COIN-OR Vendor Workshop

Bradley M. Bell, University of Washington
Robin Lougee-Heimer, IBM
Kipp Martin, University of Chicago

October 11, 2008

Outline – Where we are headed

The CoinBinary Project

Objective: *provide executables and libraries (i.e. binaries as opposed to source code) of key COIN-OR projects.*

Users often **do not wish to download and compile source code**!

The **CoinBinary** project provides a binary distribution of **CoinAll**.

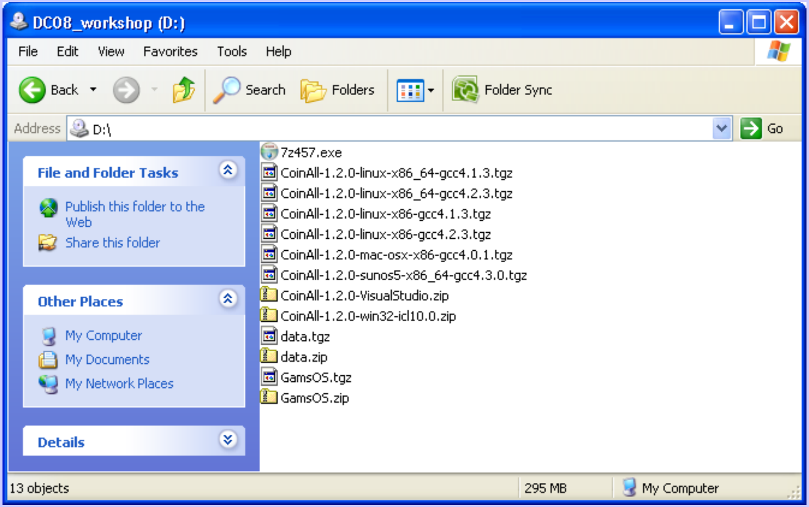
CoinAll is a set of key COIN-OR projects – but not all projects.

See: `http://www.coin-or.org/download/binary/CoinAll/`.

You have a CD with the **CoinAll** binary distribution.

The CoinAll CD

The workshop CoinAll CD:



7z457.exe
CoinAll-1.2.0-linux-x86_64-gcc4.1.3.tgz
CoinAll-1.2.0-linux-x86_64-gcc4.2.3.tgz
CoinAll-1.2.0-linux-x86-gcc4.1.3.tgz
CoinAll-1.2.0-linux-x86-gcc4.2.3.tgz
CoinAll-1.2.0-mac-osx-x86-gcc4.0.1.tgz
CoinAll-1.2.0-sunos5-x86_64-gcc4.3.0.tgz
CoinAll-1.2.0-VisualStudio.zip
CoinAll-1.2.0-win32-icl10.0.zip
data.tgz
data.zip
GamsOS.tgz
GamsOS.zip

13 objects 295 MB My Computer

The CoinAll CD

The CD with **CoinAll** contains the following:

- **CoinAll-1.2.0** – folders containing the CoinAll release 1.2.0 project compiled on **eight different platforms**.
- **data (.zip and .tgz)** – test problems in various formats
- **GAMSlinks (.tgz and .zip)** – a COIN-OR project for using the GAMS modeling language with COIN-OR solvers
  - Note: the GAMS modeling language must be obtained seperately
- **7z457.exe** – a Windows utility to unzip the zipped files

The CoinAll CD

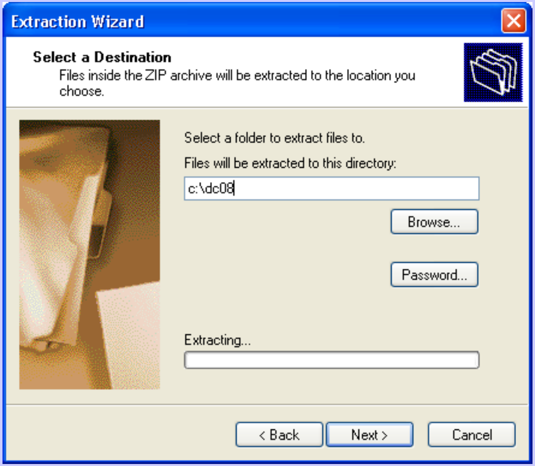
Windows Users: extract the following files from the CD into a single directory (dc08) on your hard drive:

- CoinAll-1.2.0-VisualStudio.zip
- data.zip
- GamsOS.zip (If you have GAMS on your machine and want to use COIN-OR solvers with GAMS)

Please move the **stylesheets** directory in the **data** directory into the **CoinAll-1.2.0-VisualStudio** directory.

The CoinAll CD

Windows Users: extracting the files by right-clicking on desired file and choosing **Extract All ...**



Extraction Wizard

Select a Destination

Files inside the ZIP archive will be extracted to the location you choose.

Select a folder to extract files to.

Files will be extracted to this directory:

c:\dc08

Extracting...

