

# Optimization Services 1.0 User's Manual

Robert Fourer, Jun Ma, Kipp Martin

June 16, 2007

## **Abstract**

This is the User's Manual for the Optimization Services (OS) project. The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site for more information.

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Download and Installation</b>	<b>4</b>
2.1	Obtaining the Source Code Subversion Repository (SVN) . . . . .	4
2.2	Obtaining the Source Code From a Tarball . . . . .	6
2.3	Obtaining a Visual Studio Project . . . . .	6
2.4	Obtaining the Binaries . . . . .	6
2.5	Platforms . . . . .	6
<b>3</b>	<b>The OS Library Components</b>	<b>6</b>
3.1	OSAgent . . . . .	6
3.2	OSCommonInterfaces . . . . .	7
3.3	OSModelInterfaces . . . . .	7
3.4	OSParser . . . . .	7
3.5	OSSolverInterfaces . . . . .	7
3.6	OSUtils . . . . .	8
<b>4</b>	<b>OSInstance: A General Instance API</b>	<b>8</b>
4.1	Get Methods . . . . .	8
4.2	Set Methods . . . . .	8
4.3	Calculate Methods . . . . .	8
<b>5</b>	<b>Hooking to An Algorithmic Differentiation Package</b>	<b>8</b>
<b>6</b>	<b>The OSSolverService</b>	<b>8</b>
6.1	OSSolverService Input Parameters . . . . .	9
6.2	Solving Problems Locally . . . . .	11
6.3	Solving Problems Remotely with Web Services . . . . .	11
6.3.1	The Remote <code>solve</code> Service Method . . . . .	12
6.3.2	The Remote <code>send</code> Service Method . . . . .	13
6.3.3	The Remote <code>retrieve</code> Service Method . . . . .	13
6.3.4	The Remote <code>getJobID</code> Service Method . . . . .	13
6.3.5	The Remote <code>knock</code> Service Method . . . . .	13
6.3.6	The Remote <code>kill</code> Service Method . . . . .	13
<b>7</b>	<b>Setting up a Solver Service with Tomcat</b>	<b>13</b>
<b>8</b>	<b>Examples</b>	<b>13</b>
8.1	AMPL Client: Hooking AMPL to Solvers . . . . .	13
8.2	CppAD: Using the CppAD Algorithmic Differentiation Package . . . . .	14
8.3	File Upload: Using a File Upload Package . . . . .	14
8.4	Instance Generator: Using the OSInstance API to Generate Instances . . . . .	14

**List of Figures**

**List of Tables**

# 1 Introduction

The objective of Optimization Services (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site for more information.

## 2 Download and Installation

OS is released as open source code under the Common Public License (CPL). This project was created by Robert Fourer, Jun Ma, and Kipp Martin. The code has been written primarily by Jun Ma, Kipp Martin, Robert Fourer and Huanyuan Sheng, the first two are the COIN project leaders for OS. Below we describe different methods for obtaining the C++ source code and binaries.

### 2.1 Obtaining the Source Code Subversion Repository (SVN)

The C++ source code can be obtained using Subversion. Users with Unix operating systems will most likely have an svn client. For Windows users wishing to obtain an SVN client we recommend `***kipp fillin`

The OS project page with a Wiki is available at `projects.coin-or.org\OS`. Execute the following steps to get the source code using SVN.

**Step 1:** Connect to a directory where you want the OS project to go. The following command will download the project into the directory COIN-OS

```
svn co https://projects.coin-or.org/svn/OS/stable/1.0 COIN-OS
```

**Step 2:** Connect to the distribution root directory.

```
cd COIN-OS
```

**Step 3:** Run the configure script that will generate the makefiles.

```
./configure
```

**Step 4:** Run the make files.

```
make
```

**Step 5:** Run the unitTest.

```
make test
```

Depending upon which third party software you have installed, the result of running the unitTest should look something like:

