Using the COIN-OR Server

Your CoinEasy Team

November 16, 2009

1 Overview

This document is part of the **CoinEasy** project. See `projects.coin-or.org/CoinEasy`. In this document we describe the options available to users of COIN-OR who are interested in solving optimization problems but *do not* wish to compile source code in order to build the COIN-OR projects. In particular, we show how the user can send optimization problems to a COIN-OR server and get the solution result back. The COIN-OR server, `webdss.ise.ufl.edu`, is 2x Intel(R) Xeon(TM) CPU 3.06GHz 512MiB L2 1024MiB L3, 2GiB DRAM, 4x73GiB scsi disk 2xGigE machine. This server allows the user to directly access the following COIN-OR optimization solvers:

- **Bonmin** – a solver for mixed-integer nonlinear optimization
- **Cbc** – a solver for mixed-integer linear programs
- **Clp** – a linear programming solver
- **Couenne** – a solver for mixed-integer nonlinear optimization problems and is capable of global optiomization
- **DyLP** – a linear programming solver
- **Ipopt** – an interior point nonlinear optimization solver
- **SYMPHONY** – mixed integer linear solver that can be executed in either parallel (distributed or shared memory) or sequential modes
- **Vol** – a linear programming solver

All of these solvers on the COIN-OR server may be accessed through either the GAMS or AMPL modeling languages. In Section 2.1 we describe how to use the solvers using the GAMS modeling language. In Section 2.2 we describe how to call the solvers using the AMPL modeling language. In Section 3 we describe how to call the solvers using a command line executable program `OSSolverService.exe` (or `OSSolverService` for Linux/Mac OS X users – in the rest of the document we refer to this executable using a `.exe` extension). The `OSSolverService.exe` can be used independently of a modeling language. It can send optimization instances to the solvers in MPS, OSiL (a new XML based representation standard), AMPL nl, and GAMS dat formats. A nice feature of the `OSSolverService.exe` is that it can be used in asynchronous mode for large problems. This is also described in Section 3. We show how to obtain a job id from the server, send a job to the server, check the server to see if the job is done, retrieve the job if it is done, and kill the job if it is taking too long. In Section 4 we suggest an approach for users who wish to actually write code. Finally, we describe how to download the necessary client software in Section 5. This software consists of executable programs for various platforms, the user is not required to compile code.

2 Calling the COIN-OR Server using a Modeling Language

2.1 GAMS

In this section we explain how to use the GAMS modeling language to call the COIN-OR solver server. This section is based on the assumption that the user has installed either:

1. GAMS version 22.9, 23.0, 23.1, or 23.2. In the the distribution described in Section 5 there is a file `gmsos_.zip`. Copy this file into your GAMS folder. Then run `gamsinst` and select `OS` as the default solver whenever it is listed as a solver.

2. GAMS version 23.3. No additional software is required. However, with this version you cannot select `OS` as the default solver. Rather, when building the GAMS model, place the following option command somewhere in the model *before* the `Solve` statement (however, with 23.3 you could install `gmsos_.zip` if desired and then go through the install process and select `OS` as the default solver):

   ```
   Option ProblemType = CoinOS;
   ```

   where `ProblemType` can be

   ```
   LP, RMIP, MIP, DNLP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP
   ```

   As an alternative to putting the solver option in the GAMS model, it can be given at the command line. When running the model and solving model, write:

   ```
   gams.exe  ModelName ProblemType=CoinOS
   ```

You can now solve a wide variety of problems either locally or remotely using COIN-OR solvers. In the discussion that follows we assume that folder where GAMS is installed is in the PATH command. We first discuss using GAMS to invoke the COIN-OR solvers that are distributed with GAMS. In this case the model generation and resulting optimization are all done on the local machine. Then we discuss using GAMS to call the remote COIN-OR solver service.

2.1.1 Using GAMS for a Local Solver

The `CoinOS` (similarly `OS`) solver that is packaged with GAMS is really a *solver interface* and is linked to the following COIN-OR solvers: `Bonmin`, `Cbc`, `Clp`, `Couenne`, `Glpk`, and `Ipopt` . Think of `CoinOS` as a *meta slover.* The distribution described in Section 5 contains a test problem, `eastborne.gms`, which is an integer linear program. To solve this problem for versions 22.9-23.2 type

```
gams.exe eastborne.gms
```

and for version 23.3 type

```
gams.exe eastborne.gms MIP=CoinOS
```

Executing the commands above will result in the model being solved on the local machine using the COIN-OR solver `Cbc`. By default, for continuous linear models (LP and RMIP), `CoinOS` or `OS` chooses Clp. For continuous nonlinear models (NLP, DNLP, RMINLP, QCP, RMIQCP), Ipopt is the default solver. For mixed-integer linear models (MIP), Cbc is the default solver. For mixed-integer nonlinear models (MIQCP, MINLP), Bonmin is the default solver.