The CoinAll CD

Linux and Mac OS X Users: extract from the CD the following files into a single directory (dc08) on your hard drive:

- CoinAll-1.2.0-platform-compiler.tgz, for example:

  CoinAll-1.2.0-linux-x86-gcc4.2.3.tgz (Linux 32 bit users)

  CoinAll-1.2.0-macosx-x86-gcc4.0..tgz1 (Mac OS X users)

- data.tgz

- GamsOS.tgz (If you have GAMS on your machine and want to use COIN-OR solvers with GAMS)

The CoinAll CD

The complete list:

- CoinAll-1.2.0-linux-x86_64-gcc4.1.3
- CoinAll-1.2.0-linux-x86_64-gcc4.2.3
- CoinAll-1.2.0-linux-x86-gcc4.1.3
- CoinAll-1.2.0-linux-x86-gcc4.2.3
- CoinAll-1.2.0-mac-osx-x86-gcc4.0.1
- CoinAll-1.2.0-sunos5-x86_64-gcc4.3.0
- CoinAll-1.2.0-VisualStudio
- CoinAll-1.2.0-win32-icl10.0

The CoinAll CD

Linux and Mac OS X Users: unpack your files

```
tar -xzf /media/cdrom1/data.tgz   (Ubuntu)

tar -xzf /Volumes/DC08_workshop/data.tgz (Mac OS X)
```

Or, using gnome, right-click and select **Extract to...**

What Can You Do With CoinAll?

- Use the solvers out-of-the-box to solve optimization problems
  - solve problems in `mps`, `nl`, or `osil` format
  - use with modeling languages such as AMPL, GAMS, and MPL
- Write code to call solvers as part of a larger application
- Use the APIs (application program interface) to build models
- Use the APIs to build customized solution procedures (e.g. cutting planes, column generation, etc.)
- Research/Teaching/Consulting/Business Applications

COIN-OR Projects

A few project highlights (among the 30 COIN-OR projects):

- **Clp:** Coin linear program

  Project Manager: John Forrest

- **Cbc:** Coin branch and cut – an integer programming package

  Project Manager: John Forrest

- **Ipopt:** Interior Point OPTimizer – this came from a state-of-the-art research code using interior methods to solve nonlinear optimization problems.

  Project Manager: Andreas Wächter

The above packages are used to solve real problems:
https://projects.coin-or.org/Ipopt/wiki/SuccessStories
https://projects.coin-or.org/Cbc/wiki/SuccessStories

COIN-OR Projects

A few other project highlights:

- **SYMPHONY:** an integer programming package that supports parallel processing.

  Project Manager: Ted Ralphs

- **Bonmin:** Basic Open-source Nonlinear Mixed INteger programming is for nonlinear integer programming.

  Project Manager: Pierre Bonami

- **Osi:** Open solver interface – a generic API for mixed integer linear programs. With this you can call solvers such as Cplex and Glpk.

  Project Manager: Matthew Saltzman

COIN-OR Projects

A few other project highlights:

- **GAMSlinks:** allows you to use the GAMS algebraic modeling language and call COIN-OR solvers. Not part of `CoinAll`, but on the CD.

  Project Manager: Stefan Vigerske

- **FLOPC++:** an open-source modeling language.

  Project Manager: Tim Hultberg

- **Cgl:** Cut Generation Library (Cgl) is an open collection of cutting plane implementations (”cut generators”) for use in teaching, research, and applications. Cgl can be used with other COIN-OR packages that make use of cuts, such as the mixed-integer linear programming solver Cbc.

  Project Manager: Robin Lougee-Heimer and Francois Margot

COIN-OR Projects

A few other project highlights:

- **CppAD:** a project managed by Brad Bell for doing algorithmic differentiation – a key ingredient in modern nonlinear optimization codes.

  Project Manager: Brad Bell

- **Bcp:** Branch cut-and-price. Provides, for example, a generic framework to do column generation.

  Project Manager: Laci Ladanyi

COIN-OR Projects

A few other project highlights:

- **CoinMP:** – a callable library that wraps around CLP, CGL, and CBC, providing C-API interface just like CPLEX, XPRESS, and LINDO.

  Project Manager: Bjarni Kristjansson

- **Optimization Services:** – a framework for doing distributed optimization.

  Project Manager: Jun Ma, Gus Gassmann, and Kipp Martin

Optimization Services (OS)

Optimization Services (OS) integrates numerous COIN-OR projects. The OS project provides:

- A set of XML based standards for representing optimization instances (OSiL), optimization results (OSrL), and optimization solver options (OSoL).
- A robust API for linear and nonlinear problems.
- A command line executable OSSolverService for reading problem instances (OSiL format, nl format, MPS format) and calling a solver either locally or on a remote server.
- Utilities that convert AMPL nl files into the OSiL format and MPS files into the OSiL format.

Optimization Services (OS)

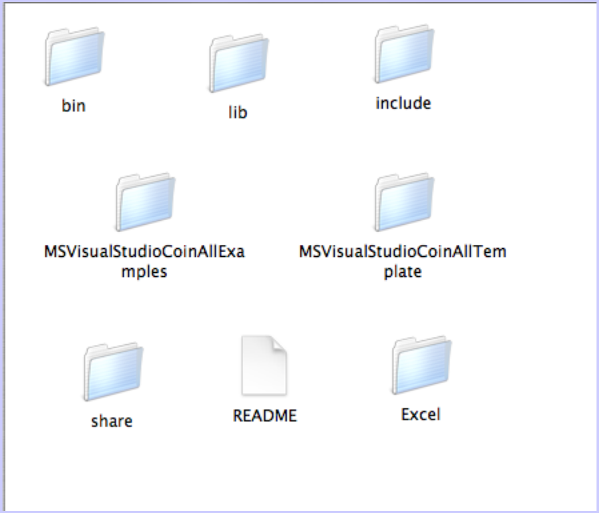
OS Continued ...

