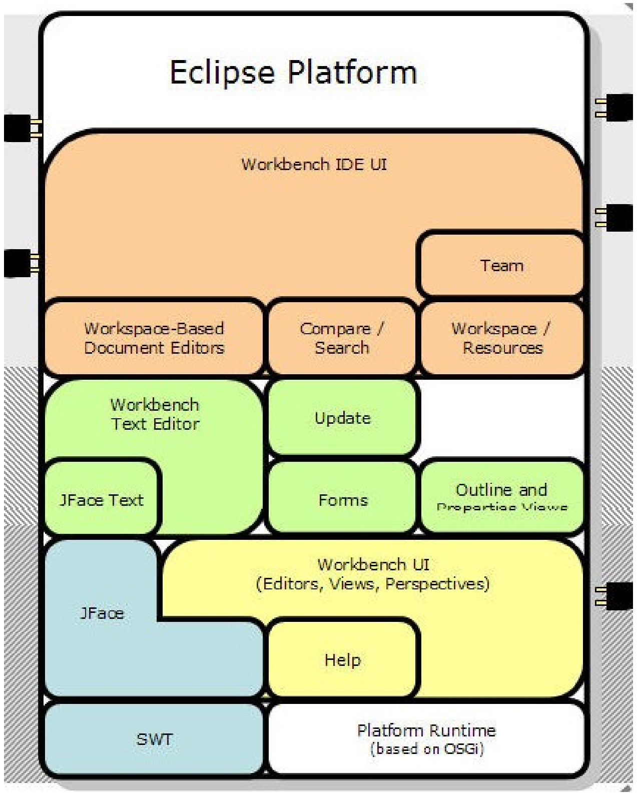
OpenModelica Eclipse Plugin and MetaModelica Exercises

Adrian Pop
adrian.pop@liu.se
PELAB/IDA/LIU, 2007-03-29

Last Updated 2011-09-15
by Peter Fritzson and Martin Sjölund

OpenModelica MDT – Eclipse Plugin

- Browsing of packages, classes, functions
- Automatic building of executables; separate compilation
- Syntax highlighting
- Code completion,
  Code query support for developers
- Automatic Indentation
- Debugger
  (Prel. version for algorithmic subset)


Eclipse Platform
Workbench IDE UI
Team
Workspace-Based Document Editors
Compare / Search
Workspace / Resources
Workbench Text Editor
Update
JFace Text
Forms
Outline and Properties Views
Workbench UI (Editors, Views, Perspectives)
JFace
Help
SWT
Platform Runtime (based on OSGi)

