Moving From DAX3 to DAX4

Changes:

- moved from **camelCase** to **snake_case** for functions
- `job.uses(file, link=`**`Link`**`.INPUT)` is now `job.uses(file, type=`**`LinkType`**`.INPUT)`
- `job.uses(file, type=LinkType.INPUT,` **`transfer`**`=True,` **`register`**`=True)` is now `job.uses(file, type=LinkType.INPUT,` **`stage_out`**`=True,` **`register_replica`**`=True`
- `add_PFN(`**`PFN`**`("uri", "site_name"))` is now just **`add_pfn(*uris, site="local"))`**
- `add_profile(`**`Profile`**`(namespace, key, value))` is now just `add_profile(namespace, key, value)`
- `job.uses(....)` is now split up into **`job.add_inputs(*inputs)`** and **`job.add_outputs(*outputs, stage_out=True, register_replica=True)`** (stage_out and register_replica would apply to all of *outputs, and so this can be called multiple times if different values of stage_out and register_replica must be set for some subset of *outputs)
- `transformation.uses(`**`exec`**`)` is now `transformation.uses(`**`*execs`**`)`
- `obj.`**`metadata`**`(key, value)` is now `obj.`**`add_metadata`**`(key, value)`
- `dag.`**`depends`**`(parent=some_parent, child=some_child)` is now `dag.`**`add_dependency`**`(parent, *children)`

Given what we have in DAX4 (middle column below), we could rewrite it to be something like what we have in the right most column below. Functionality it is the same. At least in this example, it looks a little less cluttered and simpler to read than the middle column while saving a few lines in the process...

DAX3 Black Diamond

```
#!/usr/bin/env python

from Pegasus.DAX3 import *

# Create a DAX
diamond = ADAG("diamond")

# Add some metadata
diamond.metadata("name",
"diamond")
diamond.metadata
("createdby", "Gideon Juve")

# Add input file to the DAX-
level replica catalog
a = File("f.a")
a.addPFN(PFN("gsiftp://site.
com/inputs/f.a","site"))
a.metadata("size", "1024")
diamond.addFile(a)

# Add executables to the
DAX-level replica catalog
e_preprocess = Executable
(namespace="diamond", name="
preprocess", version="4.0",
os="linux", arch="x86_64")
e_preprocess.metadata
("size", "2048")
e_preprocess.addPFN(PFN
("gsiftp://site.com/bin
/preprocess","site"))
diamond.addExecutable
(e_preprocess)

e_findrange = Executable
(namespace="diamond", name="
findrange", version="4.0",
os="linux", arch="x86_64")
e_findrange.addPFN(PFN
("gsiftp://site.com/bin
/findrange","site"))
diamond.addExecutable
(e_findrange)
```

DAX4 Black Diamond

```
#!/usr/bin/env python

from Pegasus.DAX4 import *

# Create a DAX
diamond = Workflow("diamond")

# Add some metadata
diamond.add_metadata("name",
"diamond")
diamond.add_metadata
("createdby", "Gideon Juve")

# Add input file to the DAX-
level replica catalog
a = File("f.a")
a.add_PFN("gsiftp://site.com
/inputs/f.a","site")
a.add_metadata("size",
"1024")
diamond.add_file(a)

# Add executables to the DAX-
level replica catalog
e_preprocess = Transformation
(namespace="diamond", name="
preprocess", version="4.0",
os="linux", arch="x86_64")
e_preprocess.add_metadata
("size", "2048")
e_preprocess.add_PFN
("gsiftp://site.com/bin
/preprocess","site")
diamond.add_transformation
(e_preprocess)

e_findrange = Transformation
(namespace="diamond", name="
findrange", version="4.0",
os="linux", arch="x86_64")
e_findrange.add_PFN
("gsiftp://site.com/bin
/findrange","site")
diamond.add_transformation
```

