HWRF Database

Overview

- What is it and Why is it?
- HWRF SQLite3 Database
  - produtil.datastore
  - hwrf.hwrftask.HWRFTask
- hwrf_expt

What is it and Why is it?

- Old (ksh-based) HWRF scripts had massive filesystem overheads due to stat and ls calls.
  - Check to see if wrf output is present.
  - Check to see what the filename is.
  - Check to see if post completed.
- Millions of stat and opendir/readdir calls per cycle.
  - Stretching limits of filesystem metadata capacity.

What is it and Why is it?

- Solution: when a file becomes available, script puts location and other info in a database.
  - location = /path/to/file
  - available = flag
  - other metadata
    - GRIB grid center location
    - path to GRIB index files
    - expected file size

HWRF SQLite3 Database

SQLite3 = Database Format and Library

Table “products”

id	available	location	type
geogrid::geo_nmm_nest	0	/path/to/file	Product

Table “metadata”

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

- NOTE: Actual HWRF has hundreds of entries in those tables.

Product

Example: Make a new product.

Python Code (produtil.datastore)

```
prod=Product(ds,"geo_nmm_nest","geogrid",
  location="/path/to/file",available=False,
  type="Product")
```

SQL code HWRF writes for you.

```
INSERT OR IGNORE INTO products VALUES
("geogrid::geo_nmm_nest",
0,"/path/to/file","Product")
```

Table "products"

id	available	location	type
geogrid::geo_nmm_nest	0	/path/to/file	Product

Table "metadata"

id	key	value
-empty-	-empty-	-empty-

Product

Example: Change location and availability.

Python Code (produtil.datastore)

```
prod.set_loc_avail("/new/location",True)
# or: prod.location="/new/location"
#     prod.available=True  (two operations)
```

SQL code HWRF writes for you.

```
INSERT OR REPLACE INTO products VALUES
("geogrid::geo_nmm_nest",
1,"/new/location","Product")
```

Table “products”

id	available	location	type
geogrid::geo_nmm_nest	1	/new/location	Product

Table “metadata”

id	key	value
-empty-	-empty-	-empty-

Product
Example: Add metadata

Python Code (produtil.datastore)

```
prod[“minsize”]=100000000
print repr(prod[“minsize”])  # prints “100000000”
# Value is always converted to a string!
```

SQL code HWRF writes for you.

```
INSERT OR REPLACE INTO metadata VALUES
(“geogrid::geo_nmm_nest”, ”minsize”,”100000000”)
```

Table “products”

id	available	location	type
geogrid::geo_nmm_nest	1	/new/location	Product

Table “metadata”

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

Product

Example: Query location & availability

```
prod=Product(ds,"geo_nmm_nest","geogrid",
location="/path/to/file",available=False,type="Product")
print repr(prod.location), repr(prod.available)
# Prints "/new/location" True   WHY?
```

SQL code HWRF writes for you.

```
INSERT OR IGNORE INTO products VALUES
("geogrid::geo_nmm_nest",
0,"/path/to/file","Product")
```

Table "products"

id	available	location	type
geogrid::geo_nmm_nest	1	/new/location	Product

Table "metadata"

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

Product

Example: Query location & availability

```
prod=Product(ds,"geo_nmm_nest","geogrid",
location="/path/to/file",available=False,type="Product")
print repr(prod.location), repr(prod.available)
# Prints "/new/location" True
```

SQL code HWRF writes for you.

```
INSERT OR IGNORE INTO products VALUES
("geogrid::geo_nmm_nest",
0,"/path/to/file","Product")
```

Table "products"

id	available	location
geogrid::geo_nmm_nest	1	/new/location

Table "metadata"

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

WHY?
Constructor only sets **DEFAULT** values. This allows you to use the same constructor when querying the database as when creating it.