- Client side software that is used to create Web Services SOAP packages with OSiL instances and contact a server for solution.
- Standards that facilitate the communication between clients and solvers using Web Services.
- Server software that works with Apache Tomcat.

What follows is done using the OS project, rather than projects individually.

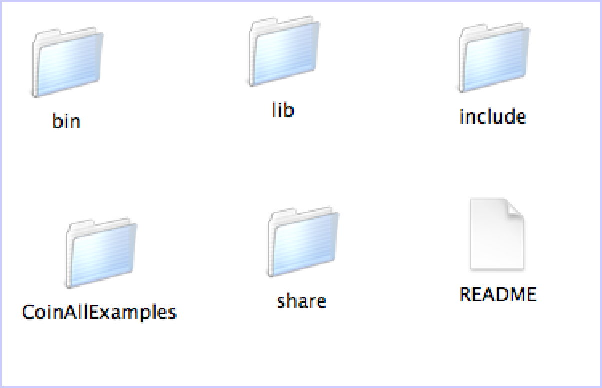
Solving a Problem

The Windows Visual Studio distribution has the following directory structure:



Solving a Problem

A Linux distribution has the following directory structure:


bin
lib
include
CoinAllExamples
share
README

Solving a Problem

We use the **OSSolverService** executable to:

- Solve a model instance using a solver that is included in your distribution. It takes three (four) formats
  - AMPL `nl` format
  - the defacto standard `mps` format (does not handle nonlinear)
  - the new OS XML `OSiL` format (does handle nonlinear)
  - GAMS `gms` format through GAMSlinks
- Use OSSolverService with the OS protocols to communicate with a remote solver service

Solving a Problem

The **bin** directory contains optimization solver executables.

You can use any of the solver executables individually or you can use **OSSolverService.exe** (which is linked to the individual solvers).

Will will solve the linear program (Par Inc.):

$$
\begin{array}{rcl}
\mathrm{MAX} \quad 10 * X1 + 9 * X2 & & \\
.7 * X1 + X2 & \leq & 630 \\
.5 * X1 + (5/6) * X2 & \leq & 600 \\
X1 + (2/3) * X2 & \leq & 708 \\
.1 * X1 + .25 * X2 & \leq & 135 \\
X1, X2 & \geq & 0
\end{array}
$$

Solving a Problem

Solve a linear program: using the `OSSolverService` . At the command line, connect (**cd**) to the bin directory and execute the following:

To solve a problem in OSiL XML format

```
OSSolverService -osil ../../data/osilFiles/parincLinear.osil
```

To solve a problem in AMPL nl format

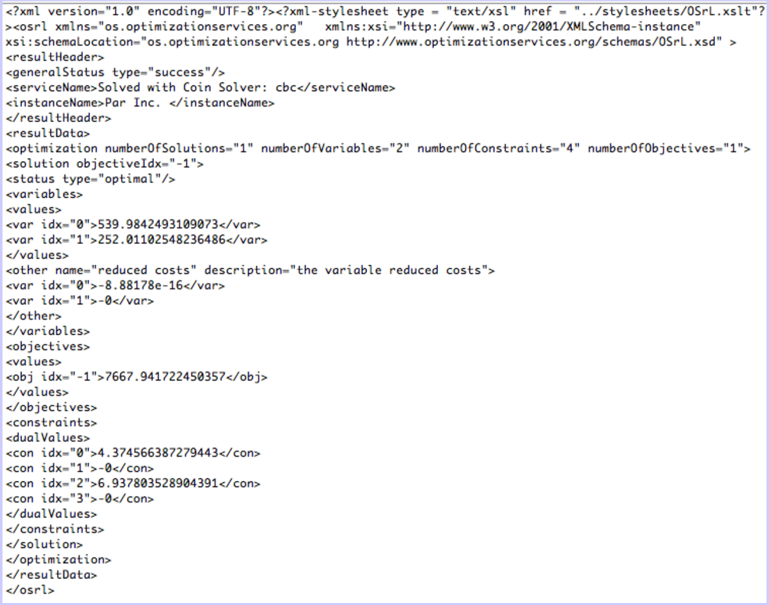
```
OSSolverService -nl ../../data/amplFiles/parinc.nl
```

To solve a problem in MPS format

```
OSSolverService -mps ../../data/mpsFiles/parinc.mps
```