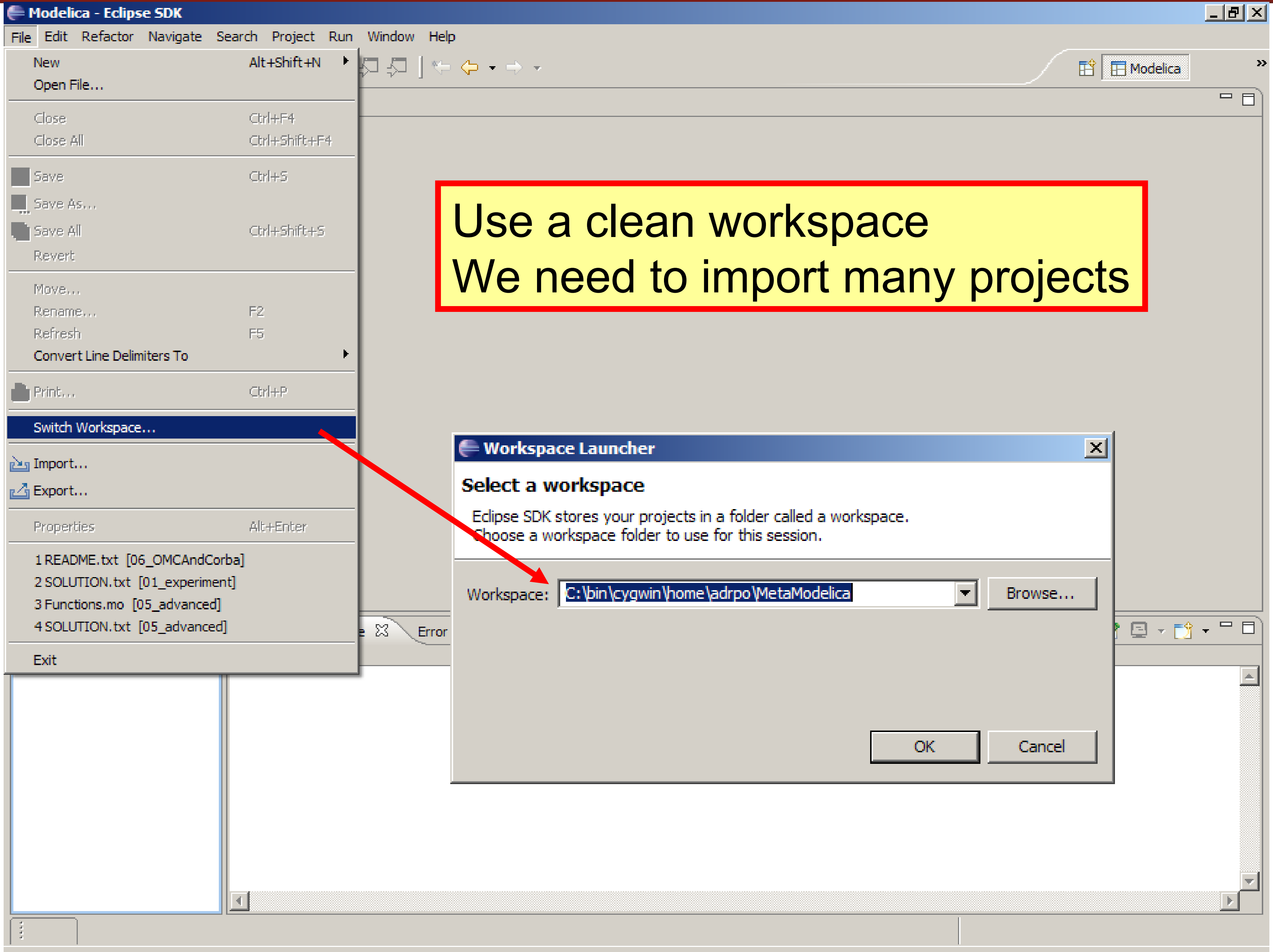
Outline

- Eclipse and Exercise Setup
- All Exercises
  - 00_simplesim
  - 01_experiment
  - 02a_exp1, 02b_exp2
  - 03_symbolicderivative
  - 04_assignment
  - 05a_assigntwotype
  - 05b_modassigntwotype
  - 06_advanced
  - 07_OMCAndCorba
  - 08-11 – as samples of Prog. Lang. modeling

Selected Exercises in this Course

- 00_simplesim
- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

Eclipse Setup – Switching Workspace



Use a clean workspace
We need to import many projects

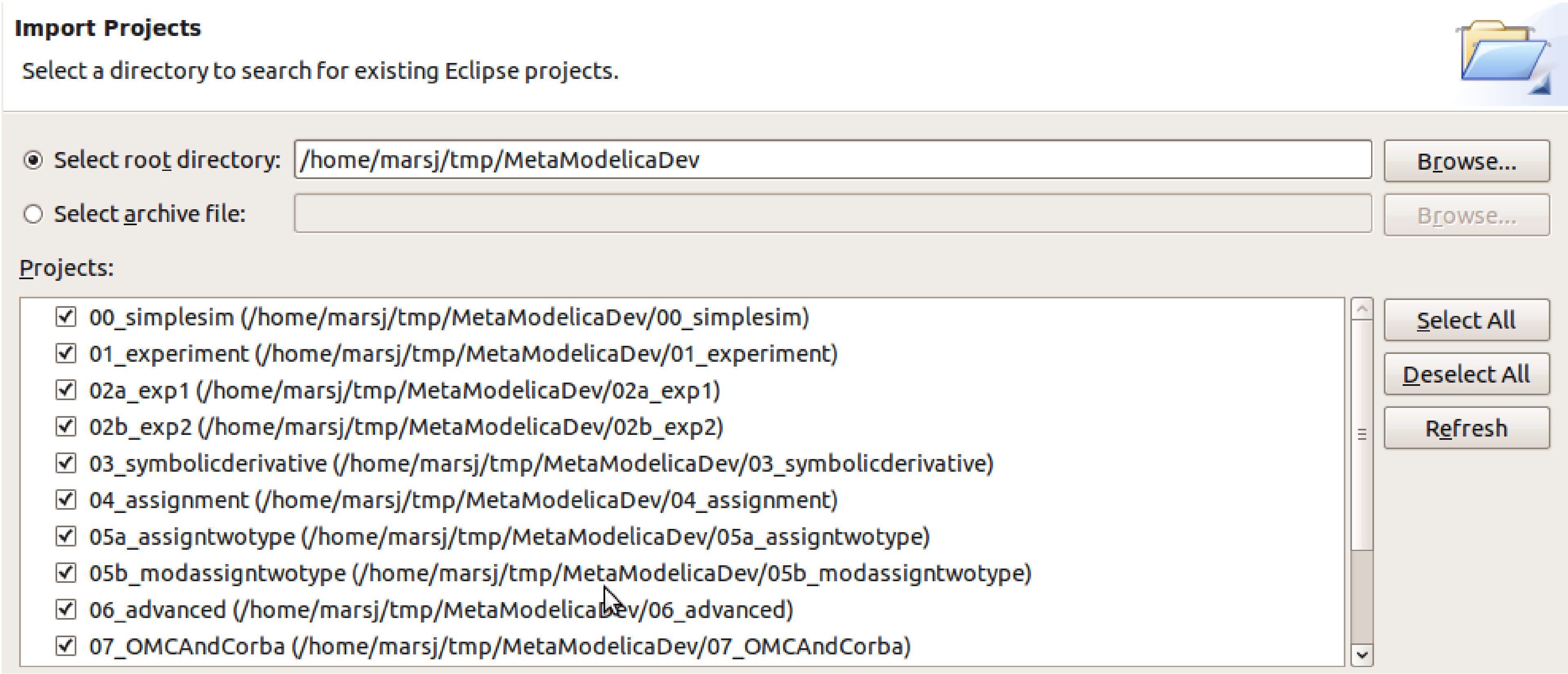
Select a workspace

Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\bin\cygwin\home\adrpo\MetaModelica

Import Projects

- File → Import
  - General → Existing projects into workspace



Import Projects

Select a directory to search for existing Eclipse projects.