HERE ARE THE UNIT TEST RESULTS:

```
Solved problem avion2.osil with Ipopt
Solved problem HS071.osil with Ipopt
Solved problem rosenbrockmod.osil with Ipopt
Solved problem parincQuadratic.osil with Ipopt
Solved problem parincLinear.osil with Ipopt
Solved problem callBack.osil with Ipopt
Solved problem callBackRowMajor.osil with Ipopt
Solved problem parincLinear.osil with Clp
Solved problem p0033.osil with Cbc
Solved problem rosenbrockmod.osil with Knitro
Solved problem callBackTest.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem p0033.osil with SYMPHONY
Solved problem parincLinear.osil with DyLP
Solved problem lindoapiaddins.osil with Lindo
Solved problem rosenbrockmod.osil with Lindo
Solved problem parincQuadratic.osil with Lindo
Solved problem wayneQuadratic.osil with Lindo
Test the MPS -> OSiL converter on parinc.mps usig Cbc
Test the AMPL nl -> OSiL converter on hs71.nl using LINDO
Test a problem written in b64 and then converted to OSInstance
Successful test of OSiL parser on problem parincLinear.osil
Successful test of OSrL parser on problem parincLinear.osrl
Successful test of prefix and postfix conversion routines on problem rosenbrockmod.osil
Successful test of all of the nonlinear operators on file testOperators.osil
Successful test of AD gradient and Hessian calculations on problem CppADTestLag.osil
```

CONGRATULATIONS! YOU PASSED THE UNIT TEST

If you do not see

CONGRATULATIONS! YOU PASSED THE UNIT TEST

then you have not passed the unitTest and hopefully some semi-intelible error message was given. CONGRATULATIONS! YOUPASSEDD THE UNIT TEST

**Step 6:** Install the libraries.

`make install`

This will install all of the libraries in the `lib` directory under the distribution root.

The above steps are fully tested on Mac/Unix, Linux, and on Windows using either MINGW/MSYS or CYGWIN. Popular compilers like gcc/g++ or windows native compiler cl.exe can all be used.

Note if you download the OS package, you get these additional COIN-OR projects.

- Cbc `projects.coin-or.org\Cbc`
- Clp `projects.coin-or.org\Clp`
- CppAD `projects.coin-or.org\CppAD`
- Dylp `projects.coin-or.org\Dylp`
- Osi `projects.coin-or.org\Osi`
- SYMPHONY `projects.coin-or.org\SYMPHONY`

## 2.2 Obtaining the Source Code From a Tarball

## 2.3 Obtaining a Visual Studio Project

## 2.4 Obtaining the Binaries

## 2.5 Platforms

	Mac	Linux	Cyg-gcc	Msys-cl	Msys-gcc	MSVS
AMPL-Client	x	x		x		
Cbc	x	x	x	x		
Clp	x	x	x	x		
Cplex	x	x				
DyLP	x	x	x	x		
Ipopt	x	x				
Knitro	x					
Lindo	x	x		x		
SYMPHONY	x	x	x	x		

### Platform Detail:

	Operating System	Compiler	Hardware
Mac	Mac OS X 10.4.9	gcc 4.0.1	Power PC
Linux	Red Hat 3.4.6-8	gcc 3.4.6	Dell Intel 32 bit chip
Cyg-gcc	Windows 2003 Server	gcc 3.4.4	Dell Intel 32 bit chip
Msys-cl	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip
Msys-gcc			
MSVS	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip

## 3 The OS Library Components

### 3.1 OSAgent

The **OSAgent** part of the library is used to facilitate communication with remote solvers. It is not used if the solver is invoked locally (i.e. on the same machine).