DAX4 Black Diamond, catalogs included

```
'''
------- Catalogs -------
Pegasus.DAX4.SiteCatalog
- add_site(name)
- get_site(name)
- has_site(name)
- write_catalog(path)
...

Pegasus.DAX4.ReplicaCatalog
- add_replica(lfn, iri, site,
regex=false)
- get_replica(lfn)
- has_replica(lfn)
- remove_replica(lfn)
- write_catalog(path)
...

Pegasus.DAX4.TransformationCatalog
- add_transformation(namespace, name,
version, os, arch, ...)
- get_transformation(namespace, name,
version, os, arch, ...)
- has_transformation(namespace, name,
version, os, arch, ...)
- remove_transformation(namespace,
name, version, os, arch, ...)
- write_catalog(path)
...

------- Workflow -------
Pegasus.DAX4.Workflow <------
requires only 'pegasus' (version),
'name', and 'jobs' (jobs and parent-
child dependencies)
- add_job(transformation, *args)
- add_dependency(parent, *children)
- add_site_catalog(sc)            #
add inline OR iri if string is given
- add_replica_catalog(rc)         #
add inline
- add_transformation_catalog(tc)  #
add inline
...
```

```
e_analyze = Executable
(namespace="diamond", name="
analyze", version="4.0",
os="linux", arch="x86_64")
e_analyze.addPFN(PFN
("gsiftp://site.com/bin
/analyze","site"))
diamond.addExecutable
(e_analyze)

# Add a preprocess job
preprocess = Job
(e_preprocess)
preprocess.addProfile
(Profile(Namespace.DAGMAN,
"RETRY", 0))
preprocess.metadata("time",
"60")
b1 = File("f.b1")
b2 = File("f.b2")
preprocess.addArguments("-a
preprocess","-T60","-i",a,"-
o",b1,b2)
preprocess.uses(a,
link=Link.INPUT)
preprocess.uses(b1,
link=Link.OUTPUT,
transfer=True)
preprocess.uses(b2,
link=Link.OUTPUT,
transfer=True)
diamond.addJob(preprocess)

# Add left Findrange job
frl = Job(e_findrange)
frl.metadata("time", "60")
c1 = File("f.c1")
frl.addArguments("-a
findrange","-T60","-i",b1,"-
o",c1)
frl.uses(b1, link=Link.
INPUT)
frl.uses(c1, link=Link.
OUTPUT, transfer=True)
diamond.addJob(frl)

# Add right Findrange job
frr = Job(e_findrange)
frr.metadata("time", "60")
c2 = File("f.c2")
frr.addArguments("-a
findrange","-T60","-i",b2,"-
o",c2)
frr.uses(b2, link=Link.
INPUT)
frr.uses(c2, link=Link.
OUTPUT, transfer=True)
diamond.addJob(frr)

# Add Analyze job
analyze = Job(e_analyze)
analyze.metadata("time",
"60")
d = File("f.d")
analyze.addArguments("-a
analyze","-T60","-i",c1,c2,"
-o",d)
analyze.uses(c1, link=Link.
INPUT)
analyze.uses(c2, link=Link.
```

```
(e_findrange)

e_analyze = Transformation
(namespace="diamond", name="
analyze", version="4.0", os="
linux", arch="x86_64")
e_analyze.add_PFN
("gsiftp://site.com/bin
/analyze","site")
diamond.add_transformation
(e_analyze)

# Add a preprocess job
preprocess = Job
(e_preprocess)
preprocess.add_profile
(Namespace.DAGMAN, "RETRY",
0)
preprocess.add_metadata
("time", "60")
b1 = File("f.b1")
b2 = File("f.b2")
preprocess.add_arguments("-a
preprocess","-T60","-i",a,"-
o",b1,b2)
preprocess.add_inputs(a)
preprocess.add_outputs(b1,
b2, stage_out=True,
register_replica=False)
diamond.add_job(preprocess)

# Add left Findrange job
frl = Job(e_findrange)
frl.add_metadata("time",
"60")
c1 = File("f.c1")
frl.add_arguments("-a
findrange","-T60","-i",b1,"-
o",c1)
frl.add_inputs(b1)
frl.add_outputs(c1,
stage_out=True)
diamond.add_job(frl)

# Add right Findrange job
frr = Job(e_findrange)
frr.add_metadata("time",
"60")
c2 = File("f.c2")
frr.add_arguments("-a
findrange","-T60","-i",b2,"-
o",c2)
frr.add_inputs(b2)
frr.add_outputs(c3,
stage_out=True)
diamond.add_job(frr)

# Add Analyze job
analyze = Job(e_analyze)
analyze.add_metadata("time",
"60")
d = File("f.d")
analyze.add_arguments("-a
analyze","-T60","-i",c1,c2,"-
o",d)
analyze.add_inputs(c1, c2)
analyze.add_outputs(d,
stage_out=True,
register_replica=True)
diamond.add_job(analyze)
```

```
'''


#!/usr/bin/env python

from Pegasus.DAX4 import *

# Create a DAX
diamond = Workflow("diamond")

# Add some metadata
diamond.add_metadata("name",
"diamond")\