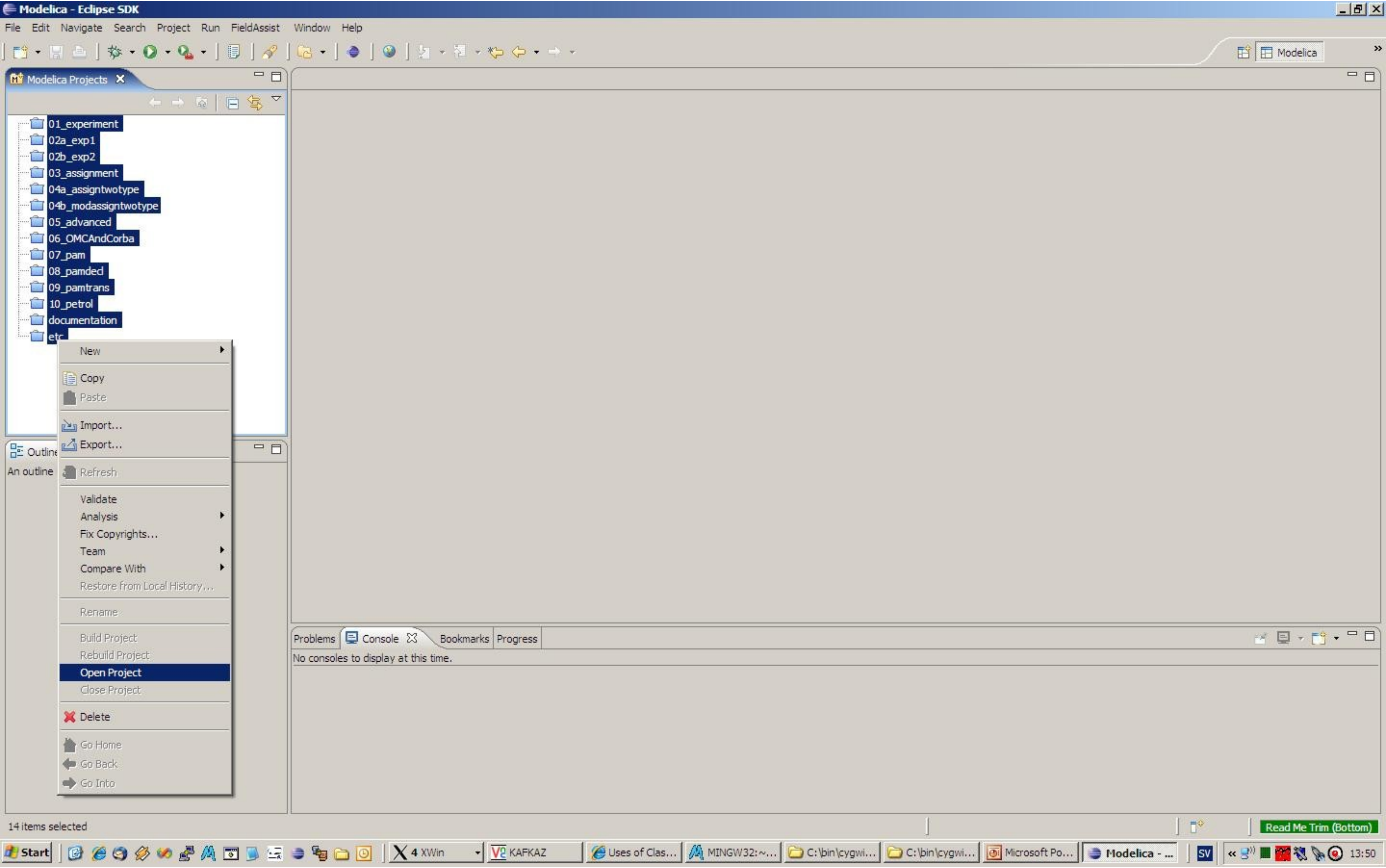
● Select root directory: /home/marsj/tmp/MetaModelicaDev

○ Select archive file:

Projects:

- ☑ 00_simplesim (/home/marsj/tmp/MetaModelicaDev/00_simplesim)
- ☑ 01_experiment (/home/marsj/tmp/MetaModelicaDev/01_experiment)
- ☑ 02a_exp1 (/home/marsj/tmp/MetaModelicaDev/02a_exp1)
- ☑ 02b_exp2 (/home/marsj/tmp/MetaModelicaDev/02b_exp2)
- ☑ 03_symbolicderivative (/home/marsj/tmp/MetaModelicaDev/03_symbolicderivative)
- ☑ 04_assignment (/home/marsj/tmp/MetaModelicaDev/04_assignment)
- ☑ 05a_assigntwotype (/home/marsj/tmp/MetaModelicaDev/05a_assigntwotype)
- ☑ 05b_modassigntwotype (/home/marsj/tmp/MetaModelicaDev/05b_modassigntwotype)
- ☑ 06_advanced (/home/marsj/tmp/MetaModelicaDev/06_advanced)
- ☑ 07_OMCAndCorba (/home/marsj/tmp/MetaModelicaDev/07_OMCAndCorba)