It is possible to control which solver is selected by `CoinOS` or `OS`. This is done by providing an *options file* to `CoinOS` or `OS`. For versions 22.9-23.2 the option fille should be named `os.opt` and this file is invoked at the command line by

```
gams.exe eastborne.gms optfile 1
```

It is possible to define multiple GAMS options files following the specific naming conventions as set out below:

```
optfile=1 corresponds to os.opt
optfile=2 corresponds to os.op2
...
optfile=99 corresponds to os.o99
```

For GAMS 23.3, since the solver is named `CoinOS`, the option file should be named `coinos.opt` and the command line call is

```
gams.exe eastborne.gms MIP=CoinOS optfile 1
```

and calling multiple GAMS options files uses the convention

```
optfile=1 corresponds to coinos.opt
optfile=2 corresponds to coinos.op2
...
optfile=99 corresponds to coinos.o99
```

We now explain the valid options that can go into the options files. The first option is the **solver** option.

solver (string): Specifes the solver that is used to solve an instance. Valid values are `clp`, `cbc`, `glpk`, `ipopt`, `bonmin`, and `couenne`. If a solver name is specified that is not recognized, the default solver for the problem type will be used. For example, if the file `coinos.opt` contains a single line

```
solver coinbonmin
```

then the `Bonmin` solver will be used to solve the problem. Following are some of the other options that can be specified in a GAMS option file when using either `OS` or `CoinOS`.

writeosil (string): If this option is used GAMS will write the optimization instance to file in OSiL (Optimization Services instance Language) XML format.

writeosrl (string): If this option is used GAMS will write the result of the optimization to file in OSrL (Optimization Services result Language) XML format.

The option just described are options for the GAMS modeling language. It is also possible to pass options directly to the COIN-OR solvers used by either `CoinOS` or `OS`. This is done by passing the name of an option file that conforms to the OSoL (Optimization Services option Language) XML standard. See `http://projects.coin-or.org/OS` for information on Optimization Services. The option

readosol (string): Specifies the name of an option fille in OSoL format that is given to the solver.

The file **solveroptions.osol** is contained in the distribution described in Section 5. This file contains four solver options; two for Cbc, one for Ipopt, and one for SYMPHONY (which is available for remote server calls, but not locally). You can have any number of options. Note the format for specifying an option:

```
<solverOption name="maxN" solver="cbc" value="5" />
```

The attribute `name` specifies that the option name is `maxN` which is the maximum number of nodes allowed in the branch-and-bound tree, the `solver` attribute specifies the name of the solver that the option should be applied to, and the `value` attribute specifies the value of the option. As a second example, consider the specification

```
<solverOption name="max_iter" solver="ipopt" type="integer" value="2000"/>
```

In this example we are specifying an iteration limit for `Ipopt`. Note the additional attribute `type` that has value `integer`. The Ipopt solver requires specifying the data type (string, integer, or numeric) for its options. For a list of options that solvers take, inside the GAMS directory see the file

```
docs/solvers/coin.pdf
```

An up-to-date online version is available at `http://www.coin-or.org/GAMSlinks/gamscoin.pdf`. **IMPORTANT:** unlike the `OSAmplClient` described in Section 2.2, `OS` and `CoinOS` do not parse the **solveroptions.osol** file, they simply pass the file directly onto the local solver or the remote solver service.

2.1.2 Using GAMS to Invoke the COIN-OR Solver Service

We now describe how to call the COIN-OR solver service. There are several reason why you might wish to use this service.

- Have access to a faster machine.
- Be able to submit jobs to run in asynchronous mode – submit your job, turn off your laptop, and check later to see if the job ran.
- Call several additional solvers (`SYMPHONY` and `DyLP`).

In order to use the COIN-OR solver service it is necessary to specify the service URL. This is done using the **service** option.

service (string): Specifes the URL of the COIN-OR solver service

Use the following value for this option.

