcClass=lhcbraw&amp
     </physical>
     <logical>
       <lfn name="/lhcb/data/2009/RAW/EXPRESS/FEST/FEST/44878/044878_0000000002.raw"/>
     </logical>
   </File>

  <File ID="04E183BE-F421-DE11-9F74-000423D94CB8">
    <physical>
      <pfn filetype="ROOT_All" name="00012345_00012345_1.rdst"/>
    </physical>
    <logical/>
  </File>

  <File ID="D87AD5EA-F421-DE11-863C-000423D94CB8">
    <physical>
      <pfn filetype="ROOT_All" name="00012345_00012345_2.dst"/>
    </physical>
    <logical/>
  </File>

</POOLFILECATALOG>
```

So in summary local testing of a workflow with default production ID / production job ID is useful for seeing exactly what the workflow does and the sanity of job options etc. etc.

Example 3 - Testing a merging production

In the below example I use a non-standard set of data but the template can always be modified for the real case. Here (because data was available and deletable) I try to merge two FEST reconstruction RDSTs.

The template for merging productions looks like the following (note the absence of the inputProduction parameter, this is in Example 4 but not necessary for local testing):

```
merge = Production()
merge.setProdType('Merge')
merge.setWorkflowName('Merge_Test_1')
merge.setWorkflowDescription('An example of merging two inputs.')
merge.setBKParameters('Fest','Fest','FEST-Reco-v0','DataTaking6153')
merge.setDBTags('head-20090112','head-20090112')
mergeEventType = '91000000'
mergeData=[]
mergeData.append('/lhcb/data/2009/RDST/00004601/0000/00004601_00000059_1.rdst')
mergeData.append('/lhcb/data/2009/RDST/00004601/0000/00004601_00000097_1.rdst')
mergeDataType='RDST'
mergeSE='Tier1-RDST'
merge.addMergeStep('v26r3',optionsFile='$STDOPTS/PoolCopy.opts',eventType=mergeEventType,inputDat
#removeInputData is False by default (all other finalization modules default to True)
merge.addFinalizationStep(removeInputData=True)
merge.setWorkflowLib('v9r9')
merge.setFileMask('dst')
merge.setProdPriority('9')
merge.createWorkflow()
```

The addFinalizationStep(removeInputData=True) means that all the standard finalization modules are picked up (they default to True) but the removeInputData module is activated. This is a new workflow module that will attempt to remove the input data files only after the output data has been successfully uploaded to an SE.

The above recipe can be used in the same way to test this production workflow. Performing this locally we see the following application standard output:

```
...
EventSelector     SUCCESS Reading Event record 117581. Record number within stream 2: 58785
EventSelector     SUCCESS Reading Event record 117582. Record number within stream 2: 58786
EventLoopMgr         INFO No more events in event selection
ApplicationMgr       INFO Application Manager Stopped successfully
InputCopyStream      INFO Events output: 117582
EventLoopMgr         INFO Histograms converted successfully according to request.
ToolSvc              INFO Removing all tools created by ToolSvc
PoolRootTreeEvt...   INFO Disconnected data IO:0697F867-4424-DE11-BDEA-000423D986F4[00012345_0001
PoolRootTreeEvt...   INFO Disconnected data IO:3657FF42-AD0C-DE11-BA92-0030487C6B62[castor://cast
d/lhcb/data/2009/RDST/00004601/0000/00004601_00000097_1.rdst]
PoolRootTreeEvt...   INFO Disconnected data IO:9A902E53-AD0C-DE11-BCC5-0030487C600A[castor://cast
d/lhcb/data/2009/RDST/00004601/0000/00004601_00000059_1.rdst]
ApplicationMgr       INFO Application Manager Finalized successfully
ApplicationMgr       INFO Application Manager Terminated successfully
```

And the produced file is of the expected size:

```
(DIRAC3-production)[paterson@lxplus222 /tmp/paterson/mergeTest]$ ls -hal 00012345_00012345_1.rdst
-rw-r--r--  1 paterson z5 6.2G Apr  8 16:26 00012345_00012345_1.rdst
```

The following options were automatically generated:

```
//////////////////////////////////////////////////////
# Dynamically generated options in a production or analysis job

from Gaudi.Configuration import *
OutputStream("InputCopyStream").Output = "DATAFILE='PFN:00012345_00012345_1.rdst' TYP='POOL_ROOTT
EventSelector().Input=[ "DATAFILE='LFN:/lhcb/data/2009/RDST/00004601/0000/00004601_00000059_1.rds
 "DATAFILE='LFN:/lhcb/data/2009/RDST/00004601/0000/00004601_00000097_1.rdst' TYP='POOL_ROOTTREE'

FileCatalog().Catalogs= ["xmlcatalog_file:pool_xml_catalog.xml"]
ApplicationMgr().EvtMax =-1
```

The production input data in the above template will be overwritten automatically for each job after the production is created and the BK query / regular expression is suitably defined and the jobs are submitted.

Example 4 - Creating a merging production

This assumes a production already exists in the system, created with the API that will have set the BKProcessingPass information in the workflow body. For the below example we will assume the recent B inclusive production outputs need to be merged (production 4614).

The template for merging productions looks similar to Example 3 but with the inputProduction parameter specified:

```
merge = Production()
merge.setProdType('Merge')
merge.setWorkflowName('Merge_BInclusive_4614')
merge.setWorkflowDescription('An example of merging two inputs.')
merge.setBKParameters('MC','2008','MC08-v1','DataTaking6153')
merge.setDBTags('head-20081002','head-20081002')
mergeEventType = '10000000'
```

```
mergeData=[]
mergeData.append('/lhcb/MC/2008/DIGI/00004614/0000/00004614_00002000_2.digi')
mergeData.append('/lhcb/MC/2008/DIGI/00004614/0000/00004614_00001998_2.digi')
mergeDataType='DIGI'
mergeSE='Tier1-RDST'
mergeProd = 4614
merge.addMergeStep('v26r3',optionsFile='$STDOPTS/PoolCopy.opts',eventType=mergeEventType,inputDat
#removeInputData is False by default (all other finalization modules default to True)
merge.addFinalizationStep(removeInputData=True)
merge.setWorkflowLib('v9r11')
merge.setFileMask('digi')
merge.setProdPriority('9')
merge.createWorkflow()
```

Setting the inputProduction parameter means that before doing anything the API will attempt to add the processing pass of the previous production to the current one (preserving the step numbering of course). If we create the workflow for this template we see:

```
2009-04-09 15:33:22 UTC mergingExample.py  INFO: Adding processing pass Step1 from production 461
2009-04-09 15:33:22 UTC mergingExample.py  INFO: {'ApplicationName': 'Boole', 'ApplicationVersion
2009-04-09 15:33:22 UTC mergingExample.py  INFO: Adding processing pass Step0 from production 461
2009-04-09 15:33:22 UTC mergingExample.py  INFO: {'ApplicationName': 'Gauss', 'ApplicationVersion
2009-04-09 15:33:22 UTC mergingExample.py  INFO: Default options for Merging are:
2009-04-09 15:33:22 UTC mergingExample.py  INFO: OutputStream("InputCopyStream").Output = "DATAFI
2009-04-09 15:33:22 UTC mergingExample.py  INFO: Setting workflow library LFN to LFN:/lhcb/applic
```

Indicating that the Gauss and Boole steps of the production 4614 were added. If we examine the generated code of the above template (by adding the 'print merge.createCode()' API method) this shows e.g.

```
...
j.BKProcessingPass = {'Step2': {'ApplicationName': 'LHCb', 'ApplicationVersion': 'v26r3', 'ExtraP
...
```

and you can see the additional LHCb merging step that has been added here. Although in this case the input production 4614 is visible in the BK we are using the workflow to retrieve this information so it doesn't have to be entered. The input production for merging can be published using create(publish=True,bkScript=True) meaning it is invisible until the resulting BK script is executed. The merging production will automatically pick up the processing pass information from this production via the inputProduction parameter. The merging production will then run with the step number corresponding to the last step of the input production + 1.

To run the above template it then becomes the same procedure as Example 1 (but for a production with defined input data).

To create the production and a local BK script using the regexp approach use:

```
mask = '/lhcb/MC/2008/DIGI/00004614.*\.digi$'
numberOfFiles=2
merge.create(publish=True,fileMask=mask,groupSize=numberOfFiles,bkScript=True)
```

to directly publish the merging production to the production management system and BK use:

```
mask = '/lhcb/MC/2008/DIGI/00004614.*\.digi$'
numberOfFiles=2
merge.create(publish=True,fileMask=mask,groupSize=numberOfFiles,bkScript=False)
```

That's all folks!