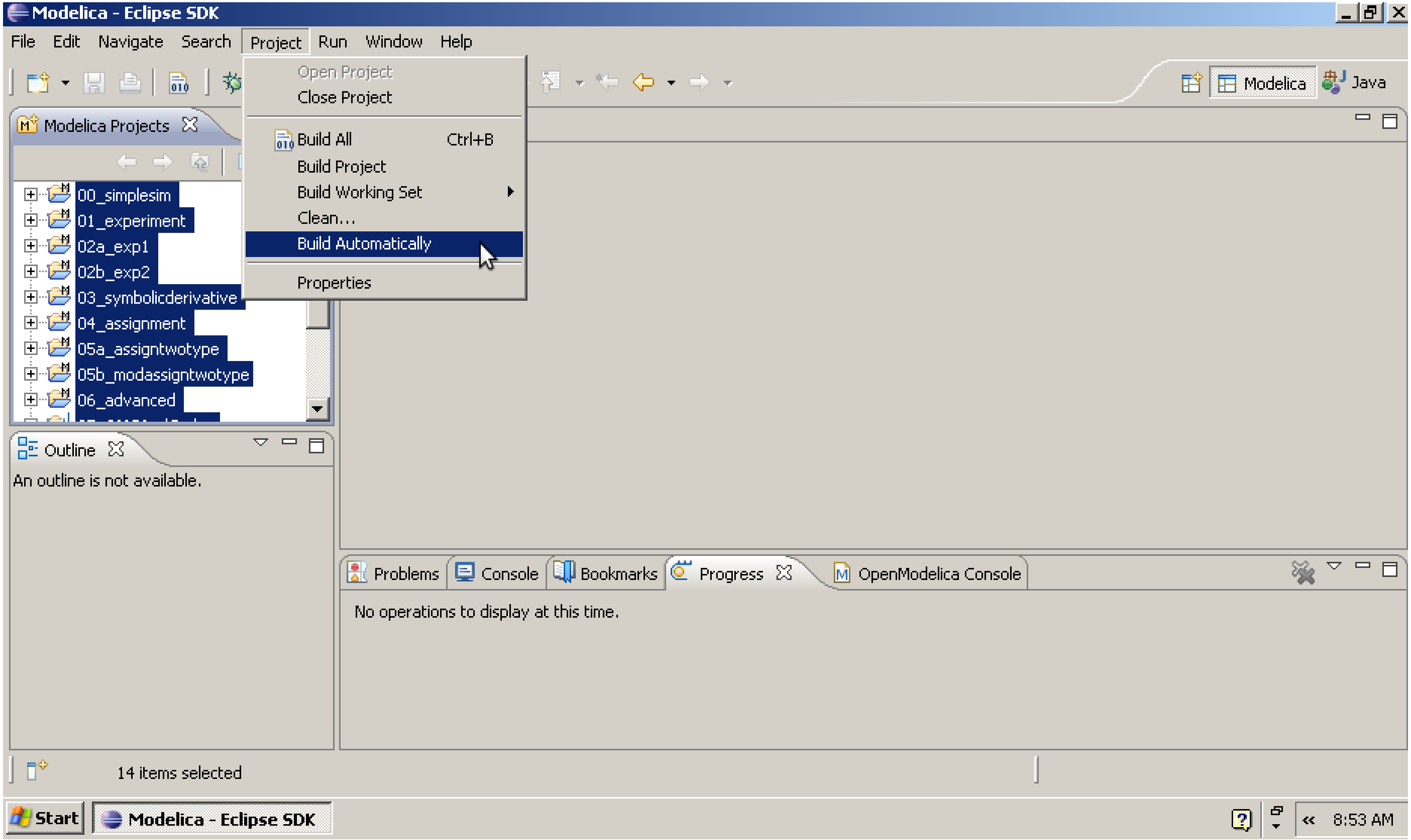
Select and Open all projects



No consoles to display at this time.