Solving a Problem

The result is printed in XML format:



```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type = "text/xsl" href = "../stylesheets/OSrL.xslt"?
><osrl xmlns="os.optimizationservices.org"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="os.optimizationservices.org http://www.optimizationservices.org/schemas/OSrL.xsd" >
<resultHeader>
<generalStatus type="success"/>
<serviceName>Solved with Coin Solver: cbc</serviceName>
<instanceName>Par Inc. </instanceName>
</resultHeader>
<resultData>
<optimization numberOfSolutions="1" numberOfVariables="2" numberOfConstraints="4" numberOfObjectives="1">
<solution objectiveIdx="-1">
<status type="optimal"/>
<variables>
<values>
<var idx="0">539.9842493109073</var>
<var idx="1">252.01102548236486</var>
</values>
<other name="reduced costs" description="the variable reduced costs">
<var idx="0">-8.88178e-16</var>
<var idx="1">-0</var>
</other>
</variables>
<objectives>
<values>
<obj idx="-1">7667.941722450357</obj>
</values>
</objectives>
<constraints>
<dualValues>
<con idx="0">4.374566387279443</con>
<con idx="1">-0</con>
<con idx="2">6.937803528904391</con>
<con idx="3">-0</con>
</dualValues>
</constraints>
</solution>
</optimization>
</resultData>
</osrl>
```

Solving a Problem

More detail – variables values

```
<values>
    <var idx="0">539.9842493109073</var>
    <var idx="1">252.01102548236486</var>
</values>
```

The objective function value

```
<objectives>
<values>
<obj idx="-1">7667.941722450357</obj>
</values>
</objectives>
```

Solving a Problem

You can also print the result to a file.

Use the osrl option

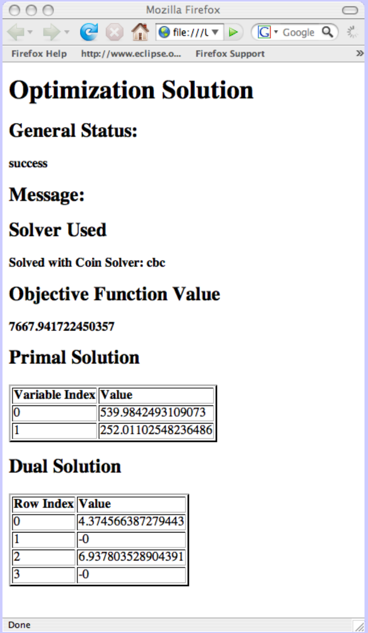
```
OSSolverService -osil ../../data/osilFiles/parincLinear.osil
    -osrl result.xml
```

You can display the result in a browser using XSLT.

Copy **data/stylesheets** into the root of the CoinAll distribution.

Open in your browser

Solving a Problem



Optimization Solution

General Status:

success

Message:

Solver Used

Solved with Coin Solver: cbc

Objective Function Value

7667.941722450357

Primal Solution

Variable Index	Value
0	539.9842493109073
1	252.01102548236486

Dual Solution

Row Index	Value
0	4.374566387279443
1	-0
2	6.937803528904391
3	-0

Solving a Problem

To solve a **linear program** set the solver options to:

- clp
- dylp

To solve a **mixed-integer linear program** set the solver options to:

- cbc
- symphony

To solve a **continuous nonlinear program** set the solver options to:

- ipopt

To solve a **mixed-integer nonlinear program** set the solver options to:

- bonmin

Solving a Problem

Solving a **linear integer** program:

```
OSSolverService -osil ../../data/osilFiles/p0033.osil
      -solver cbc
```

Solving a **nonlinear** optimization problem

```
OSSolverService -osil ../../data/osilFiles/rosenbrockmod.osil
      -solver ipopt
```

Solving a **mixed-integer nonlinear** optimization problem

```
OSSolverService -osil ../../data/osilFiles/bonminEx1.osil
      -solver bonmin
```

Solving a Problem

It is possible to build the OSSolverService to work with other solvers but they are not included due to licensing issues.

- Glpk
- Cplex
- LINDO

Solving a Problem

Continually writing out command line options a pain. Use a `configuration` file instead. Do something like:

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

where the file `testLocal.config` is

```
-osil ../../data/osilFiles/parincLinear.osil
-solver cbc
```

Note: the only option that is required is the location of an instance file

Solving a Problem

Finally – you can call solvers remotely.

Specify a **service location** of the remote solver service.

```
OSSolverService -osil ../data/osilFiles/parincLinear.osil
  -serviceLocation
     http://gsbkip.chicagogsb.edu/os/OSSolverService.jws
```

Solving a Problem

To get help

```
OSSolverService -h
```

–OR–

```
OSSolverService --help
```

Using a Modeling Language

Where are we?

Well, we can solve an optimization problem, assuming we have it in OSiL, nl, gams, or MPS format. **Where does the OSiL, nl, gams, or MPS file come from**? Ugh!

There are two general ways to create the optimization instance.

- Use a modeling language!!! The AMPL, GAMS, and MPL solvers work with COIN-OR solvers. You can also use FlopC++ (but it requires a compiler and knowledge of C++).
- Build the instance using COIN-OR libraries.

USING OSAmplClient

In the CD distribution again locate the bin directory. The bin directory contains an application **OSAmplClient**.

Inside of AMPL you "declare" OSAmplClient to be solver. It takes care of the rest.