## 3.2 OSCommonInterfaces

## 3.3 OSModelInterfaces

## 3.4 OSParsers

## 3.5 OSSolverInterfaces

The `OSSolverInterfaces` library is designed to facilitate linking the OS library with various solver APIs. We first describe how to take a problem instance in OSiL format and connect to a solver that has a COIN-OR OSI interface. See the OSI project [www.projects.coin-or.org/0si](http://www.projects.coin-or.org/0si). We then describe hooking to the COIN-OR nonlinear code `Ipopt`. See [www.projects.coin-or.org/Ipopt](http://www.projects.coin-or.org/Ipopt). Finally we describe hooking to two commercial solvers KNITRO and LINDO.

The OS library has been tested with the following solvers using the Osi Interface.

- Cbc
- Clp
- Cplex
- DyLP
- Glpk
- SYMPHONY

In the `OSSolverInterfaces` library there is an abstract class `DefaultSolver` that has the following key members:

```
std::string osil;  
std::string osol;  
std::string osrl;  
OSInstance *osinstance;  
OSResult *osresult;
```

and the pure virtual function

```
virtual void solve() = 0 ;
```

In order to use a solver through the COIN-OR `Osi` interface it is necessary to an object in the `CoinSolver` class which inherits from the `DefaultSolver` class and implements the appropriate `solve()` function. We illustrate with the `Clp` solver.

```
DefaultSolver *solver = NULL;  
solver = new CoinSolver();  
solver->m_sSolverName = "clp";
```

Assume that the data file containing the problem has been read into the string `osil` and the solver options are in the string `osol`. Then the `Clp` solver is invoked as follows.

```
solver->osil = osil;  
solver->osol = osol;  
solver->solve();
```

Finally, get the solution in `OSrL` format as follows

```
cout << solver->osrl << endl;
```

Even though `LINDO` and `KNITRO` are commercial solvers and do not have a `COIN-OR Osi` interface these solvers are used in exactly the same manner as a `COIN-OR` solver. For example, to invoke the `LINDO` solver we do the following.

```
solver = new LindoSolver();
```

Similarly for `KNITRO` and `Ipopt`. In the case of the `KNITRO`, the `KnitroSolver` class inherits from both `DefaultSolver` class and the `KNITRO NlpProblemDef` class. See [Kipp--putinKnitromanual](#) for more information on the `KNITRO` solver C++ implementation and the `NlpProblemDef` class. Similarly, for `Ipopt` the `IpoptSolver` class inherits from both the `DefaultSolver` class and the `Ipopt TNLP` class. See [Kipp--putinIpoptmanual](#) for more information on the `Ipopt` solver C++ implementation and the `TNLP` class.

In the examples above the problem instance was assumed to be read from a file into the string `osil` and then into the class member `solver->osil`. However, everything can be done entirely in memory. For example, it is possible to use the `OSInstance` class to create an in-memory problem representation and give this representation directly to a solver class that inherits from `DefaultSolver`. The class member to use is `osinstance`. This is illustrated in the example given in Section 8.4.

## 3.6 OSUtils

# 4 OSInstance: A General Instance API

## 4.1 Get Methods

## 4.2 Set Methods

## 4.3 Calculate Methods

# 5 Hooking to An Algorithmic Differentiation Package

# 6 The OSSolverService

The `OSSolverService` is a command line executable designed to pass problem instances to solvers and get the result back to be displayed either to standard output or a specified browser. The `OSSovlerService` can be used to invoke a solver locally or on a remote server. It can work either synchronously or asynchronously.



## 6.1 OSSolverService Input Parameters

At present, the `OSSolverService` takes the following parameters. The order of the parameters is irrelevant. Not all the parameters are required. However, two pieces of information *must* be contained in the input parameters. First, the problem instance location must be specified (either locally or on a remote machine). Second, the solver 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`. 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`. 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`. It is not necessary to specify this option.

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

**-serviceMethod method** this is the solver service required. 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`. 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`. The default file format is OSiL so this option is not required.