14 items selected

Don't build projects automatically

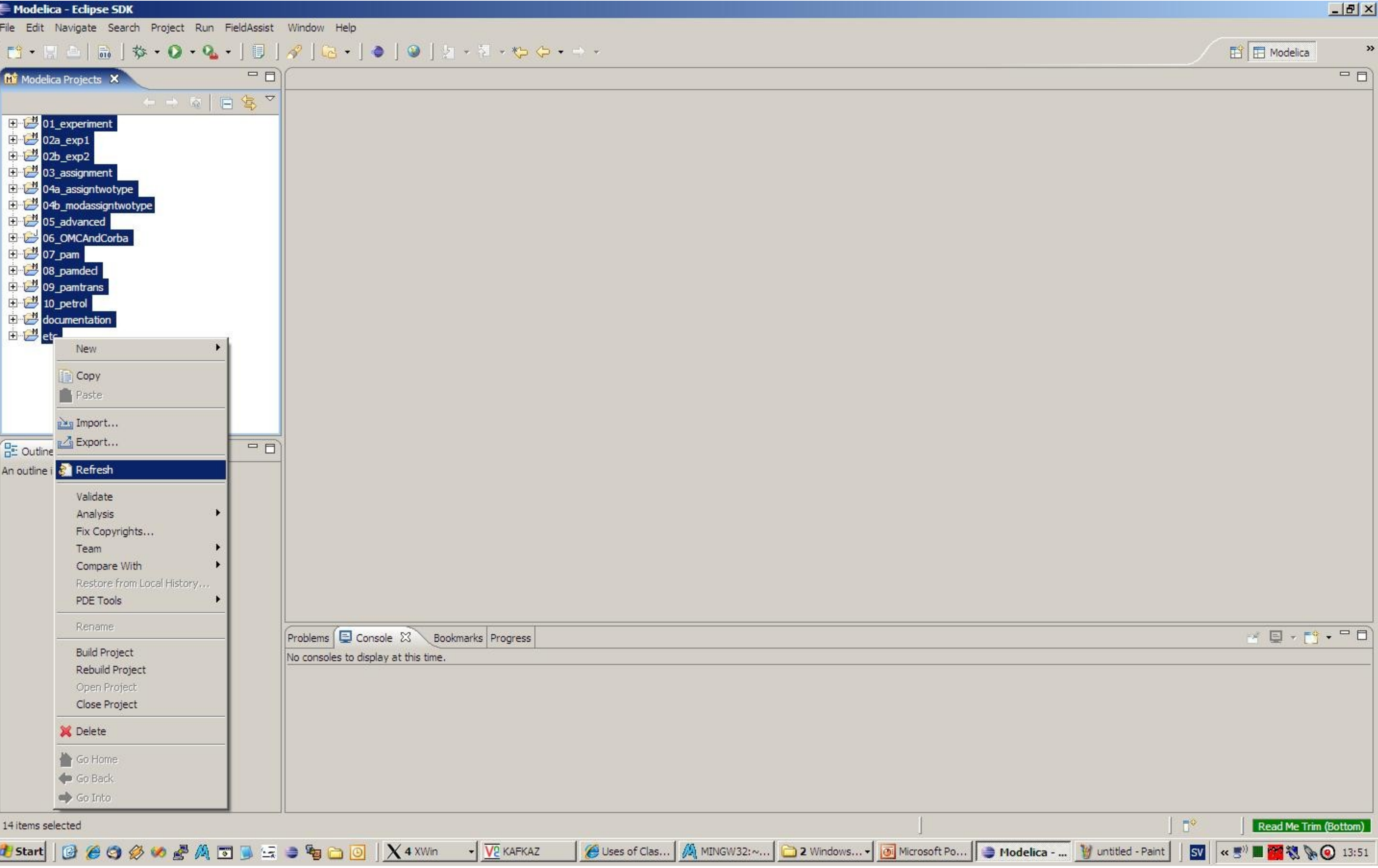


An outline is not available.

No operations to display at this time.

