2012

OptiRisk SYSTEMS

Author: Christian Valente

[AMPLNET]

Object library description

This document is intended to provide information for users of the AMPLNET library.

Original Version	01/03/2012
Last Revision [2]	02/03/2012

AMPLNET

OBJECT LIBRARY DESCRIPTION

TABLE OF CONTENTS

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK

Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095

E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

1. INTRODUCTION

1.1 WHAT IS AMPLNET

AMPLNET is a .NET assembly which allows developers to access the features of the AMPL interpreter from within a development environment. All model generation and solver interaction is handled directly by AMPL, which leads to great stability and speed; the library just acts as an intermediary, and the added overhead depends mostly on how much data is read back from AMPL. The size of the model as such is irrelevant. Functions for assigning data to AMPL parameters are provided, to give the programmer more control on the read data.

1.2 WHO CAN USE AMPLNET

The intended user of the library is the developer who needs to connect an application to optimisation models and solvers, or the analyst with some experience in programming who wants to build a quick proof-of-concept application.

1.3 SYSTEM REQUIREMENTS

As AMPLNET uses AMPL for model generation, the AMPL executable is needed and is generally provided. The solution is then obtained through external solvers. In general, a valid AMPL setup is necessary and sufficient condition for the correct execution of AMPLNET.

.NET framework 4.0 is required, although the functionalities used are fully implemented by Mono.NET. Testing on such framework will be started soon.

1.4 ABOUT THIS MANUAL

This document intends to guide a developer in the process of implementing an “AMPLNET based” application, so far the presentation of all the classes comprising the library is out of its scope.

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK

Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095

E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

2. GETTING STARTED

2.1 INSTALLATION

AMPLNET is distributed in various modules.

- AMPLNET.DLL is the main assembly, contains all the implementation of the object library
- AMPL.EXE, AMPLTABL.DLL, OPTIRISKDLL.DLL, AFORTMP.EXE are a complete AMPL execution environment
- LICENCE.KEY is the license file needed for licensing both AMPLNET and the included AMPL.

The easiest deployment strategy is to copy all the files to the resulting application directory. In case the target machine has AMPL installed already, the files at point 2 *can* be omitted, provided that AMPL.EXE is accessible from any location in the machine (aka its folder is specified in the PATH environment variable) or that the environment AMPLPATH is specified and points to the location of AMPL.EXE.

For development, the only file to be referenced is AMPLNET.DLL. That contains all needed classes and functions.

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK

Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095

E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

3. TUTORIALS

This section shows examples of using AMPLNET in C#. Where screenshots are displayed, they refer to Visual Studio 2010. A free version (Visual Studio Express) is available for download at Microsoft website. (ADD REFERENCE)

3.1 EXAMPLE 1: HELLO WORLD FROM AMPL

This tutorial will illustrate the steps necessary to setup a machine for AMPLNET-based development, from creating the Visual Studio project to the first communication with AMPL.

3.1.1 CREATING A NEW SOLUTION AND PROJECT

Open Visual Studio 2010 (or Visual Studio Express) and create a new empty solution. This will automatically create a directory with the solution name.

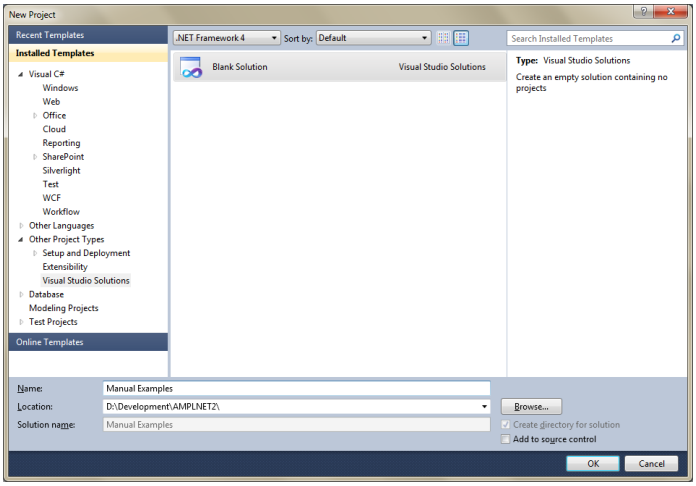

Figure 1 Creating a new solution

Create a new C# console application project:

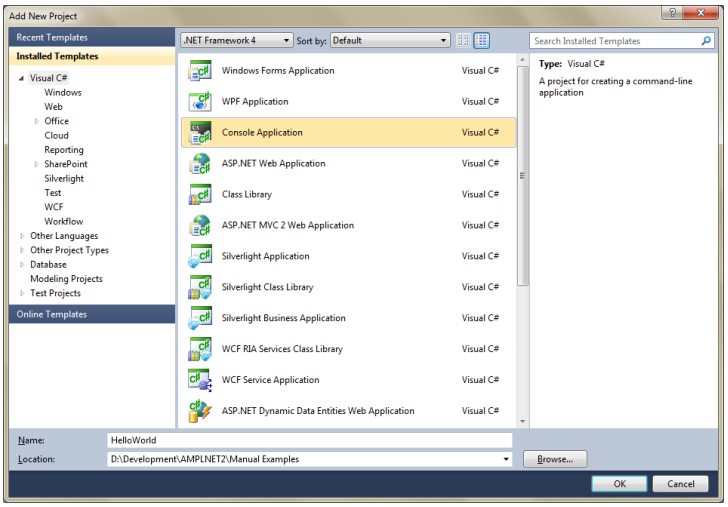

Figure 2 Creating a new project

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK
Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095
E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

3.1.2 OPTIONAL STEPS, JUST FOR MACHINES WITH NO AMPL INSTALLATION

If developing on a machine with no existing and working AMPL installation, and do not wish to do so, follow the following steps to enable AMPLNET-based development.

Copy the AMPLNET directory in the solution directory then right click on the project name in Visual Studio, click on `Add existing items`, and choose to add all the files in AMPLNET folder **with the exception of AMPLNET.DLL**.

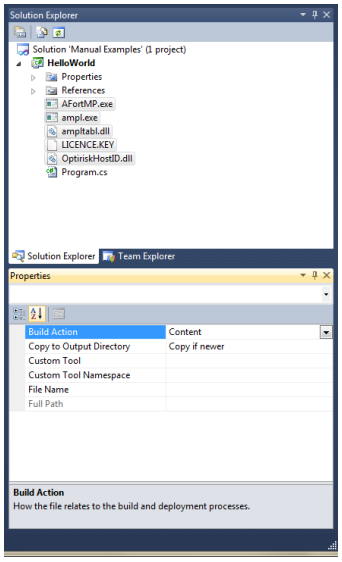

Figure 3 Configure for deployment

Then select the files in the Solution explorer, right click and choose `Properties` in the context menu.

As shown in Figure 3, configure the properties of all the AMPLNET files to:

- **`Build Actions`** `Content`
- **`Copy to output directory`** `Copy if newer`

This will ensure that the AMPL environment files are deployed automatically to the project execution folder when needed.

If the machine is to be used consistently for AMPLNET development, the author advises to install AMPL in a directory specified in the PATH environment variable, so that all these steps can be skipped – and the resulting projects will be cleaner.

3.1.3 ADD A REFERENCE TO THE AMPLNET ASSEMBLY

Right click on the references folder of the project, click on `Add new reference` and add AMPLNET.DLL as shown in **Error! Reference source not found. Error! Reference source not found.**.

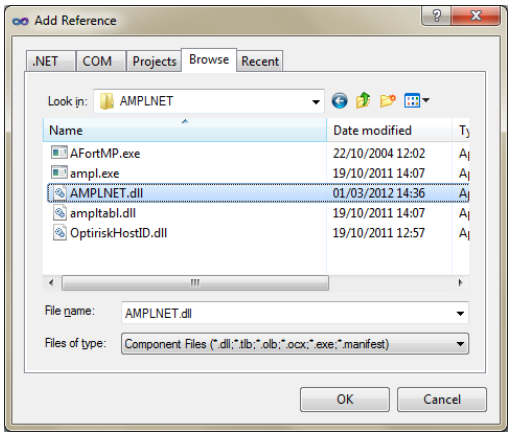

Figure 4 Add reference

This will give access to all the types and methods defined in the library. Also, Visual Studio will automatically copy the assembly to the deployed application folder. The reference AMPLNET should be now visible in the folder References. Note that AMPLNET is compiled with `AnyCPU` flag therefore, under windows, it will run as 32 bits on 32 machines or if called by 32 bits assemblies or as 64 bits otherwise. This is completely unrelated to the architecture of the underlying AMPL.EXE and relative solvers.

