OS Release Procedure

1. Run the **nightlyBuild.py** script.

2. Test the examples. They are in **OS/examples**. Do a **make install** before running these.

   a. Connect to the **algorithmicDiff** folder, build and run **OSAlgorithmicDiffTest.cpp**. This takes no arguments. This will test a bunch of the AD routines.

   b. Connect to the **instanceGenerator** folder, build and run **OSInstanceGenerator.cpp**. This takes no arguments.

   c. Connect to the **osTestCode** folder, build and run **OSTestCode.cpp**. This takes a single argument which is the location of any OSiL file.

3. Test the applications. They are in **OS/applications**.

   a. Test **OSAmplClient**. This is not a stand-alone application and is designed to be called from **ampl**. Probably the easiest way to test this is to test the **OSAmplClient** that gets installed in the **bin** directory as a result of **make install**. To make life easy, temporarily copy your **ampl** executable into this **bin** directory. Also copy the test problem **hs71.nl** from **OS/data/amplFiles/** into the **bin** directory. Do five tests. Three local and two remote.

   Test 1: Inside **ampl** execute the following

   ```
   model hs71.mod;
   option solver OSAmplClient;
   option OSAmplClient_options "solver xyz";
   solve;
   ```

   The result should be an error saying:

   ```
   <message>a supported solver has not been selected</message>
   ```

   Test 2: Inside **ampl** execute the following

   ```
   model hs71.mod;
   option solver OSAmplClient;
   option OSAmplClient_options "solver ipopt";
   solve;
   display x1;
   ```

The result of **display x3** should be 3.82115.

Test 3: Inside **ampl** execute the following

```
model hs71.mod;
option solver OSAmplClient;
option OSAmplClient_options "solver cbc";
solve;
```

You should get an error message saying:

```
<message>Cbc cannot do nonlinear or quadratic</message>
```

Test 4: Inside **ampl** execute the following

```
model hs71.mod;
option solver OSAmplClient;
option OSAmplClient_options "solver ipopt";
option ipopt_options "service http://gsbkip.chicagogsb.edu/os/OSSolverService.jws";
solve;
display x1;
```

The result of **display x3** should be 3.82115.

Test 5: Inside **ampl** execute the following

```
model hs71.mod;
option solver OSAmplClient;
option OSAmplClient_options "solver clp";
option clp_options "service http://gsbkip.chicagogsb.edu/os/OSSolverService.jws";
solve;
display x3;
```

You should get an error message saying"

```
<message>Clp cannot do nonlinear or quadratic or integer</message>
```

There is command script, **testAmpl.run** in the directory **OS/data/amplFiles** that contains the commands for all of these test. Simply start **ampl** and execute

```
include testAmpl.run;
```

b. Test the **OSFileUpload** application. Edit **OSFileUpload.cpp**. First comment out line 79 and then modify line

```
osagent = new OSSolverAgent("http://******/os/servlet/OSFileUpload");
```

to

```
osagent = new OSSolverAgent("http://gsbkip.chicagogsb.edu/os/servlet/OSFileUpload");
```

Rebuild and run. This application takes one command line argument which is the file to be uploaded.

4. Test the **OSSolverService**.

   a. Test running a local solver. (These examples assume that the **OS/data** directory is one level above the directory in which **OSSolverService** is running. Test for OSiL, mps, and nl files.

   ```
   OSSolverService -config ../data/configFiles/testLocal.config
   OSSolverService -config ../data/configFiles/testLocalMPS.config
   OSSolverService -config ../data/configFiles/testLocalNL.config
   ```

   You should get the OSrL for the simple test problem. In all of these look for `<obj idx="-1">-7667.94</obj>` in the MPS test and `<obj idx="-1">-7667.94</obj>` in the other two.

   b. Test the service methods on the remote server.
   Step 1: Test remote **solve()** method for OSiL, mps, and nl files.

   ```
   OSSolverService -config ../data/configFiles/testRemote.config
   OSSolverService -config ../data/configFiles/testRemoteMPS.config
   OSSolverService -config ../data/configFiles/testRemoteNL.config
   ```

   You should get the OSrL for the simple test problem in each case. In all of these look for `<obj idx="-1">-7667.94</obj>`.

   Step 2: Test remote **getJobID()** method.

   ```
   OSSolverService  -config ../data/configFiles/testRemotegetJobID.config
   ```

   You will get a long jobID.

   Step 3: Test remote **send()** method. Use the **send()** method with the jobID just generated. To do this open the file

   `/data/osolFiles/sendWithJobID.osol`

   and replace the existing jobID with the one just generated. Then run

   ```
   OSSolverService  -config ../data/configFiles/testRemoteSend.config
   ```

   The result should be "send is true."

   Step 4: Test remote **knock()** method. See if the job is complete.

   ```
   OSSolverService  -config ../data/configFiles/testRemoteKnock.config
   ```

You do not need to put in jobID information. The knock will get the status of all jobs. However, if want just the status of the job you submitted put your jobID in the `knock.osol` file.

Step 5: Test remote **retrieve()** method. Get the result.

```
OSSolverService  -config ../data/configFiles/testRemoteRetrieve.config
```

Before executing this command make sure to put your jobID into the file **retrieve.osol** . Also, either delete the `-browser` option or put in the path to your browser. The result of the optimization will be put into a file called **test.osrl** that will be in the directory in which you are running the **OSSolverService.**

IMPORTANT: Please do NOT commit the changes to these config files.

5. Test **OSCommon**. Build the OSCommon library. Build the **OSCommon** library. Do a **make install**. Then connect to **apiExamples** directory, build and run the **apiExample.**