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
```

IMPORTANT: Using the **solver** option will have *no effect* on which solver the remote solver service uses. The remote solver service uses the same defaults as the local service. For linear models (LP and RMIP), Clp is the default solver. For continuous nonlinear models (NLP, DNLP, RMINLP, QCP, RMIQCP), Ipopt is the default solver. For mixed-integer linear models (MIP), Cbc is the default solver. For mixed-integer nonlinear models (MIQCP, MINLP), Bonmin is the default solver. In order to control the solver used, it is necessary to specify the name of the solver inside the XML tag `<solverToInvoke>`. The example `solveroptions.osol` file contains the XML tag

```
<solverToInvoke>symphony</solverToInvoke>
```

If the `coinos.opt` file is

```
solver coinipopt
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
readosol  solveroptions.osol
writeosrl temp.osrl
```

then `Ipopt` is ignored as a solver option and the remote server uses the `SYMPHONY` solver. Valid values for the remote solver service specified in the `<solverToInvoke>` tag are `clp`, `cbc`, `dylp`, `glpk`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`.

By default, the call to the server is a `synchronous` call. The GAMS process will wait for the result and then display the result. This may not be desirable when solving large optimization models. The user may wish to submit a job, turn off his or her computer, and then check at a later date to see if the job is complete. In order use the remote solver service in this fashion, i.e. `asynchronously` it is necessary to use the `service_method` option.

service_method (string) Specifies the method to execute on a server. Valid values for this option are `solve`, `getJobID`, `send`, `knock`, `retrieve`, and `kill`. We explain how to use each of these.

The default value of **service_method** is `solve.` A `solve` invokes the the remote service in synchronous mode. When using the `solve` you can optionally specify a set of solver options in an OSoL file by using the `readosol` option. The remaining values for the **service_method** option are used for an asynchronous call. We illustrate them in the order in which they would most logically be executed.

service_method getJobID: When working in asynchronous mode, the server needs to uniquely identify each job. The `getJobID` service method will result in the server returning a unique job id. For example if the following `coinos.opt` file is used

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method getJobID
```

with the command

```
gams.exe eastborne.gms optfile=1
```

the user will a rather long job id returned to the screen as output. Any job id can be sent to the server as long as it has not been used before. Assume that the job id returned is `coinor12345xyz`. This job id is used to submit a job to the server with the **send** method.

service_method send: When working in asynchronous mode, use the `send` service method to submit a job. When using the `send` service method option an option is required and the option file must specify a job id that has not been used before. Assume that in the `coinos.opt` we specify the options:

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method send
readosol sendWithJobID.osol
```

The `sendWithJobID.osol` option file is identical to the `solveroptions.osol` option file except that it has an additional XML tag:

```
<jobID>coinor12345xyz</jobID>
```

We then execute

```
gams.exe eastborne.gms optfile=1
```

If all goes well, the response to the above command should be: "Problem instance successfully send to OS service." At this point the server will schedule the job and work on it. It is possible to turn off the user computer at this point. At some point the user will want to know if the job is finished. This is accomplished using the **knock** service method.

service_method knock: When working in asynchronous mode, this is used to check the status of job. Consider the following `coinos.opt` file:

```
service http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
service_method knock
readosol sendWithJobID.osol
readospl knock.ospl
writeospl knockResult.ospl
```

The **knock** service method requires two inputs. The first input is the name of an option file, in this case `sendWithJobID` specified through the **readosol** option. In addition, a file in OSpL (Optimization Services process Language) is required. Do not worry about the format of this file, use the `knock.opsl` file provided in the binary distribution. This file name is specified using the **readospl** option. If no job id is specified in OSoL file then the status of all jobs on the server will be returned in the file specified by the **writeospl** option. If a job id is specified in the OSoL file, then only information on the specified job id is returned in the file specified by the **writeospl** option. In this case the file name is `knockResult.ospl`. We then execute

```
gams.exe eastborne.gms optfile=1
```

The file **knockResult.ospl** will contain the information

```
<job jobID="coinor12345xyz">
<state>finished</state>
<serviceURI>http://192.168.0.219:8443/os/OSSolverService.jws</serviceURI>
<submitTime>2009-11-10T02:13:11.245-06:00</submitTime>
<startTime>2009-11-10T02:13:11.245-06:00</startTime>
<endTime>2009-11-10T02:13:12.605-06:00</endTime>
<duration>1.36</duration>
 </job>
```

Note the the job is complete as indicated in the `<state>` tag. It is now time to actually retrieve the job solution. This is done with the **retrieve** option.

service_method retrieve: When working in asynchronous mode, this is used to retrieve the job solution. It is necessary when using **knock** to specify an option file and in that option file specify a job id. Consider the following `coinos.opt` file:

```
service http://kipp.chicagobooth.edu/os/OSSolverService.jws
service_method retrieve
readosol sendWithJobID.osol
writeosrl answer.osrl
```

We then execute

```
gams.exe eastborne.gms optfile=1
```

and the result is written to the file answer.osrl.

Finally there is a **kill** service method which is used to kill a job that is a mistake or running too long on the server.

service_method kill: When working in asynchronous mode, this is used to terminate a job. You should specify an OSoL file containing the JobID by using the readosol option.

2.1.3 GAMS Summary:

GAMS SUMMARY:

1. If you are using GAMS 22.9, or 23.0, or 23.1, or 23.2, put **gmsos_.zip** in the GAMS directory, execute `gamsinst` and select `OS` as the default solver whenever it is listed as a solver. If you are using GAMS 23.3 place the statement `Option ProblemType = CoinOS;` somewhere in the model *before* the `Solve` statement.

2. If no options are given, then the model will be solved locally and Clp will be used for linear programs, Cbc for integer linear programs, Ipopt for continuous nonlinear programs, and Bonmin for nonlinear integer.

3. In order to control behavior (for example, whether a local or remote solver is used) an option file, **os.opt** or **coinos.opt**, must be used as follows

   ```
   gams.exe  eastborne.gms optfile=1
   ```

4. The **os.opt** or **coinos.opt** file is used to specify *eight potential options*:

   - **service (string):** using the COIN-OR solver server, this is done by giving the option

     ```
     service  http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
     ```

   - **readosol (string):** whether or not to send the solver an option file, this is done by giving the option

     ```
     readosol  solveroptions.osol
     ```

   - **solver (string):** if a local solve is being done, a specific solver is specified by the option

     ```
     solver solver_name
     ```

     Valid values are `clp`, `cbc`, `glpk`, `ipopt`, `bonmin`, and `couenne`. When the COIN-OR solver service is being used, the only way to specify the solver to use is through the `<solverToInvoke>` tag in an OSoL file. In this case the valid values for the solver are `clp`, `cbc`, `dylp`, `glpk`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`.

   - **writeosrl (string):** the solution result can be put into an XML file by specifying the option

     ```
     writeosrl  osrl_file_name
     ```

   - **writeosil (string):** the optimization instance can be put into an XML file by specifying the option

```
writeosil  osil_file_name
```

- **writeospl (string):** Specifies the name of an OSpL file in which the answer from the `knock` or `kill` method is written

  ```
  writeospl  write_ospl_file_name
  ```

- **readospl (string):** Specifies the name of an OSpL file that the `knock` method sends to the server

  ```
  readospl  read_ospl_file_name
  ```

- **service_method (string):** Specifies the method to execute on a server. Valid values for this option are `solve`, `getJobID`, `send`, `knock`, `retrieve`, and `kill`.

2.2 Using AMPL

This section is based on the assumption that the user has installed AMPL on his or her machine. It is possible to call all of the COIN-OR solvers listed in Section 1 directly from the AMPL (see `http://www.ampl.com`) modeling language. In this discussion we assume the user has already obtained and installed AMPL. In the download described in Section 5 there is an executable, `OSAmplClient.exe` that is linked to all of the COIN-OR solvers listed in Section 1. From the perspective of AMPL, the `OSAmplClient` acts like an AMPL "solver". The `OSAmplClient.exe` can be used to solve problems either locally or remotely. In the following discussion we assume that the AMPL executable `ampl.exe`, the `OSAmplClient`, and the test problem `eastborne.mod` are all in the same directory.

2.2.1 Using OSAmplClient for a Local Solver

The problem instance `eastborne.mod` is an AMPL model file included in distribution described in Section 5. To solve this problem locally by calling the `OSAmplClient.exe` from AMPL, first start AMPL and then open the `eastborne.mod` file inside AMPL. The test model `eastborne.mod` is a linear integer program.

```
# take in sample integer linear problem
# assume the problem is in the AMPL directory
model eastborne.mod;
```

The next step is to tell AMPL that the solver it is going to use is `OSAmplClient.exe`. Do this by issuing the following command inside AMPL.