    .add_metadata("createdby",
"Gideon Juve")

# Add input file to the
replica_catalog
rc = ReplicaCatalog()
a = rc.add_file("f.a", "gsiftp://site.
com/inputs/f.a", "site", regex=False)\
        .add_metadata("size", "1024")

# Add executables to the
transformation catalog
tc = TransformationCatalog()
e_preprocess = tc.add_transformation
(namespace="diamond", name="
preprocess", version="4.0", os="
linux", arch="x86_64")\
                .add_metadata("size",
"2048")\
                .add_site(site="
site", uri="gsiftp://site.com/bin
/preprocess", type="installed")

e_findrange = tc.add_transformation
(namespace="diamond", name="
findrange", version="4.0", os="
linux", arch="x86_64")\
                .add_site(site="
site", uri="gsiftp://site.com/bin
/findrange", type="installed")

e_analyze = tc.add_transformation
(namespace="diamond", name="analyze",
version="4.0", os="linux", arch="
x86_64")\
                .add_site(site="
site", uri="gsiftp://site.com/bin
/analyze", type="installed")

# Add a preprocess job
b1 = File("f.b1")
b2 = File("f.b2")
preprocess = diamond.add_job
(e_preprocess, "-a preprocess", "-
T60", "-i", a, "-o", b1, b2)\ # ADAG.
add_job(exec, *args)
                .add_profile
(Namespace.DAGMAN, "RETRY", 0)\
                .add_metadata("time",
"60")\
                .add_inputs(a)\
                .add_outputs(b1, b2,
stage_out=True,
register_replica=False) # job.
add_outputs(*lfns, stage_out=True,
register_replica=False)
```

```
INPUT)
analyze.uses(d, link=Link.
OUTPUT, transfer=True,
register=True)
diamond.addJob(analyze)

# Add dependencies
diamond.depends
(parent=preprocess,
child=frl)
diamond.depends
(parent=preprocess,
child=frr)
diamond.depends(parent=frl,
child=analyze)
diamond.depends(parent=frr,
child=analyze)

# Write the DAX to stdout
import sys
diamond.writeXML(sys.stdout)

# Write the DAX to a file
f = open("diamond.dax","w")
diamond.writeXML(f)
f.close()
```

```
# Add dependencies
diamond.add_dependency
(preprocess, frl, frr) #
Workflow.add_dependency
(parent, *children)
diamond.add_dependency(frl,
analyze) # Workflow.
add_dependency(parent,
*children)
diamond.add_dependency(frr,
analyze) # Workflow.
add_dependency(parent,
*children)

# Write the DAX to stdout
import sys
diamond.write_yaml(sys.
stdout)

# Write the DAX to a file
f = open("diamond.yml","w")
diamond.write_yaml(f)
f.close()
```

```
# Add left Findrange job
c1 = File("f.c1")
frl = diamond.add_job(e_findrange, "-
a findrange", "-T60", "-i", b1, "-o",
c1)\ # ADAG.add_job(exec, *args)
        .add_metadata("time", "60")\
        .add_inputs(b1)\
        .add_outputs(c1,
stage_out=True)

# Add right Findrange job
c2 = File("f.c2")
frr = diamond.add_job(e_findrange, "-
a findrange", "-T60", "-i", b2, "-o",
c2)\ # ADAG.add_job(exec, *args)
        .add_metadata("time", "60")\
        .add_inputs(b2)\
        .add_outputs(c2,
stage_out=True)\

# Add Analyze job
d = File("f.d")
analyze = diamond.add_job(e_analyze, *
["-a analyze", "-T60", "-i", c1, c2,
"-o", d])\ # ADAG.add_job(exec, *args)
            .add_metadata("time",
"60")\
            .add_inputs(c1, c2)\
            .add_outputs(d,
stage_out=True, register_replica=True)

# Add dependencies
diamond.add_dependency(preprocess,
frl, frr)\
    .add_dependency(frl, analyze)\
    .add_dependency(frr, analyze)


If you wanted the catalogs inline,
then:
diamond.add_replica_catalog(rc)
diamond.add_transformation_catalog(tc)
diamond.add_site_catalog(sc) # not
defined above, but if it was


# Write the DAX to a file
f = open("diamond.yml","w")
diamond.write_yaml(f)
f.close()

'''
# Additionally, we could use
contextlib and rewrite like the
following:
with ReplicaCatalog("rc.yml") as rc, \
     SiteCatalog("sc.yml") as sc, \
     TransformationCatalog("tc.yml")
as tc, \
     Workflow("diamond") as wf:

     # do stuff

# with block ends and all the
catalogs + workflow are written to
separate files
'''
```