There is not a size limitation on the solvers, however the AMPL you download off is constrained in size. I think the "free" version is about 300 constraints and variables.

USING OSAmplClient

Use the following sequence to solve a problem from AMPL. We assume that the AMPL model is **hs71.mod.**

Execute **ampl.exe** at the command line. Once inside **ampl.exe** do the following:

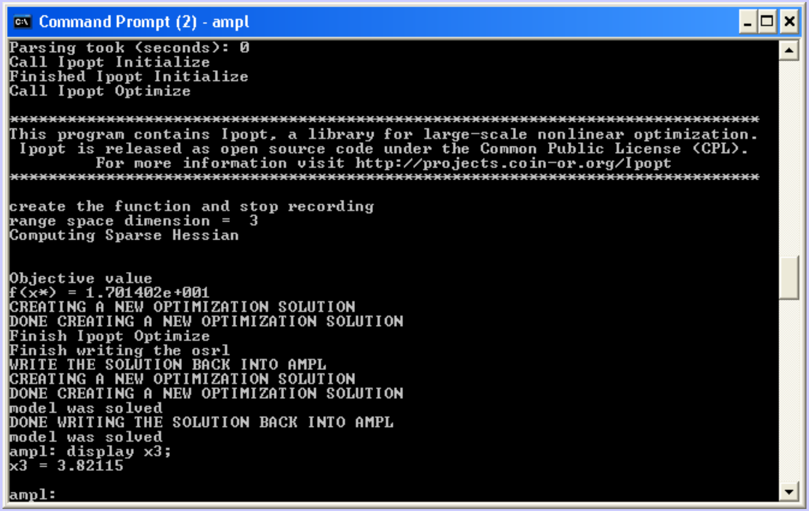
```
model hs71.mod;
# tell AMPL that the solver is OSAmplClient
option solver OSAmplClient;

# now tell OSAmplClient to use Ipopt
option OSAmplClient_options "solver ipopt";

# now solve the problem
solve;
```

USING OSAmplClient

AMPL result of **solve:**



```
Parsing took (seconds): 0
Call Ipopt Initialize
Finished Ipopt Initialize
Call Ipopt Optimize

******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Common Public License (CPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

create the function and stop recording
range space dimension =  3
Computing Sparse Hessian


Objective value
f(x*) = 1.701402e+001
CREATING A NEW OPTIMIZATION SOLUTION
DONE CREATING A NEW OPTIMIZATION SOLUTION
Finish Ipopt Optimize
Finish writing the osrl
WRITE THE SOLUTION BACK INTO AMPL
CREATING A NEW OPTIMIZATION SOLUTION
DONE CREATING A NEW OPTIMIZATION SOLUTION
model was solved
DONE WRITING THE SOLUTION BACK INTO AMPL
model was solved
ampl: display x3;
x3 = 3.82115

ampl:
```

USING OSAmplClient

You can also call a remote (i.e. over the network) solver from inside AMPL.

In order to call a remote solver service, after the command

```
option OSAmplClient_options "solver ipopt";
```

set the solver `service` option to the address of the remote solver service.

```
option ipopt_options
  "service http://gsbkip.chicagogsb.edu/os/OSSolverService.jws";
```

USING GAMS

You can also use GAMS (General Algebraic Modeling System).
Used in a fashion very similar to AMPL.

However, you need to download the COIN-OR project **GAMSlinks.**

In the GAMSlinks folder select the platform you have.

Copy **gmsos_.zip** into the GAMS 22.8 directory.

Execute the command **gamsinst**.

USING GAMS

Inside the GAMS 22.8 directory execute the following command.

```
gams rbrockmod nlp=os
```

The option optfile=1 tells GAMS to read the file os.opt for additional options. The os.opt file is

```
writeosil osil.xml
writeosrl osrl.xml
solver ipopt
```

USING GAMS

To solve a problem remotely in GAMS add the **service** option to the option file:

```
writeosil osil.xml
writeosrl osrl.xml
service http://gsbkip.chicagogsb.edu/os/OSSolverService.jws
solver ipopt
```

USING MPL

You can use MPL in conjunction with CoinMP to call COIN-OR solvers.

See: `http://www.maximalsoftware.com/academic/`

Writing Applications

The `CoinAll` distribution provides libraries of COIN-OR projects that can be used to build applications.

You Can:

- Write code to call solvers as part of a larger application
- Use the APIs (application program interface) to build models
- Use the APIs to build customized solution procedures (e.g. cutting planes, column generation, etc.)

Writing Applications

The example problems are in the **CoinAllExamples** or **MSVisualStudioCoinAllExamples**.

addCuts– illustrates how to add cuts to a branch-and-cut algorithm

CppAD – illustrates how to use the CppAD project – Brad Bell

instanceGenerator – illustrates how to use the libraries to generate a nonlinear programming problem

osTestCode — illustrates how to use the libraries to generate a linear program and call a solver

template – an empty project (set up to link to all the libs) that you can use for your own applications

Writing Applications

Windows – Visual Studio project files configured to link with the libraries in the **lib** directory and include the header files in the **include** directory.