FileProduct
Delivery of Files

Python Code (produtil.datastore)

```
prod=FileProduct(ds,"geo_nmm_nest","geogrid",
  location="/path/to/file",available=False,
  type="Product")
```

SQL code HWRF writes for you.

```
INSERT OR IGNORE INTO products VALUES
("geogrid::geo_nmm_nest",
0,"/path/to/file","Product")
```

Table "products"

id	available	location	type
geogrid::geo_nmm_nest	0	/path/to/file	Product

Table "metadata"

id	key	value
-empty-	-empty-	-empty-

FileProduct
Delivery of Files

- `prod.deliver(location,frominfo,keep=True,`
  `logger=None,copier=None)`
  - Deliver file from frominfo to location, set available to True.
- `prod.undeliver(delete=True,logger=None)`
  - Set available=False, delete file if delete=True

UpstreamFile
Monitoring Files from External Workflow

- `prod.check(frominfo,minsize=None,maxage=None,`
  `logger=None`
  - Check frominfo for file of specified minimum size and age
  - Default: use prod's metadata “minsize,” “minage”
- `prod.undeliver()`
  - Set available=False, do NOT delete
- `prod.deliver()`
  - Raises exception. You cannot deliver a file from an external workflow.

Task

Example: New Task

Python Code (produtil.datastore)

```
task=Task(ds,"geogrid")
```

SQL code HWRF writes for you.

```
INSERT OR IGNORE INTO products VALUES
("**task**::geogrid", 0,"","Task")
```

Table "products"

id	available	location	type
task::geogrid	0		Task
geogrid::geo_nmm_nest	1	/new/location	Product

Table "metadata"

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

Task

Example: Task State

```
task=Task(ds,"geogrid")
task.state=RUNNING      # = 10
... do things ...
task.state=COMPLETED    # = 30
```

SQL code HWRF writes for you.

```
UPDATE OR IGNORE products SET available=10

UPDATE OR IGNORE products SET available=30
```

Table "products"

id	available	location	type
task::geogrid	30		Task
geogrid::geo_nmm_nest	1	/new/location	Product

Table "metadata"

id	key	value
geogrid::geo_nmm_nest	minsize	100000000

Task

Example: RUNNING/COMPLETED/FAILED

```
task=Task(ds,”geogrid”)
try:
    task.state=RUNNING      # = 10
    ... do things ...
    task.state=COMPLETED    # = 30
except Exception as e: # don't catch BaseException
    task.state=FAILED       # = -10
    raise
```

Task
Example: RUNNING/COMPLETED/FAILED

```
task=Task(ds,”geogrid”)
try:
    task.state=RUNNING      # = 10
    ... do things ...
    task.state=COMPLETED    # = 30
except Exception as e: # don't catch BaseException
    task.state=FAILED       # = -10
    raise
```

- Problems:
  - What Products does the task create?
  - How does the script level run this task?
  - How does the task log any errors?
  - How do we reconfigure the task?
- Subclass HWRFTask!!

SimpleGeogrid
Why HWRFTask?

```
class SimpleGeogrid(HWRFTask):
  def __init__(self,*args,**kwargs):
    super(SimpleGeogrid,self).__init__(*args,**kwargs)
    self.__geogrid_product=FileProduct(
      self.dstore,"geo_nmm_nest",self.taskname,
      location=os.path.join(self.outdir,"geo_nmm_nest")
  def products(self,**kwargs):
    yield self.__geogrid_product
  def run(self):
    try:
      task.state=RUNNING      # = 10
      ... do things ...
      self.__geogrid_product.deliver(...)
      task.state=COMPLETED    # = 30
    except Exception as e: # don't catch BaseException
      self.log().error("cry: %s"%(str(e),),
          exc_info=True)
      task.state=FAILED       # = -10
      raise
```

SimpleGeogrid
HWRFTask Properties

- task.outdir - output directory
- task.workdir - scrub area
- task.taskname - task name in database
  - should be used as “category” for task's products
- task.section - section in config file
- task.dstore - produtil.datastore.Datastore for the database
- task.did - database identifier

SimpleGeogrid
Member Functions

```
class SimpleGeogrid(HWRFTask):
  def __init__(self,*args,**kwargs):
    super(SimpleGeogrid,self).__init__(*args,**kwargs)
    self.__geogrid_product=FileProduct(
      self.dstore,"geo_nmm_nest",self.taskname,
      location=os.path.join(self.outdir,"geo_nmm_nest")
  def products(self,**kwargs):
    yield self.__geogrid_product
  def run(self):
    try:
      task.state=RUNNING      # = 10
      ... do things ...
      self.__geogrid_product.deliver(...)
      task.state=COMPLETED    # = 30
    except Exception as e: # don't catch BaseException
      self.log().error("cry: %s"%(str(e),),
          exc_info=True)
      task.state=FAILED       # = -10
      raise
```