```
# tell AMPL that the solver is OSAmplClient
option solver OSAmplClient;
```

It is not necessary to provide the `OSAmplclient.exe` solver with any options. You can just issue the `solve` command in AMPL as illustrated below.

```
# solve the problem
solve;
```

If no options are specified, by default, **Clp** is used for linear programs. For continuous nonlinear models **Ipopt** is used. For mixed-integer linear models, **Cbc** is used. For mixed-integer nonlinear models **Bonmin** is used. If you wish to specify a specific solver, use the `solver` option. For example, since the test problem `eastborne.mod` is a linear integer program, `Cbc` is used by default. If you want to instead use `SYMPHONY`, then you would pass a `solver` option to the `OSAmplclient.exe` solver as follows.

```
# now tell OSAmplClient to use SYMPHONY instead of Cbc
option OSAmplClient_options "solver symphony";
```

Valid values for the `solver` option are `bonmin`, `cbc`, `clp`, `couenne`, `dylp`, `symphony`, and `vol`. Always specify the name of the solver entirely in lower case.

2.2.2 Using OSAmplClient to Invoke the COIN-OR Solver Server

Next, assume that you have a large problem you want solve on the remote solver. It is necessary to specify the location of the server solver as an option to OSAmplClient. In this case, the string of options for `OSAmplClient_options` is:

```
serviceLocation  http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
```

This will send the problem to the solver server at location **webdss.ise.ufl.edu**. The `serviceLocation` option is used to specify the location of a solver server. If, in addition, we want to use the **SYMPHONY** solver, the string of options to `OSAmplClient_options` is:

```
serviceLocation  http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
solver symphony
```

Finally, the user may wish to pass options to the individual solver. This is done by providing an options file. A sample options file, **solveroptions.osol** is provided with this distribution. The name of the options file is the vlaue of the `optionFile` option. the string of options to `OSAmplClient_options` is now

```
serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService"
solver symphony
optionFile solveroptions.osol
```

This `solveroptions.osol` file contains four solver options; two for **Cbc**, one for **Ipopt**, and one for **SYMPHONY**. You can have any number of options. Note the format for specifying an option:

```
<solverOption name="maxN" solver="cbc" value="5" />
```

The attribute `name` specifies that the option name is `maxN` which is the maximum number of nodes allowed in the branch-and-bound tree, the `solver` attribute specifies the name of the solver that the option should be applied to, and the `value` attribute specifies the value of the option. As a second example, consider the specification

```
<solverOption name="max_iter" solver="ipopt" type="integer" value="2000"/>
```

In this example we are specifying an iteration limit for `Ipopt`. Note the additional attribute `type` that has value `integer`. The Ipopt solver requires specifying the data type (string, integer, or numeric) for its options. Different solvers have different options, and we recommend that the user look

at the documentation for the solver of interest in order to see which options are available. A good summary of options for COIN-OR solvers is `http://www.coin-or.org/GAMSlinks/gamscoin.pdf`.

It is also possible to specify the name of the solver and the server location in the options file. Indeed, if you examine the file **solveroptions.osol** you will see that there is an XML tag `<solverToInvoke>` and that the solver given is **symphony**. There is also an XML tag `<serviceURI>` that can be used to specify the location of the server. For the option file **solveroptions.osol** passing the following options to `OSAmplClient_options`

```
solver symphony
optionFile solveroptions.osol
serviceURI http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService";
```

has exactly the same effect as passing the following options to `OSAmplClient_options`

```
optionFile solveroptions.osol;
```

It is possible to override the solver option and the server location given in the option file. For example, if the following options are passed to `OSAmplClient_options`

