



# Object Oriented GAMS API: Java, Python and .NET

Clemens Westphal

[cwestphal@gams.com](mailto:cwestphal@gams.com)

GAMS Software GmbH

GAMS Development Corporation

[www.gams.com](http://www.gams.com)



GAMS



Rotterdam, September 5, 2013

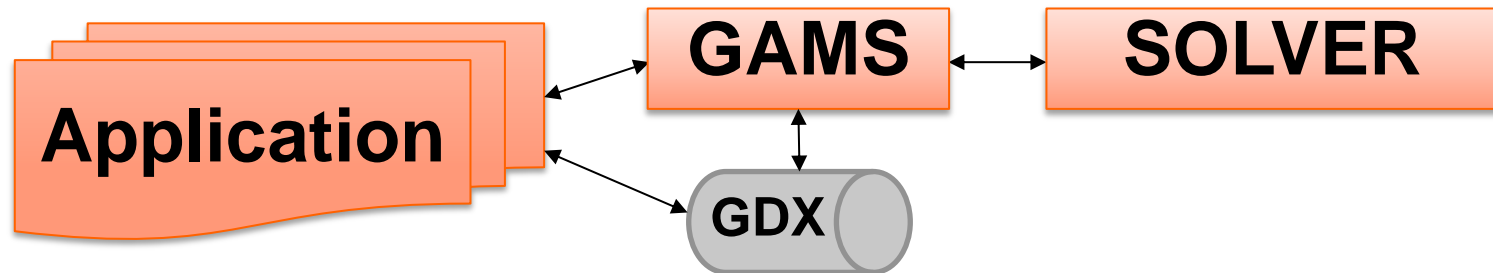


# Outline

- **Introduction**
- **Seamless Integration**
- **Small example in C#, Java and Python**
- **Scenario Solving**
- **Simple Encapsulation of a GAMS Model**



# Calling GAMS from your Application



## Creating Input for GAMS Model

→ Data handling using GDX API

## Callout to GAMS

→ GAMS option settings using Option API

→ Starting GAMS using GAMS API

## Reading Solution from GAMS Model

→ Data handling using GDX API



## Low level APIs → object oriented API

- Low level APIs
  - GDX, OPT, GAMSX, GMO, ...
  - High performance and flexibility
  - Automatically generated imperative APIs for several languages (C, Delphi, Java, Python, C#, ...)
- Object-oriented GAMS API
  - Additional layer on top of the low level APIs
  - Object-oriented
  - Written by hand to meet the specific requirements of different object-oriented languages



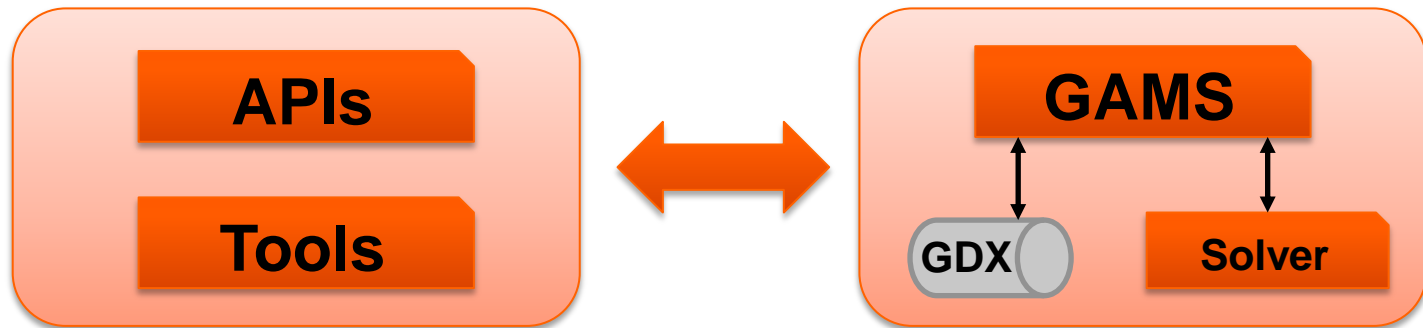
## Features of the object oriented API

- No modeling capability. Model is still written in GAMS
- Prepare input data and retrieve results in a convenient way → *GAMSDatabase*
- Control GAMS execution → *GAMSJob*
- Seamless integration of GAMS into other programming environments
- Scenario Solving: Feature to solve multiple very similar models in a dynamic and efficient way.  
→ *GAMSModelInstance*



# Seamless Integration

- GAMS concept: Separation of tasks



- Use GAMS for modeling and optimization tasks
- Programming languages like C# (.NET), Java and Python are well-suited for developing applications (GUI, Web, ...)
  - frameworks available for a wide range of specific task:
    - GUI and Web development, ...
- The object oriented GAMS API provides a convenient link to run GAMS in such environments



# Seamless Integration

- Example: Small transport Desktop application written in C#
- Convenient data preparation
- Representation of the results in a predefined way
- Modeling details are hidden from the user