14 items selected

Refresh all projects

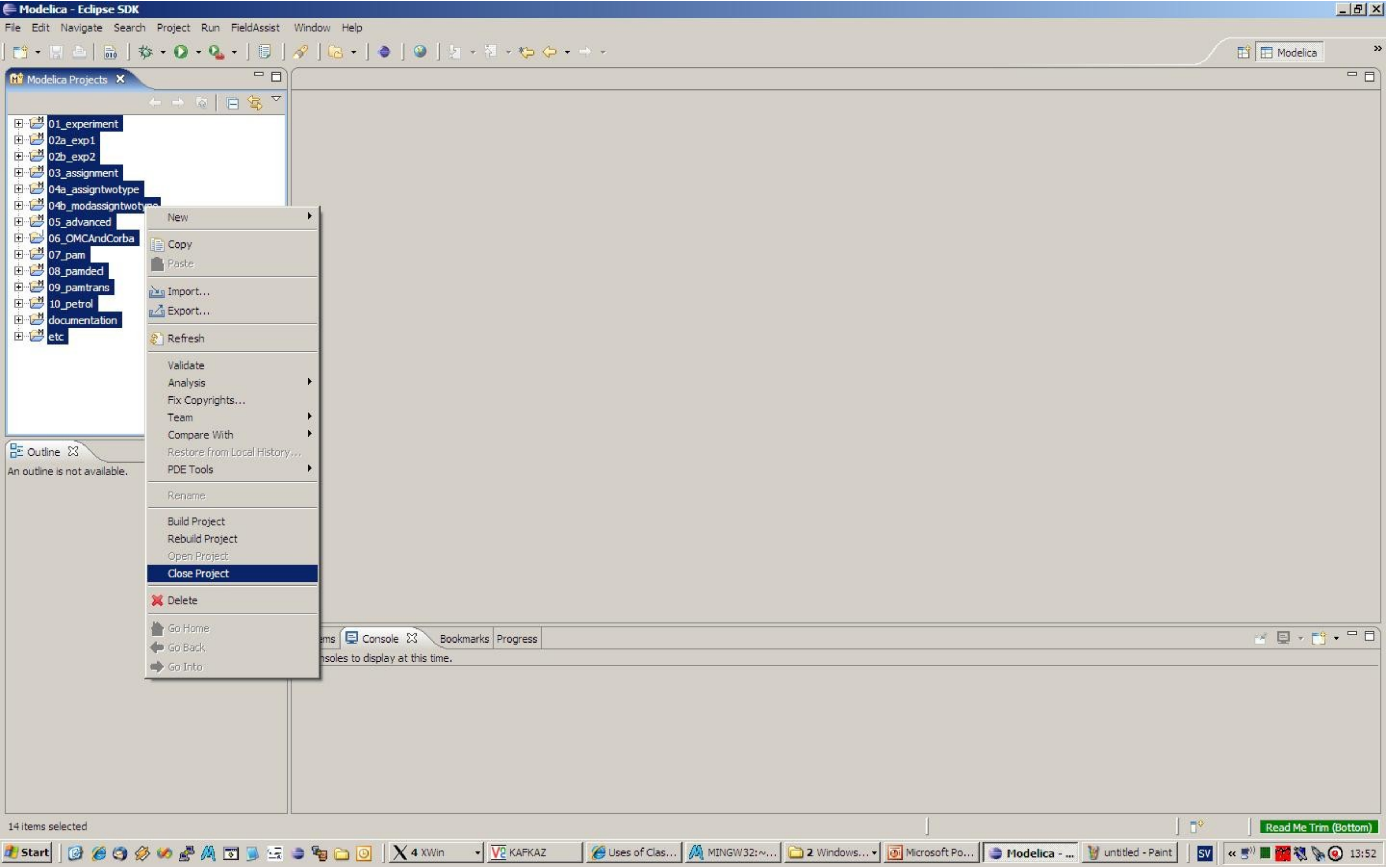


An outline i

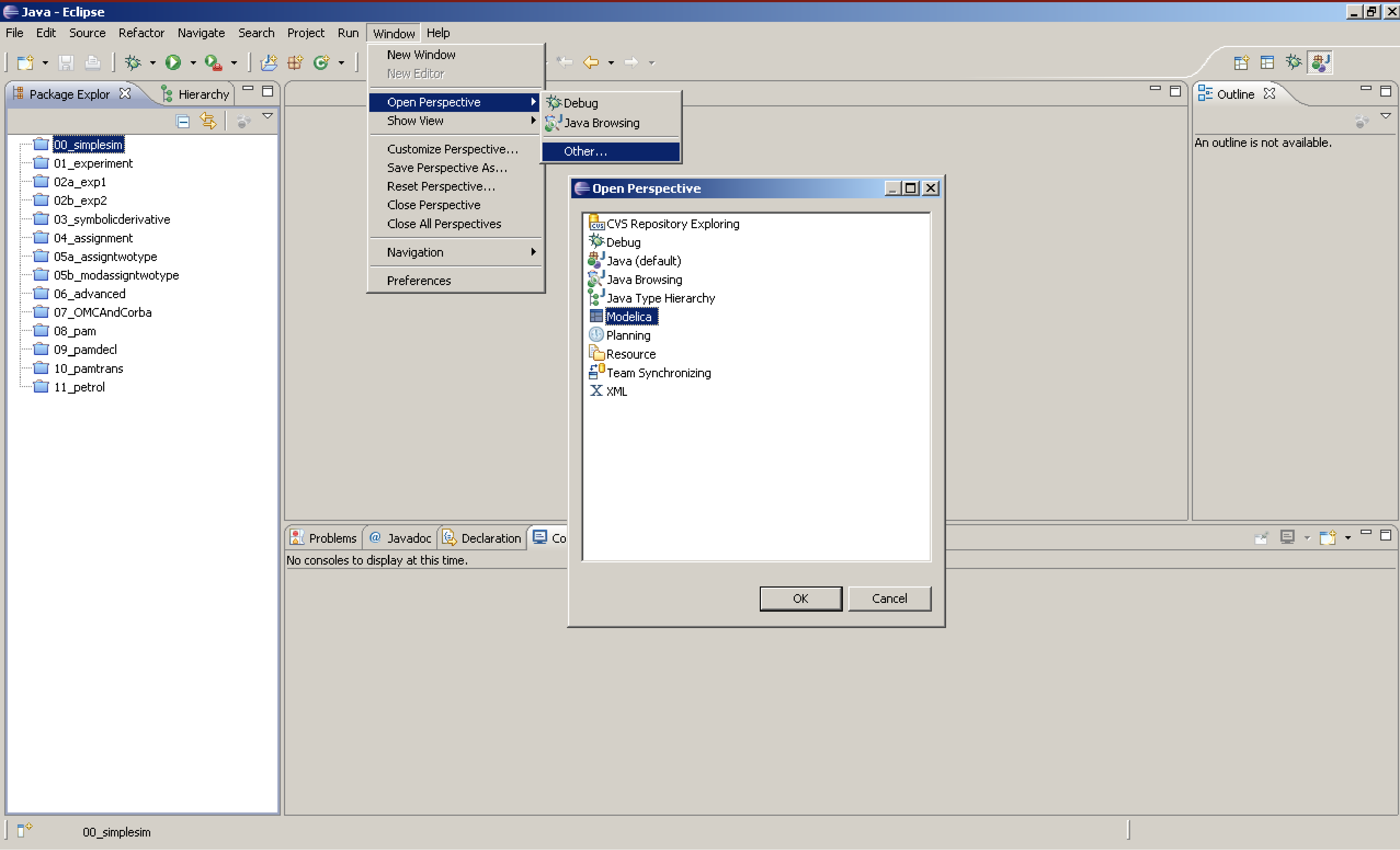
No consoles to display at this time.