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK

Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095

E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

3.1.4 C# CODE

The following source code, copied in the main function of the project, will create an instance of the AMPLNET object and use AMPL's printing routines to display a "hello world" string. The resulting string is then returned to C#. The only two statements related to AMPLNET are the creation of the Ampl instance (class `AMPL.Lang.AMPL`) and the command `interpretString`, which sends a string for interpretation to AMPL and gets the string output back.

```
string m; // will store the string output of AMPL

AMPL.Lang.AMPL Ampl = new AMPL.Lang.AMPL(new AMPL.Environment());
Ampl.interpretString("Teststring", "printf \"Hello world from AMPL!\";", out m);

Console.WriteLine(m);
```

3.2 EXAMPLE 2: INPUT AND SOLVE A MODEL

This example will show how to instruct AMPL to load a model and data stored in external (AMPL) files, solve the model and get the solution vectors and objective value.

After creating a new project, repeat steps 3.1.2 and 3.1.3.

Add the model and data files (diet.mod and diet.dat) to the project, so that they will be automatically deployed in the destination directory.

Type in the following source code:

```
// Create ampl instance
AMPL.Lang.AMPL Ampl = new AMPL.Lang.AMPL(new AMPL.Environment());
// Load model and data files, current directory assumed
Ampl.interpretFile("diet.mod");
Ampl.interpretFile("diet.dat");
// Solve the model
if (Ampl.Commands.solve() != SolutionStatus.Solved)
{
	Console.WriteLine("Problems while solving the file.");
	return;
}
// Display objective formulation and value
ObjectiveMap o = Ampl.getObjective("total_cost");
Console.WriteLine(string.Format("Objective {0}", o.Declaration));
Console.WriteLine(string.Format("Value: {0}", o.get().Value));


// Display variable instances values
VariableMap Buy = Ampl.getVariable("Buy");
Console.WriteLine("Solution vector, variable Buy");
foreach (Variable instance in Buy)
	Console.WriteLine(instance.ToString());
```

It can be noted that we access an ObjectiveMap and a VariableMap to interrogate AMPLNET about the objective function and a part of the solution vector. These are collections of objectives and vari-

OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK
Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095
E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/

ables. To access the single instance, the function get() is used in case of the objective, which gets the only instance of the objective, and the iterator is used to access all instances of the variable Buy.

The output of the program above is:

```
Objective minimize total_cost: sum{j in FOOD} cost[j]*Buy[j];
Value: 118.059403239557
Solution vector, variable Buy
[BEEF] Value: 5.3606  Dual: 0
[CHK]  Value: 2       Dual: 0
[FISH] Value: 2       Dual: 0
[HAM]  Value: 10      Dual: 0
[MCH]  Value: 10      Dual: 0
[MTL]  Value: 10      Dual: 0
[SPG]  Value: 9.3061  Dual: 0
[TUR]  Value: 2       Dual: 0
```

The input data of an optimisation model is stored in its parameters; these can be scalar or vectorial entities. Two ways are provided to change the value of vectorial parameter: change specific values or change all values at once. The example shows an example of both ways, reassigning the values of the parameter costs firstly specifically, then altogether. Each time, it then solves the model and get the objective function. The function used to change the values is overloaded, and is in both cases `ParameterMap.let`.

```
// Reassign data - specific instances
ParameterMap cost = Ampl.getParameter("cost");
cost.let( new AMPL.Tuple[] { new AMPL.Tuple("BEEF"), new AMPL.Tuple("HAM") },
          new double[] { 5.01, 4.55 });
Console.WriteLine("Increased costs of beef and ham.");
// Resolve and display objective
Ampl.Commands.solve();
Console.WriteLine(string.Format("New objective value: {0}", o.get().Value));

// Reassign data - all instances
cost.let(new double[] {3, 5, 5, 6, 1, 2, 5.01, 4.55 });
Console.WriteLine("Updated all costs");
// Resolve and display objective
Ampl.Commands.solve();
Console.WriteLine(string.Format("New objective value: {0}", o.get().Value));
```

Adding the statements above adds the following to the output of Example2:

```
Increased costs of beef and ham.
New objective value: 144.415720375107
Updated all costs
New objective value: 164.54375
```


OPTIRISK R&D HOUSE, One Oxford Road Uxbridge, Middlesex, UB9 4DA, UK

Tel.: +44 (0) 1895 819 483/488, Fax: +44 (0) 1895 813 095

E-mail: optirisk@optirisk-systems.com URL: http://www.optirisk-systems.com/