```
optionFile solveroptions.osol
solver bonmin;
```

then **Bonmin** will be used rather than **SYMPHONY**.

2.2.3 AMPL Summary

1. tell AMPL to use the OSAmplClient as the solver:

   ```
   option solver OSAmplClient;
   ```

2. specify options to the OSAmplclient solver by using the AMPL command **OSAmplClient_options**

3. there are three possible options to specify:

   - the name of the solver using the **solver** option, valid values for this option are `clp`, `cbc`, `dylp`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`.
   - the location of the remote server using the **serviceURI** option
   - the location of the option file using the **optionFile** option

4. the **solver** and **service URI** options can be specified in the options file using the `<solverToInvoke>` and `<serviceURI>` XML tags, respectively.

5. specifying the **solver** or **serviceURI** directly through **OSAmplClient_options** will override the settings in the options file

6. if no options are specified using **OSAmplClient_options**, by default, for continuous linear models clp is used. For continuous nonlinear models ipopt is used. For mixed-integer linear models (MIP), cbc is used. For mixed-integer nonlinear models bonmin is used. All solvers are invoked locally.

7. the options given to **OSAmplClient_options** can be given in any order

3 Calling COIN-OR Solvers Using OSSolverService

The **CoinEasy** binary distribution contains an executable, `OSSolverService.exe` that can be used to solve problems directly on a user's desktop machine, or used to call the remote solver service and solve the problem on the remote server. The `OSSolverService.exe` can take optimization instances in OSiL, AMPL nl format, GAMS dat format, and MPS format.

At present, the `OSSolverService.exe` takes the following parameters (i.e. options). The order of the parameters is irrelevant. Not all the parameters are required. However, if the `solve` or `send` service methods (refer back to Section for 2.1.2) are invoked a problem instance location must be specified.

-osil xxx.osil This is the name of the file that contains the optimization instance in OSiL format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService.exe`. If this option is not specified then the instance location must be specified in the OSoL solver options file.

-osol xxx.osol This is the name of the file that contains the solver options. It is assumed that this file is available in a directory on the machine that is running `OSSolverService.exe`. It is not necessary to specify this option.

-osrl xxx.osrl This is the name of the file that contains the solver solution. A valid file path must be given on the machine that is running `OSSolverService.exe`. It is not necessary to specify this option. If this option is not specified then the solver solution is displayed to the screen.

-osplInput xxx.ospl The name of an input file in the OS Process Language (OSpL), this is used as input to the `knock` method.

-osplOutput xxx.ospl The name of an output file in the OS Process Language (OSpL), this the output string from the `knock` and `kill` method.

-serviceLocation url This is the URL of the solver service. It is not required, and if not specified it is assumed that the problem is solved locally.

-serviceMethod methodName This is the method on the solver service to be invoked. The options are `solve`, `send`, `kill`, `knock`, `getJobID`, and `retrieve`. The use of these options is illustrated in the examples below. This option is not required, and the default value is `solve`.

-mps xxx.mps This is the name of the mps file if the problem instance is in mps format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService.exe`. The default file format is OSiL so this option is not required.

-nl xxx.nl This is the name of the AMPL nl file if the problem instance is in AMPL nl format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService.exe`. The default file format is OSiL so this option is not required.

-dat gamscntr.dat This is the name of the GAMS control file. It is necessary to edit this file and make sure that the path to the GAMS executable is correct for your machine.

-solver solverName Valid values for this option are `clp`, `cbc`, `dylp`, `ipopt`, `bonmin`, `couenne`, `symphony`, and `vol`. This is an optional option. By default, for continuous linear models Clp is used. For continuous nonlinear models Ipopt is used. For mixed-integer linear models (MIP), Cbc is used. For mixed-integer nonlinear models Bonmin is used.

-config pathToConfigureFile This parameter specifies a path on the local machine to a text file containing values for the input parameters. This is convenient for the user not wishing to constantly retype parameter values.

Typing at the command line

```
OSSolverService.exe --help
```

will provide the list of input parameters. The following command will invoke the `Cbc` solver (by default) on the local machine to solve the problem instance `eastborne.osil`. When invoking the commands below involving `OSSolverService.exe` we assume that: 1) the user is connected to the directory where the `OSSolverService.exe` executable is located, and 2) that all files referred to are in the same directory as the `OSSolverService.exe`.

3.1 Solving Problems Locally

Generally, when solving a problem locally the user will use the `solve` service method. The `solve` method is invoked synchronously and waits for the solver to return the result. In this case, the `OSSolverService.exe` reads a file on the hard drive (for example, an `nl` or `OSiL` file) with the optimization instance. The optimization instance is parsed into a string which is passed to the `OSLibrary`, which is linked with various solvers. The result of the optimization is passed back to the `OSSolverService.exe` as a string in OSrL format. A simple example of a call is

```
OSSolverService.exe  -osil   eastborne.osil
```

Notice that a solver is not specified. Since `eastborne.osil` is a linear integer program, the default solver **Cbc** is used.

Here is a more complicated set of options we could pass to the `OSSolverService.exe` at the command line:

```
-osil  eastborne.osil
-solver bonmin
-osrl  temp.osrl
-osol solveroptions.osol
```

The first parameter `-osil astborne.osil` is the local location of the OSiL file, `eastborne.osil`, that contains the problem instance. The second parameter, `-solver bonmin`, is the solver to be invoked, in this case **Bonmin** instead of the default **Cbc**. The third parameter `-osrl temp.osrl` specifies that the result should be written in OSrL format to the file `tem.osrl`. The forth paramter is an options file, `solveroptions.osol` to be passed to the Bonmin solver. The parameters listed above could all be given at the command line or they could be put in a text file, `configfile.config`, and used as

```
OSSolverService.exe  -config configfile.config
```

3.2 Solving Problems Remotely with Web Services

We now provide examples that illustrate using the `OSSolverService.exe` executable to call the COIN-OR remote solver service. In the following sections we illustrate each of the six service methods. This is basically a repeat of Section 2.1.2 applied to the `OSSolverService.exe`. All of the concepts introduced in Section 2.1.2 carry over here.

3.2.1 The `solve` Service Method

First we illustrate a remote `solve` call. A example problem instance `eastborne.osil` is solved on the COIN-OR solver server by passing the following options to the `OSSolverService.exe`.

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osil eastborne.osil
```

No solver is specified so by default the `Cbc` solver is used by the `OSSolverService.exe`. The default `serviceMethod` is `solve` and that is what get invoked by using the two options above. An equivalent set of options is

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osil eastborne.osil
-serviceMethod solve
-solver Cbc
```

The file with the optimization instance does not need to reside on the user's machine. The user can tell the COIN-OR solver server to get the optimization instance off of another machine. Consider the following options:

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osol remoteSolve.osol
```

Examine the file `remoteSolve.osol` that is provided with this distribution. This option file `remoteSolve.osol` contains an XML tag `<instanceLocation>`. See below.

```
 <instanceLocation locationType="http">
 http://www.coin-or.org/OS/parincLinear.osil
 </instanceLocation>
```

The `<instanceLocation>` has an attribute `locationType` that has value `http`. This tells the solve server to use an `http` get method to get the problem instance `parincLinear.osil` from the location `http://www.coin-or.org/OS`.

3.2.2 The `getJobID` Service Method

The `send` method is used for asynchronous communication. Before submitting a job with the `send` method a `JobID` is required. The `OSSolverService.exe` can get a `JobID` from the solver server using the following options.

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-serviceMethod getJobID
```

A string is returned and printed to standard out with the `JobID`.

3.2.3 The `send` Service Method

When the `solve` service method is used, the `OSSolverService.exe` does not finish execution until the solution is returned from the remote solver service. When the `send` method is used the instance is communicated to the remote service and the `OSSolverService.exe` terminates after submission. An example set of options for the `send` method is

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osol sendWithJobID.osol
-nl  eastborne.nl
-serviceMethod send
```

Since the `send` method involves asynchronous communication the remote solver service must keep track of jobs. The `send` method requires a `JobID`. In the above example, the `jobID` is provided in the `<jobID>` XML tag in the `sendWithJobID.osol` file. For example:

```
<jobID>coinor12345xyz</jobID>
```

If no `JobID` is specified the `OSSolverService.exe` method first invokes the `getJobID` service method to get a `JobID`, puts this information into an OSoL file it creates (or inserts in the current OSoL file that does not have the `JobID` tag), and sends the information to the server. Of course a value for `serviceLocation` must be provided.

3.2.4 The `knock` Service Method

The `OSSolverService.exe` process terminates after executing the `send` method. Therefore, it is necessary to know when the job is completed on the remote server. One way is to include an email address in the `<contact>` element with the attribute `transportType` set to `smtp`. A second way to check on the status of a job is to use the `knock` service method. For example, assume a user wants to know if the job with `JobID coinor12345xyz` is complete. A user would make the request with `OSSolverService.exe` using the options:

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osplInput  knock.ospl
-osol  sendWithJobID.osol
-serviceMethod knock
```

where the `sendWithJobID.osol` file has the `JobID` XML tag of the job for which we are seeking information. The result of this request is a string in OSpL format, with the data contained in its `processData` section. The result is displayed on the screen; if the user desires it to be redirected to a file, an additional option should be added that specifies a valid path name on the local system, e.g.,

```
-osplOutput result.ospl
```

The resulting XML in the OSpL language should have a `<state>` element contained in the