14 items selected

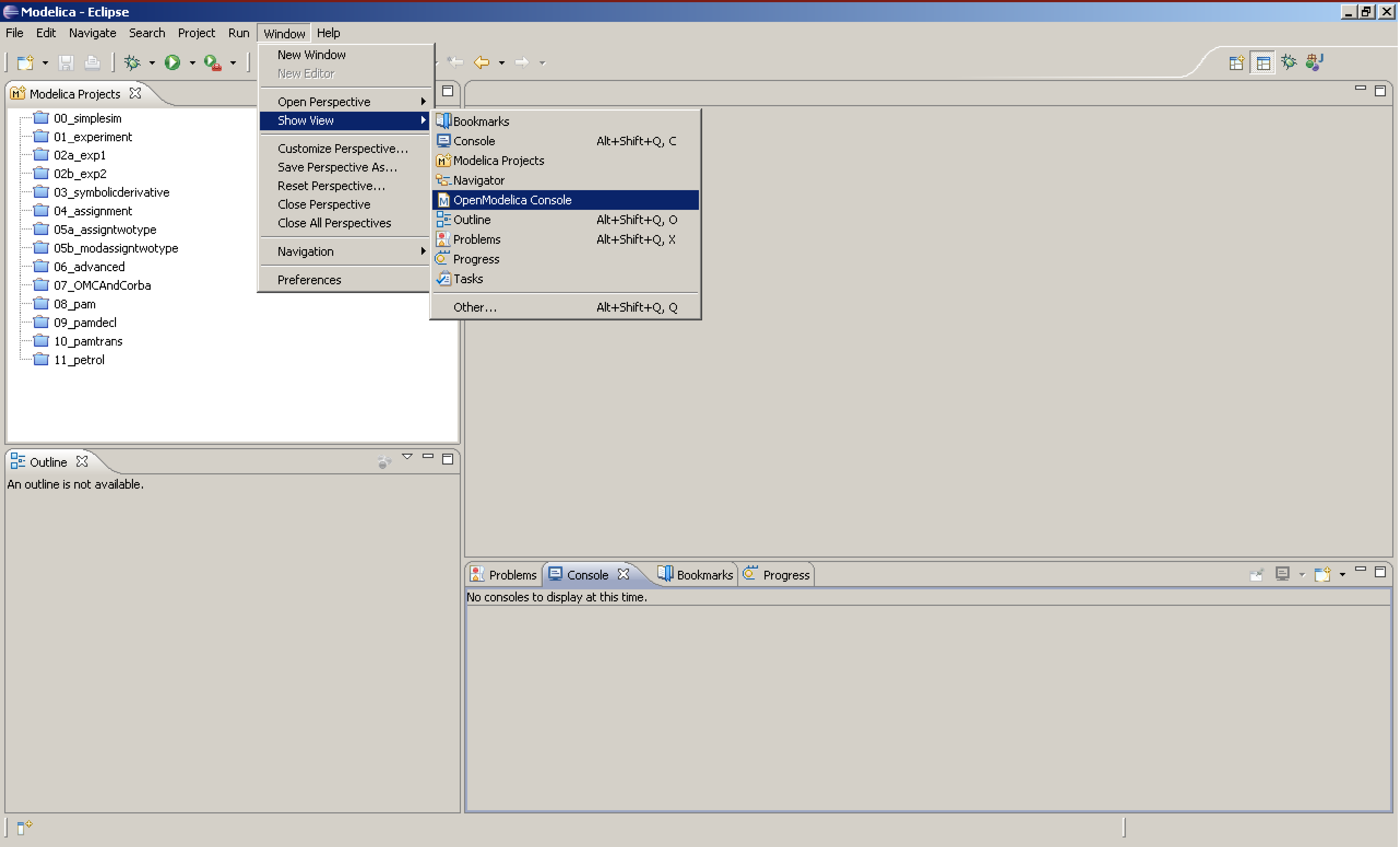
Close all projects



Switch to Modelica Perspective



Show the OpenModelica Console View



An outline is not available.

No consoles to display at this time.