**-solver solverName** this parameter is required if the problem is solved locally. Possible values for default OS installation are `tt clp` (COIN-OR Clp), `cbc` (COIN-OR Cbc), `dylp` (COIN-OR DyLP), and `symphony` (COIN-OR SYMPHONY). Other solvers supported (if the necessary libraries are present) are `cplex` (Cplex through COIN-OR Osi), `glpk` (glpk through COIN-OR Osi), `ipopt` (COIN-OR Ipopt), `knitro` (Knitro), and `lindo` LINDO.

**-browser browserName** this parameter is a path to the browser on the local machine. If this optional parameter is specified then the solver result in OSrL format is transformed using XSLT into HTML and displayed in the browser.

**-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.

#### Some 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 file.
2. The **serviceMethod** option must be specified at the command line or in a configuration file. The service method cannot be specified in the OSoL options file.
3. If the **solver** option is not specified, the COIN-OR solver **Cbc** is the default solver used. In this case an error is thrown if the problem instance has quadratic or other nonlinear terms.
4. If the options **=send**, **kill**, **knock**, **getJobID**, or **retrieve** are specified, a **serviceLocation** must be specified.
5. The default **serviceMethod** used is **solve** if one is not specified.
6. When calling a remote server, the solver option must be specified in the OSoL file.

The input parameters to the **OSSolverService** may be given entirely in the command line or in a configuration file. We first illustrate giving all the parameters in the command line. The following command will invoke the **Clp** solver on the local machine to solve the problem instance **parincLinear.osil**.

```
OSSolverService -solver clp -osil ../data/parincLinear.osil
```

Alternatively, these parameters can be put into a configuration file. Assume that the configuration file of interest is **testlocalclp.config**. It would contain the two lines of information

```
-osil ../data/parincLinear.osil  
-solver clp
```

Then the command line is

```
OSSolverService -config ../data/testlocalclp.config
```

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. For example,

```
OSSolverService -config ../data/testlocalclp.config -solver lindo
```

or

```
OSSolverService -solver lindo -config ../data/testlocalclp.config
```

will result in the LINDO solver being used even though **Clp** is specified in the **testlocalclp** configure file.

## 6.2 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. The `solve` method has two arguments. The first argument is the OSoL file path and the second argument is the OSiL file path. The OSoL file path may be empty. Here is an illustration of using a configure file, `testlocal.config`, to invoke `Ipopt` locally.

```
-osil ../data/parincQuadratic.osil
-solver ipopt
-serviceMethod solve
-browser /Applications/Firefox.app/Contents/MacOS/firefox
-osrl /Users/kmartin/temp/test.osrl
```

The first line of `testlocal.config` gives the local location of the OSiL file, `parincQuadratic.osil`, that contains the problem instance. The second parameter, `-solver ipopt`, is the solver to be invoked. The third parameter `-serviceMethod solve` is not really needed, but included only for purpose of illustration. The default solver service is `solve`. The fourth parameter is the location of the browser on the local machine. It will read the OSrL file on the local machine using the path specified by the value of the `osrl` parameter, in this case `/Users/kmartin/temp/test.osrl`.

In the configuration file `testlocalosol.config` we illustrate specifying the instance location in an OSoL file.

```
-osol ../data/demo.osol
-solver clp
```

The file `demo.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="os.optimizationservices.org
http://www.optimizationservices.org/schemas/OSoL.xsd">
  <general>
    <instanceLocation locationType="local">
      ../data/parincLinear.osil
    </instanceLocation>
  </general>
</osol>
```

## 6.3 Solving Problems Remotely with Web Services

We now illustrate using the `OSSolverService` executable to call a remote solver service. By remote solver service we mean a solver service that is called using Web services. Both the `OSSolverService` and the Web service could be on the same physical machine.

In the following sections we illustrate each of the six service methods. All of these examples are based upon a solver service running on a remote server running Apache Tomcat. See [tomcat.apache.org](http://tomcat.apache.org). The Web Service running on the server is a Java program based on Apache Axis. See [ws.apache.org/axis](http://ws.apache.org/axis). This is described in greater detail in Section 7. This Web Service is called `OSSolverService.jws`. This Java program calls a C++ executable `OSSolverService` on the server. So there is an `OSSolverService` on the client that calls the Web Service `OSSolverService.jws` that in turn calls the executable `OSSolverService` on the server. See Figure ??.

kip – put in a figure of the client `OSSolverService` call `OSSolverService.jws` which in turn calls `OSSolverService`.

### 6.3.1 The Remote solve Service Method

First we illustrate a simple call to `OSSolverService.jws` and request a solution using the COIN-OR Clp solver. The call on the client machine is