SimpleGeogrid
Products

```
class SimpleGeogrid(HWRFTask):
  def __init__(self,*args,**kwargs):
    super(SimpleGeogrid,self).__init__(*args,**kwargs)
    self.__geogrid_product=FileProduct(
      self.dstore,"geo_nmm_nest",self.taskname,
      location=os.path.join(self.outdir,"geo_nmm_nest")
  def products(self,**kwargs):
    yield self.__geogrid_product
  def run(self):
    try:
      task.state=RUNNING      # = 10
      ... do things ...
      self.__geogrid_product.deliver(...)
      task.state=COMPLETED    # = 30
    except Exception as e: # don't catch BaseException
      self.log().error("cry: %s"%(str(e),),
          exc_info=True)
      task.state=FAILED       # = -10
      raise
```

SimpleGeogrid
Exception Handling

```
class SimpleGeogrid(HWRFTask):
  def __init__(self,*args,**kwargs):
    super(SimpleGeogrid,self).__init__(*args,**kwargs)
    self.__geogrid_product=FileProduct(
      self.dstore,"geo_nmm_nest",self.taskname,
      location=os.path.join(self.outdir,"geo_nmm_nest")
  def products(self,**kwargs):
    yield self.__geogrid_product
  def run(self):
    try:
      task.state=RUNNING      # = 10
      ... do things ...
      self.__geogrid_product.deliver(...)
      task.state=COMPLETED    # = 30
    except Exception as e: # don't catch BaseException
      self.log().error("cry: %s"%(str(e),),
          exc_info=True)
      task.state=FAILED       # = -10
      raise
```

WRF Inputs/Outputs
WRF Tasks

- All inputs/outputs are Products
  - Forecast outputs are UpstreamFile
  - Most others are FileProducts
- All WRF Tasks derive from WRFTaskBase
  - RealNMM - runs real_nmm
  - WRFAnl - generates analysis time restart files
  - WRFGhost - same, but also history stream
  - WRFAnl4Trak - 1 minute history files, tweaked to lie that it is for analysis time.
  - WRFAtmos - uncoupled forecast job
  - ...others...

WRF Inputs/Outputs
Example

- NOTE: For a long example:
  - examples/simple_hwrf/simple.py

```
real=RealNMM(ds,conf,'wrfexe',wrf,
    taskname='real')
anl=WRFAnl(ds,conf,'wrfexe',wrf,
    taskname='wrfanl')
anl.add_real(real)
```

  - **What does add_real() do!?**

WRF Inputs/Outputs
add_real()

```
class WRFTaskBase(FcstTask):
   def add_real(self,r):
      return self.add_fort65(r)\
                  .add_wrfinput(r)...
   def add_wrfinput(self,r)
      return self.add_input('wrfinput',
         WRFInput2WRF(r))
class WRFInput2WRF(Input2Fcst):
   def get_inputs(...):
      p=self.src.wrfinput_at_time(atime,domain)
      self.link_product(p,...)
```

WRF Inputs/Outputs
wrfinput_at_time()

```
class RealNMM(WRFTaskBase):
    def wrfinput_at_time(self,atime,domain):
        if(right atime and domain):
            return self.prod_wrfinput
        else: # wrong atime or wrong domain
            return None
    def make_products(...):
        self.prod_wrfinput=FileProduct(
            self.dstore,'wrfinput_d01',...)
```

WRF Inputs/Outputs
Backup Inputs

```
# Primary data source: GDAS merge:
realwrf.add_wrfinput(gdas_merge)

# Backup data source: GFS relocation stage 3
# Used if gdas_merge.get_wrfinput returns None
realwrf.add_wrfinput(gfsinit.rstage3)

class RelocationTask(...):
    def wrfinput_at_time(self,atime,domain):
        if(right atime and domain):
            return self.get_wrfinput(domain)
    def get_wrfinput(self,domain):
        pass # (Defined in subclasses)
```

WRF Inputs/Outputs
More Info

- Many different inputs/outputs
  - See hwrf.fcsttask.WRFTaskBase for list
- GSI and relocation use similar pattern.
  - hwrf.gsi and hwrf.relocate modules