Transport

Data

Load Data

distance capacity / demand

plants	market	distance
Seattle	New-York	2.5
Seattle	Chicago	1.7
Seattle	Topeka	1.8
San-Diego	New-York	2.5
San-Diego	Chicago	1.8
San-Diego	Topeka	1.4

bpar: min max steps

0.2	1	0.05
-----	---	------

Solve

Results

table chart

bmult	ModelStatus	SolverStatus	Objective
-------	-------------	--------------	-----------



# Small Example - C#

## Transport.cs

```
using System;
using GAMS;

namespace TransportSeq
{
    class Transport1
    {
        static void Main(string[] args)
        {
            GAMSWorkspace ws = new GAMSWorkspace();
            GAMSJob t1 = ws.AddJobFromString(GetModelText());

            t1.Run();
            foreach (GAMSVariableRecord rec in t1.OutDB.GetVariable("x"))
            {
                Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "):");
                Console.WriteLine("    level=" + rec.Level);
                Console.WriteLine("    marginal=" + rec.Marginal);
            }
        }
    }
}
```





# Small Example - C#

```
static String GetModelText()  
{  
    String model = @"  
Sets  
    i    canning plants    / seattle, san-diego /  
    j    markets           / new-york, chicago, topeka / ;  
  
Parameters  
  
    a(i) capacity of plant i in cases  
        /    seattle      350  
           san-diego    600 /  
  
    b(j) demand at market j in cases  
        /    new-york     325  
           chicago       300  
           topeka        275 / ;  
  
    < . . . >  
  
Solve transport using lp minimizing z ;"  
  
    return model;  
}  
}
```



# Small Example - Java

## Transport.java

```
package TransportSeq;

import com.gams.api.*

class Transport1
{
    static void main(String[] args)
    {
        GAMSWorkspace ws = new GAMSWorkspace();

        GAMSJob t1 = ws.addJobFromString(getModelText());
        t1.run();

        for (GAMSVariableRecord rec : t1.OutDB().getVariable("x"))
        {
            System.out.println("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "):");
            System.out.println("    level    =" + rec.getLevel());
            System.out.println("    marginal =" + rec.getMarginal());
        }
    }
}
```



# Small Example - Python

## transport.py

```
from gams import *

if __name__ == "__main__":
    ws = GamsWorkspace()

    t1 = ws.add_job_from_string(get_model_text())
    t1.run()

    for rec in t1.out_db["x"]:
        print rec
```



# Scenario Solving - Loop

```
Loop (s ,  
      f = ff(s) ;  
      solve mymodel min z using lp ;  
      objrep(s) = z.l ;  
);
```

- Data exchange between solves possible
- Model rim can change
- Each solve needs to regenerate the model
- User updates GAMS Symbols instead of matrix coefficients



# Scenario Solving - GUSS

```
set dict / s.scenario.''  
      f.param      .ff  
      z.level      .objrep /  
solve mymodel min z using lp scenario dict;
```

- Save model generation and solver setup time
- Hot start (keep the model hot inside the solver and use solver's best update and restart mechanism)
- Apriori knowledge of all scenario data
- Model rim unchanged from scenario to scenario



# Scenario Solving - GAMSModelInstance

```
foreach (string s in scen)
{
    f.FirstRecord().Value = v[s];
    modelInstance.Solve();
    objrep[s] = z.FirstRecord().Level;
}
```

- Save model generation and solver setup time
- Hot start (keep the model hot inside the solver and use solver's best update and restart mechanism)
- Data exchange between solves possible
- Model rim unchanged from scenario to scenario



# GAMSModelInstance - Example

- *bmult* is one parameter of the model which gets modified before we solve the instance:

```
GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
bmult.AddRecord().Value = 1.0;
mi.Instantiate("transport us lp min z", opt, new GAMSMODIFIER(bmult));
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };

foreach (double b in bmultlist)
{
    bmult.FirstRecord().Value = b;
    mi.Solve();
    <...>
    Console.WriteLine(" Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
}
```



# GAMSModelInstance - Example

- Updating bounds of a variable:

```
GAMSVariable x = mi.SyncDB.AddVariable("x", 2, VarType.Positive, "");
GAMSPParameter xup = mi.SyncDB.AddParameter("xup", 2, "upper bound on x");
mi.Instantiate("transport us lp min z", modifiers: new GAMSM Modifier(x, UpdateAction.Upper, xup));

foreach (GAMSSetRecord i in t7.OutDB.GetSet("i"))
    foreach (GAMSSetRecord j in t7.OutDB.GetSet("j"))
    {
        xup.Clear();
        xup.AddRecord(i.Keys[0], j.Keys[0]).Value = 0;
        mi.Solve();
        <...>
        Console.WriteLine(" Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
    }
```