DAX3 Invokes

```
from Pegasus.DAX3 import *

# Create a DAX
adag = ADAG("workflow")

# Add invoke
adag.addInvoke(Invoke("start", "/usr/bin/echo
hello"))
# adag.invoke("start", "/usr/bin/echo hello")
```

DAX4 Hooks

```
from Pegasus.DAX4 import *

# Create a DAX
adag = Workflow("workflow")

# Add shell hook
adag.add_shell_hook("start", "/usr/bin/echo
hello")

# Add web hook hook
adag.add_webhook_hook("start", ...)
```

DAX3 Transformations

```
from Pegasus.DAX3 import *

# Create a DAX
adag = ADAG("workflow")

# Create some executables
e1 = Executable("executable_1")
e2 = Executable("executable_2")
e3 = Executable("executable_3")

# Create a transformation
t = Transformation(e3)
t.uses(e1)
t.uses(e2)
t.uses(e3)
```

DAX4 Transformations

```
from Pegasus.DAX4 import *

# Create a DAX
adag = Workflow("workflow")

# Create some transformations
e1 = Transformation("executable_1")
e2 = Transformation("executable_2")


# Create a transformation that requires other transformations
t = Transformation("executable_3")
t.requires(e1, e2)
```

Jobs

```
'''
Trying to be declarative, but then workflow.add(...) might need to be
workflow.jobs(Job(...)) to be consistent. Don't like how the function names
aren't verbs.
'''
job = workflow.add(Job(transformation, arguments))\
    .profiles(namespace, "key", "value")\
    .metadata({"time": "60"})\
    .metadata({"key": "value"})\
    .hooks(ShellHook(args))\
    .inputs(a)\
    .outputs(b1, b2, stage_out=True)

'''
Verbose, explicitly stating operations on job.
The following two assignments are the same.
'''
job = workflow.add_job(transformation, arguments...)\
    .add_profile(namespace, "key", "value")\
    .add_metadata(**{"time": "60", "key": "value"})\
    .add_shell_hook(args)\
    .add_inputs(a)\
    .add_outputs(b1, b2, stage_out=True)

job = workflow.add_job(Job(transformation, args))\
    .add_profile(Profile(namespace, "key", "value"))\
    .add_metadata(*[Metadata("time", "60"), Metadata("key", "value")])\
    .add_shell_hook(args)\
    .add_inputs(a)\
    .add_outputs(b1, b2, stage_out=True)

'''
List/Set semantics.Can't chain as job.<member>.add(something) won't
return back a ref to the job (unless we make it but that would look awkward)
'''
job = workflow.add(Job(transformation, arguments))
# List semantics for (job.args)
job.arguments.append("A")
job.arguments.extend( ["A", "B"] )
# Set semantics for (job.profiles)
job.profiles.add(namespace, "key", "value")
# Dict semantics for job.{<namespace>,metadata}.*
job.env.key = "value"
job.env["key"] = "value"
job.env.update( {"key1": "value1", "key2": "value2"} )
job.metadata.add({"time": "60"})
job.metadata.add({"key": "value"})
job.hooks.add(ShellHook(args))
job.inputs.add(a)
job.outputs.add(b1, b2, stage_out=True)
```