Linux and Mac OS X – Makefiles configured to link with the libraries in the **lib** directory and include the header files in the **include** directory.

First, we just call a solver from an application to see if everything is working. We illustrate with the **osTestCode** example in **CoinAllExamples**.

Writing Applications

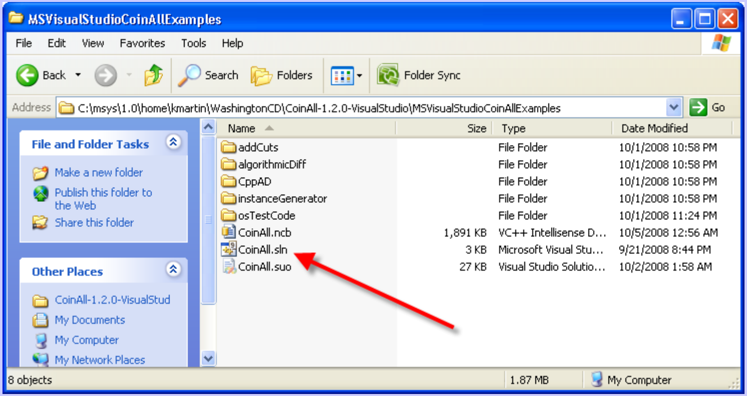
First, we just call a solver from an application to see if everything is working. We illustrate with the **osTestCode** example in **CoinAllExamples**.

The **osTestCode** does the following:

- Builds an in-memory instance of the linear programming example we solved earlier.
- Creates a Clp solver object (more on this later)
- Optimizes the problem with the Clp solver object

Writing Applications

Test with Windows. Locate the solution file in **MSVisualStudioCoinAllExamples**.



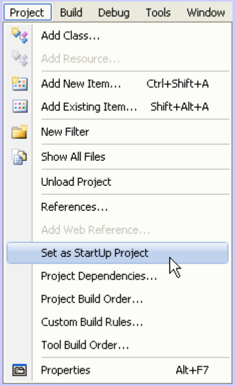
C:\msys\1.0\home\kmartin\WashingtonCD\CoinAll-1.2.0-VisualStudio\MSVisualStudioCoinAllExamples

Name	Size	Type	Date Modified
addCuts		File Folder	10/1/2008 10:58 PM
algorithmicDiff		File Folder	10/1/2008 10:58 PM
CppAD		File Folder	10/1/2008 10:58 PM
instanceGenerator		File Folder	10/1/2008 10:58 PM
osTestCode		File Folder	10/1/2008 11:24 PM
CoinAll.ncb	1,891 KB	VC++ Intellisense D...	10/5/2008 12:56 AM
CoinAll.sln	3 KB	Microsoft Visual Stu...	9/21/2008 8:44 PM
CoinAll.suo	27 KB	Visual Studio Solutio...	10/2/2008 1:58 AM

8 objects 1.87 MB

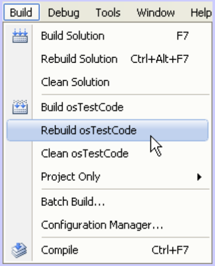
Writing Applications

Set the **osTestCode** project as the default project.



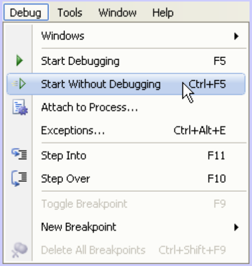
Writing Applications

Rebuild the **osTestCode** project.



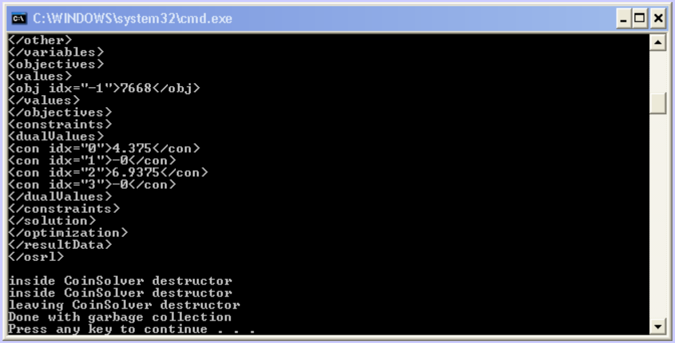
Writing Applications

Run the **osTestCode** project.



Writing Applications

The result of running the **osTestCode** project.



```
</other>
</variables>
<objectives>
<values>
<obj idx="-1">7668</obj>
</values>
</objectives>
<constraints>
<dualValues>
<con idx="0">4.375</con>
<con idx="1">-0</con>
<con idx="2">6.9375</con>
<con idx="3">-0</con>
</dualValues>
</constraints>
</solution>
</optimization>
</resultData>
</osrl>

inside CoinSolver destructor
inside CoinSolver destructor
leaving CoinSolver destructor
Done with garbage collection
Press any key to continue . . .
```