```
<job jobID="coinor12345xyz">
```

element indicating that the job is finished.

When making a `knock` request, the OSoL string can be empty. In this example, if the OSoL string had been empty the status of more than one job may be returned. In our default solver service implementation, there is a configuration file `OSParameter` that has a parameter `MAX_JOBIDS_TO_KEEP`. The current default setting is 100. In a large-scale or commercial implementation it might be wise to keep problem results and statistics in a database. Also, there are values other than `getAll` (i.e., get all process information related to the jobs) for the OSpL `action` attribute in the `<request>` tag. For example, the `action` can be set to a value of `ping` if the user just wants to check if the remote solver service is up and running. For details, check the OSpL schema.

3.2.5 The `retrieve` Service Method

The `retrieve` method is used to retrieve the solver solution. This method has a single string argument which is an OSoL string. A user would make a `retrieve` request with `OSSolverService.exe` using the options:

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osol sendWithJobID.osol
-serviceMethod retrieve
-osrl test.osrl
```

The OSoL file `sendWithJobID.osol` contains a tag `<jobID>` that is communicated to the remote service. The remote service locates the result and returns it as a string. The `<jobID>` should reflect a `<jobID>` that was previously submitted using a `send()` command. The result is returned as a string in OSrL format to the file test.osrl.

3.2.6 The `kill` Service Method

If the user submits a job that is taking too long or is a mistake, it is possible to kill the job on the remote server using the `kill` service method. A user would make a `kill` request with `OSSolverService.exe` using the options:

```
-serviceLocation http://webdss.ise.ufl.edu:2646/OSServer/services/OSSolverService
-osol sendWithJobID.osol
-serviceMethod kill
```

The OSoL file `sendWithJobID.osol` contains a tag `<jobID>` that is communicated to the remote service. The result is returned in OSpL format.

3.3 Summary Rules

1. When using the `send()` or `solve()` methods a problem instance file location **must** be specified either at the command line, in the configuration file, or in the `<instanceLocation>` element in the OSoL options file.

2. The default `serviceMethod` is `solve` if another service method is not specified. The service method cannot be specified in the OSoL options file.

3. If the `solver` option is not specified, by default, for continuous linear models **Clp** is used. For continuous nonlinear models **Ipopt** is used. For mixed-integer linear models (MIP), **Cbc** is used. For mixed-integer nonlinear models **Bonmin** is used.

4. If the options `send`, `kill`, `knock`, `getJobID`, or `retrieve` are specified, a `serviceLocation` must be specified.

5. Parameters specified in the configure file are overridden by parameters specified at the command line. This is convenient if a user has a base configure file and wishes to override only a few options.

Summary of Communication Protocols:

- `solve(osil, osol):`

  - Takes OSiL and OSoL and returns OSrL (string/file version)
  - Synchronous call, blocking request/response

- `getJobID(osol)`
  - Gets a unique job id generated by the solver service
  - Maintain session and state on a distributed system

- `send(osil, osol)`
  - Same signature as the solve function but returns a boolean
  - Asynchronous (server side), non-blocking call

- `knock(ospl, osol)`
  - Get process and job status information from the remote server

- `retrieve(osol)`
  - Retrieving result from anywhere anytime

- `kill(osol)`
  - kill remote optimization jobs
  - Critical in long running optimization jobs

4 More Sophisticated Methods

We have presented two approaches to using the COIN-OR solver server. One approach is to use a modeling language (GAMS or AMPL) to generate the problem and send it to the server. The second approach is to use the **OSSolverService.exe** to send an optimization instance in OSiL, MPS, AMPL nl, or GAMS dat to the server. An alternative to these two methods is for the user to write C++ code and generate an application that links to various COIN-OR. This approach is described in a separate document. See . Numerous examples are given in the document. For example, how to use a C++ to directly generate the model.

5 Obtaining the Files

The applications and example files discussed in the document are contained in a single download. The user should be up and running and ready to duplicate every example in this document by downloading the distribution at *****. This distribution contains:

- `OSAmplClient.exe` an executable used by AMPL to connect to the CoinAll solvers
- `OSSolverService` a command line executable used to run the CoinAll solvers
- `gmsos_.zip` (to be put in the GAMS directory if you use GAMS)
- `eastborne.gms` a GAMS model test problem

- `eastborne.mod` an AMPL model test problem
- `eastborne.nl` an test problem in AMPL nl format
- `eastborne.osil` a test problem in OSiL format
- `knock.ospl` – a sample file in OSpL format
- `remoteSolve.osol` – an options file for call instances on a third-party machine
- `sendWithJobID.osol` – a sample options file
- `solveroptions.osol` – a sample options file