DAX4 Workflow "generic add"

```
#!/usr/bin/env python
from Pegasus.DAX4 import *

# Create a DAX
diamond = Workflow("diamond")

# Add some metadata
diamond.add({"name": "diamond"})

# Add input file to the replica_catalog
```

DAX4 Workflow "adds"

```
#!/usr/bin/env python
from Pegasus.DAX4 import *

# Create a DAX
diamond = Workflow("diamond")

# Add some metadata
diamond.add_metadata("name", "diamond")

# Add input file to the replica_catalog
rc = ReplicaCatalog()
sc = SiteCatalog()
# say that we've added some replicas and sites to
the catalogs

tc = TransformationCatalog()
e_preprocess = tc.add_transformation(name="
preprocess")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/preprocess", type="
installed")

e_findrange = tc.add_transformation(name="
findrange")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/findrange", type="installed")

e_analyze = tc.add_transformation(name="analyze")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/analyze", type="installed")
# Add jobs
job1 = diamond.add_job(e_preprocess, args)
job2 = diamond.add_job(e_findrange, args)
job3 = diamond.add_job(e_findrange, args)
job4 = diamond.add_job(e_analyze, args)

# Add dependencies
diamond.add_dependency(job1, job2, job3)\
    .add_dependency(job2, job4)\
    .add_dependency(job3, job4)

diamond.add_replica_catalog(rc)
diamond.add_transformation_catalog(tc)
diamond.add_site_catalog(sc)

# Write the DAX to a file
f = open("diamond.yml","w")
diamond.write_yaml(f)
f.close()
```

```
rc = ReplicaCatalog()
sc = SiteCatalog()
# say that we've added some replicas and sites to
the catalogs

tc = TransformationCatalog()
e_preprocess = tc.add_transformation(name="
preprocess")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/preprocess", type="
installed")

e_findrange = tc.add_transformation(name="
findrange")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/findrange", type="installed")

e_analyze = tc.add_transformation(name="analyze")\
                .add_site(site="site", uri="
gsiftp://site.com/bin/analyze", type="installed")
# Add jobs
job1 = diamond.add(e_preprocess, args)
job2 = diamond.add(e_findrange, args)
job3 = diamond.add(e_findrange, args)
job4 = diamond.add(e_analyze, args)

# Add dependencies
diamond.add(job1, job2, job3)
diamond.add(job2, job4)
diamond.add(job3, job4)

diamond.add(rc)
diamond.add(tc)
diamond.add(sc)

# Write the DAX to a file
f = open("diamond.yml","w")
diamond.write_yaml(f)
f.close()

'''
Using Workflow.add(*args) as supposed to:
- Workflow.add_job(transformation, *args)
- Workflow.add_dependency(parent, *children)
- Workflow.add_metadata(dict or key, value)
- Workflow.add_transformation_catalog(tc)
- Workflow.add_site_catalog(sc)
- Workflow.add_replica_catalog(rc)

Could work as functools.singledispatch makes the
dispatch happen
based only on the first argument, and in our
function signatures above,
the first arguments are all unique.

If we go with this route, then chaining things
together may not be ideal
as it is not obvious what Workflow.add(...) will
return. When we say
Workflow.add_job(transformation, *args), it might
be more apparent that
a Job would be returned, and therefore you can
chain job related functions
together.
'''
```