Writing Applications

Test in Linux and Mac OS X

Step 1: Connect to the directory **CoinAllExamples/osTestCode**

Step 2: type **Make**

Step 3: type **./osTestCode**

Writing Applications – Call a Solver

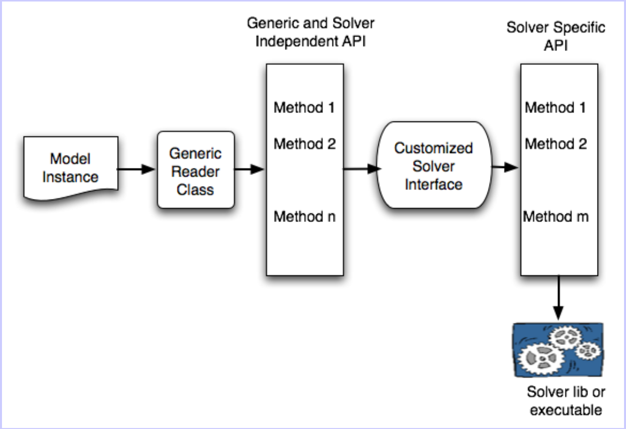
The BIG PICTURE: most optimization solvers have an API (application program interface).

You can use the solver API to create the solver-specific representation of an optimization instance.

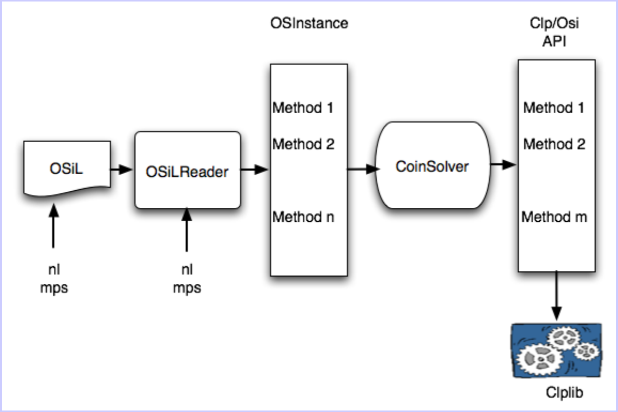
The OS library *wraps around* the solver-specific APIs, takes a generic non-solver-specific model instance, and converts it into the solver-specific representation.

Writing Applications – Call a Solver

An illustration!



Writing Applications – Call the Clp Solver



Writing Applications – Call a Solver

Basic Idea for calling a solver: the OS library contains **solver classes**.

- **Step 1:** Create a specific solver object
- **Step 2:** Feed the solver object a generic solver-independent model instance
- **Step 3:** Build the solver-specific model instance (customized for each solver API (Application Program interface))
- **Step 4:** Solve the problem

Call a Solver (with Code)

Step 1: Create a Clp solver object

```
CoinSolver *solver = new CoinSolver();
solver->sSolverName ="clp";
```

Step 2: Feed the solver object a generic solver-indepenent model instance

```
solver->osinstance = osinstance;
```

Step 3: Build the Clp-specific model instance

```
solver->buildSolverInstance();
```

Step 4: Solve the problem using Clp

```
solver->solve();
```

Writing Applications – Call a Solver

Now change to **Ipopt** solver.

Change:

```
CoinSolver *solver = new CoinSolver();
solver->sSolverName ="clp";
```

To:

```
IpoptSolver *solver = new IpoptSolver();
//solver->sSolverName ="clp";
```

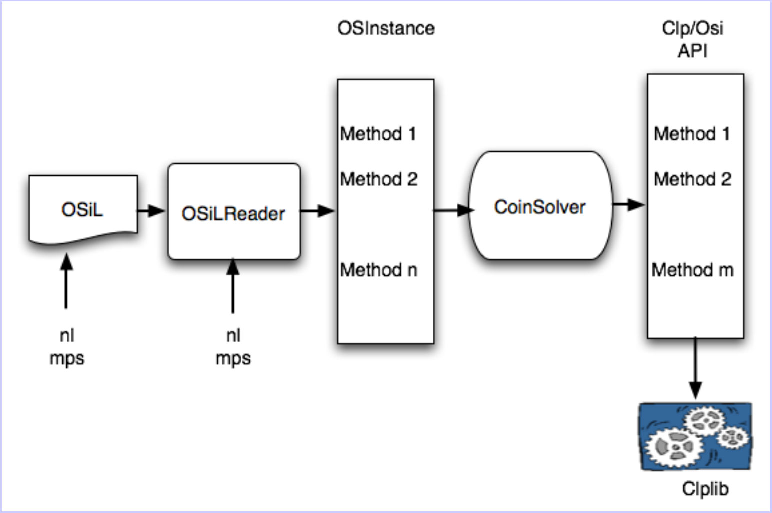
Writing Applications – Call a Solver

The following solver classes are available.

- OSBonMinSolver
- OSCoinSolver (based on OSI interface)
  - Clp
  - Cbc
  - Cplex (not included with CD build)
  - DyLP
  - Glpk (not included with CD build)
  - SYMPHONY
- OSIpoptSolver
- OSLindoSolver (not included with CD build)

Writing Applications – Build an Instance

You can either: 1) read an instance from a file or string; or 2) create the instance in-memory.



Writing Applications – Build an Instance

The **OSInstance** class provides an API for connecting a model instance with a solver.