```
OSSolverService -config ../data/testremote.config
```

where the `testremote.config` file is

```
-osil ../data/parincLinear.osil
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
```

No solver is specified so by default the Cbc solver will be used on the server.

Now use an OSoL options file

```
OSSolverService -osol ../data/remoteSolve1.osol -osil ../data/parincLinear.osil
```

where `remoteSolve1.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
  <general>
    <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
  </general>
  <optimization>
    <other name="os_solver">clp</other>
  </optimization>
</osol>
```

In this case we specify a solver to use, name Clp.

Next we illustrate a call to the remote SolverService and specify an OSiL instance that is on the remote machine.

```
OSSolverService -osol ../data/remoteSolve2.osol
```

where the `remoteSolve2.osol` file is

```

<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
  <general>
    <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
    <instanceLocation locationType="local">/home/kmartin/files/code/OSRepository/li
  </general>
  <optimization>
    <other name="os_solver">clp</other>
  </optimization>
</osol>

```

if we were to change to the `locationType` attribute in the `<instanceLocation>` element to `http` then we could specify the instance location to on yet another machine. This is illustrate below for `remoteSovle3.osol`.

```

<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
  <general>
    <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
    <instanceLocation locationType="http">http://gsbkip.chicagogsb.edu/testproblems
  </general>
  <optimization>
    <other name="os_solver">clp</other>
  </optimization>
</osol>

```

kipp illustrate the above with a figure

Illustrate when data is uploaded locally or when data is uploaded from another server.

### 6.3.2 The Remote send Service Method

illustrate getting email with smpt

### 6.3.3 The Remote retrieve Service Method

### 6.3.4 The Remote getJobID Service Method

### 6.3.5 The Remote knock Service Method

### 6.3.6 The Remote kill Service Method

## 7 Setting up a Solver Service with Tomcat

## 8 Examples

### 8.1 AMPL Client: Hooking AMPL to Solvers

The `amplClient` executable is designed to work with the AMPL program. See [www.ampl.com](http://www.ampl.com). The `amplClient` acts like an AMPL “solver.” The `amplClient` is linked with the OS

library and can be used to solve problems either remotely. In both cases the `amplClient` uses the `OSnl2osil` class to convert the AMPL generated `nl` file (which represents the problem instance) into the corresponding instance representation in the OSiL format.

For example, assume that there is a problem instance, `hs71.mod` in AMPL model format. To solve this problem locally by calling the `amplClient` from AMPL first start AMPL and then execute the following commands. In this case we are assuming that the local solver used is `Ipopt`.

```
# take in problem 71 in Hock and Schittkowski
# assume the problem is in the AMPL directory
model hs71.mod;
# tell AMPL that the solve is amplClient
option solver amplClient;
# now tell amplClient to use Ipopt
option amplClient_options "solver ipopt";
# the name of the nl file (this is optional)
write gtestfile;
# now solve the problem
solve;
```

This will invoke `Ipopt` locally and the result in OSrL format will be displayed on the screen. In order to call a remote solver service, after the command

```
option amplClient_options "solver ipopt";
```

provide an option which has the address of the remote solver service.

```
option ipopt_options "http://128.135.130.17:8080/ipopt/IPOPTSolverService.jws";
```

## 8.2 CppAD: Using the CppAD Algorithmic Differentiation Package

## 8.3 File Upload: Using a File Upload Package

## 8.4 Instance Generator: Using the OSInstance API to Generate Instances