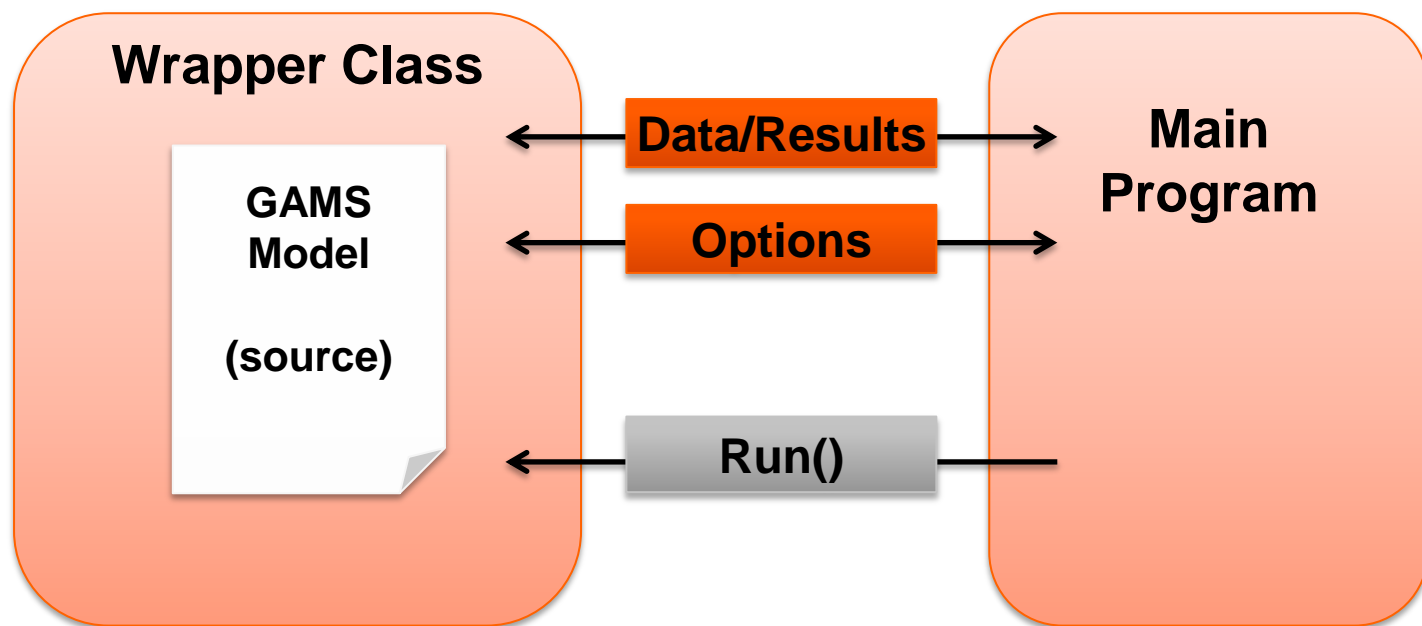


# Simple Encapsulation of a GAMS Model

- Wrapper class that encapsulates a single GAMS model
- Very simple interface:
  - Properties to communicate input data and results
  - Property to change options like the solver to use
  - Run() method to run the model
- Most implementation details are hidden in the wrapper class
- Highest level of abstraction (that is useful):
  - Data exchange, options, run()



# Simple Encapsulation of a GAMS Model





# Simple Encapsulation of a GAMS Model

...

```
GAMSWorkspace ws = new GAMSWorkspace();  
Transport t = new Transport(ws);
```

```
foreach (string p in plants)  
    t.i.AddRecord(p);  
foreach (string m in markets)  
    t.j.AddRecord(m);  
foreach (string p in plants)  
    t.a.AddRecord(p).Value = capacity[p];  
foreach (string m in markets)  
    t.b.AddRecord(m).Value = demand[m];
```

...

```
t.opt.AllModelTypes = "Gurobi";  
t.Run(output: Console.Out);  
Console.WriteLine("Objective: " + t.z.FirstRecord().Level);  
foreach (GAMSVariableRecord rec in t.x)  
    Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + "  
marginal=" + rec.Marginal);
```



# Summary

- Object Oriented API provides an additional abstraction layer on top of the low level GAMS APIs
- Powerful and convenient link to other programming languages
- .NET, Java and Python APIs are part of the current GAMS release available at [www.gams.com](http://www.gams.com). Many examples available:
  - Sequence of Transport examples
  - Cutstock, Warehouse, Benders Decomposition
- .NET/Mono and C++ under development



# Contacting GAMS

## Europe

**GAMS Software GmbH**  
**P.O. Box 40 59**  
**50216 Frechen**  
**Germany**

Phone: +49 221 949 9170  
Fax: +49 221 949 9171

[info@gams.de](mailto:info@gams.de)

## USA

**GAMS Development Corp.**  
**1217 Potomac Street, NW**  
**Washington, DC 20007**  
**USA**

Phone: +1 202 342 0180  
Fax: +1 202 342 0181

[sales@gams.com](mailto:sales@gams.com)  
[support@gams.com](mailto:support@gams.com)

<http://www.gams.com>