- **set() and add()** methods for creating models
- **get()** methods for getting information about a problem
- **calculate()** methods for finding gradient and Hessians using algorithmic differentiation

Writing Applications – Build an Instance

Let's create a problem from scratch.

The COIN-OR software is object oriented.

Key Idea 1: Create an **OSInstance** object.

```
OSInstance *osinstance =  new OSInstance();
```

Put some variables in

```
osinstance->setVariableNumber( 2);

osinstance->addVariable(0, "x0", 0, OSDBL_MAX, 'C', OSNAN, "");
osinstance->addVariable(1, "x1", 0, OSDBL_MAX, 'C', OSNAN, "");
```

Writing Applications – Build an Instance

Don't forget the constraints!

```
osinstance->setConstraintNumber( 4);

osinstance->addConstraint(0, "row0", -OSDBL_MAX, 630, 0);
osinstance->addConstraint(1, "row1", -OSDBL_MAX, 600, 0);
osinstance->addConstraint(2, "row2", -OSDBL_MAX, 708, 0);
osinstance->addConstraint(3, "row3", -OSDBL_MAX, 135, 0);
```

In practice, you would generate the constraints inside a loop(s) rather than enumerating them one-by-one.

Writing Applications – Build an Instance

There are similar constructs for:

- the objective function
- constraints with all linear terms
- quadratic constraints
- constraints with general nonlinear terms

Writing Applications – Build an Instance

Exercise: changing a constraint upper bound.

It is possible to access the OSInstance object directly.

Change the first constraint's upper bound from 708 to 500.

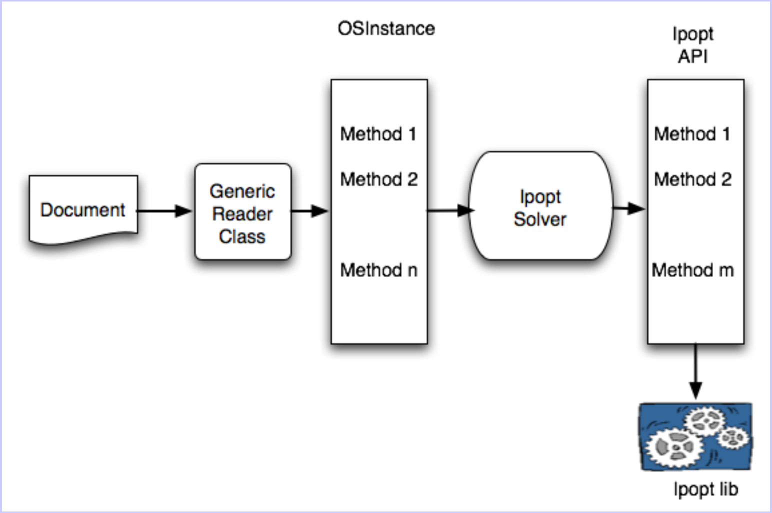
```
osinstance->instanceData->constraints->con[0]->ub = 500;
```

See **examples/osTestCode** for the details of this example.

See **examples/instanceGenerator** for an example of building a nonlinear model.

Writing Applications – Build an Instance

Deja vu all over again – where we are:



Writing Applications – Build an Instance

Nonlinear solvers – some nonlinear solvers, e.g. **Bonmin** or **Ipopt**, often want function and derivative evaluations rather than an explicit formulation.

The OS library has **calculate()** methods for function and derivative information.

For example, the Ipopt API has a method **eval_jac_g(...)** that requires the Jacobian of the constraint matrix. Just have this method call

```
osinstance->calculateAllConstraintFunctionGradients(...)
```

Writing Applications – Build an Instance

Some nonlinear solvers, e.g. LINDO, want a problem in an expression list format.

Not a problem!

The OS library has methods that will return the problem as postfix or prefix expression lists.

Writing Applications – Add Cutting Planes

The example for this is **addCuts**. It illustrates using the **Cgl** (Constraint generation library) project.

The example problem illustrates:

Step 1: Read a problem and create an internal generic instance.

Step 2: Create a solver object.

Step 3: Solve the LP relaxation.

Step 4: Add cut generators.

Step 5: Go into branch-and-bound

Writing Applications – Add Cutting Planes

Here we go with code:

Create a **Clp** solver object and give it the instance

```
CoinSolver *solver  = NULL;
solver = new CoinSolver();
solver->sSolverName ="clp";
solver->osinstance = osinstance;
```

Build the solver-specific instance and solve.

```
solver->buildSolverInstance();
solver->osiSolver->initialSolve();
```

Writing Applications – Add Cutting Planes

Create the cut generators and a **Cbc** model

```
CglKnapsackCover cover;
CglSimpleRounding round;
CglGomory gomory;
CbcModel *model = new CbcModel( *solver->osiSolver);
```

add the cutgenerators

```
model->addCutGenerator(&gomory, 1, "Gomory");
model->addCutGenerator(&cover, 1, "Cover");
model->addCutGenerator(&round, 1, "Round");
```

solve

```
model->branchAndBound();
```

Getting Help

Try https://projects.coin-or.org/OS

See the 88 page User's Manual

http://www.coin-or.org/OS/doc/osUsersManual_1.1.pdf