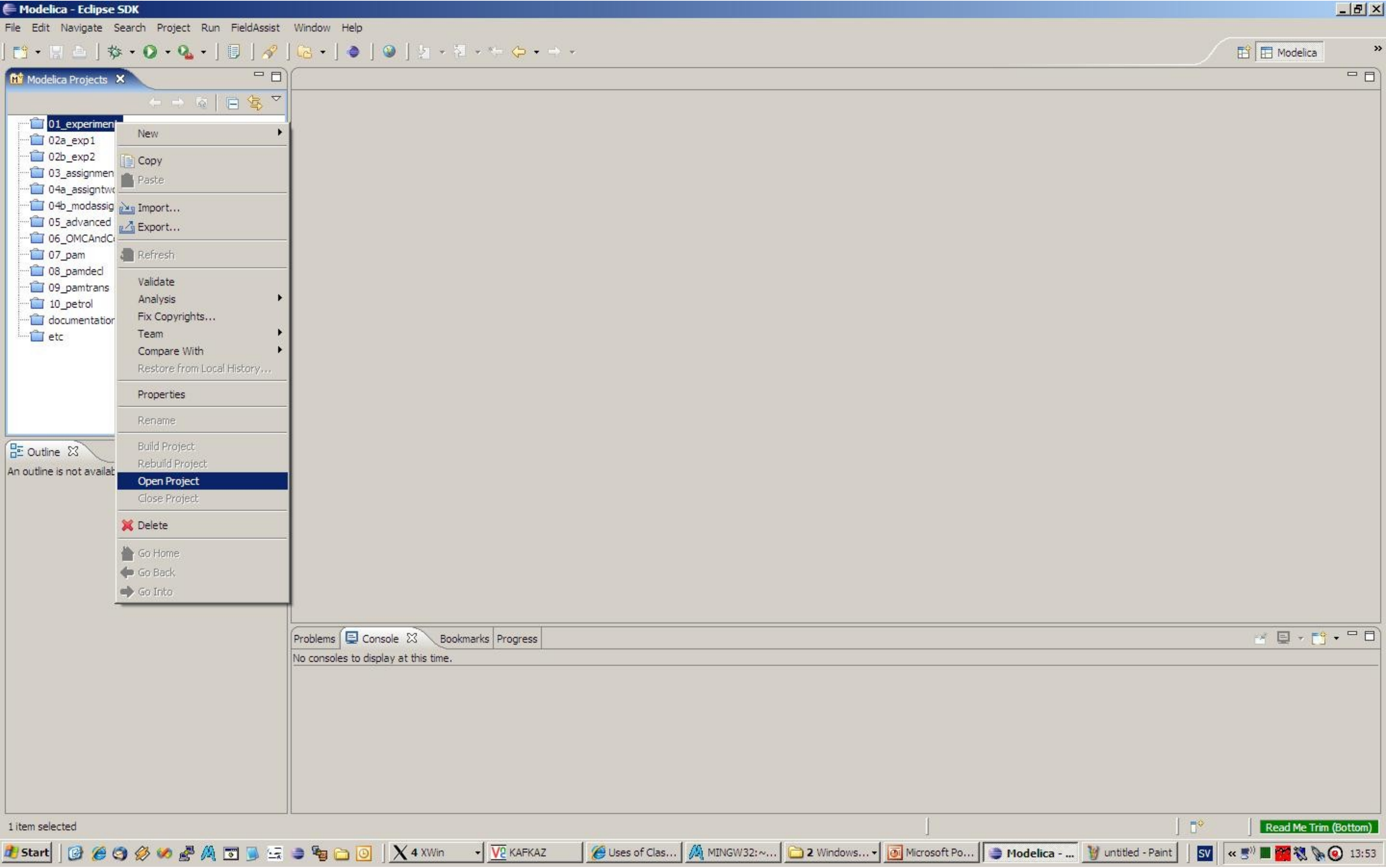
Exercises - 00_simplesim

- See **`README.txt`** in the Eclipse project
- In this exercise you perform a simple simulation in the MDT Eclipse environment
- Assignment
  - Type or copy a simple model into an Eclipse project
  - Open the Eclipse view “OpenModelica console”
  - simulate with the simulate command
  - plot with the plot command.
- Note: In the following exercises you will no longer use the “OpenModelica console”

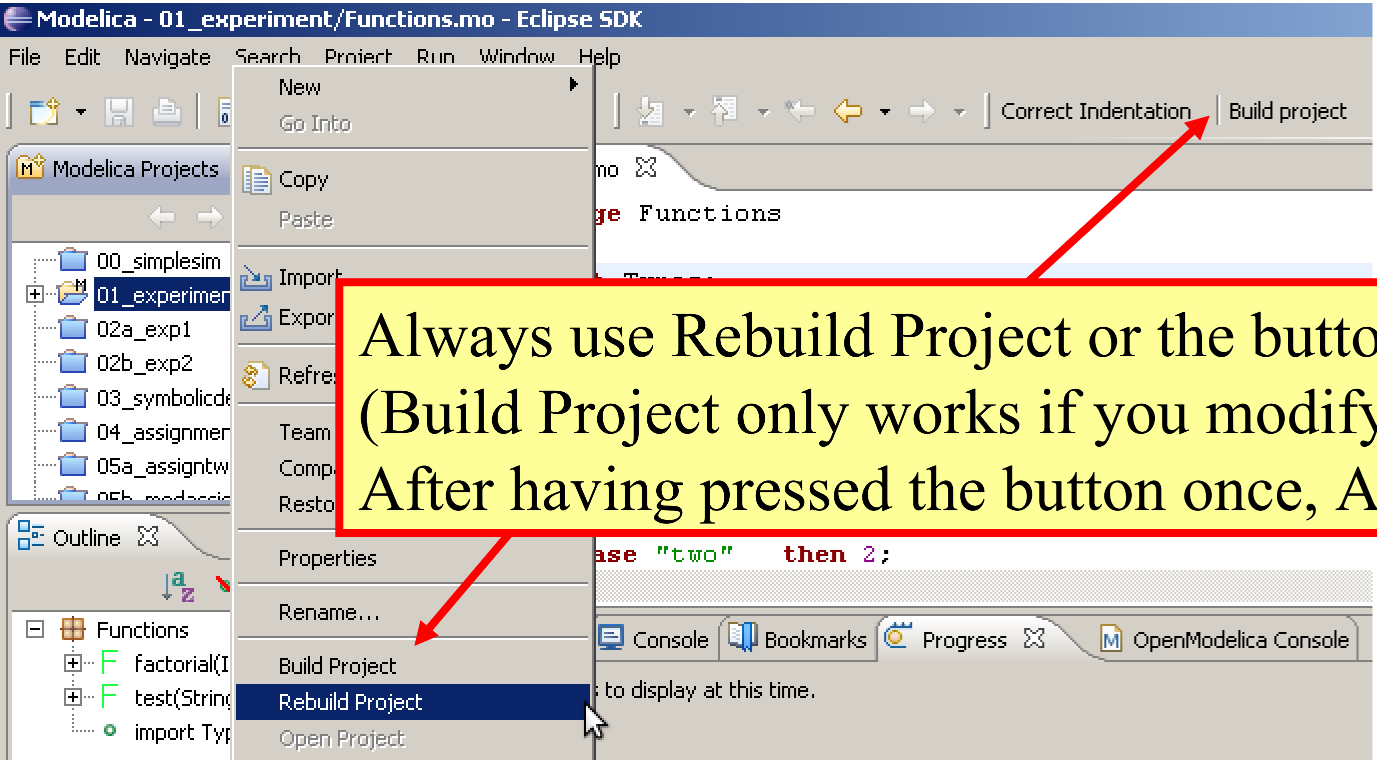
Exercises 01 - 04

- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

Open the first project



Build the first project

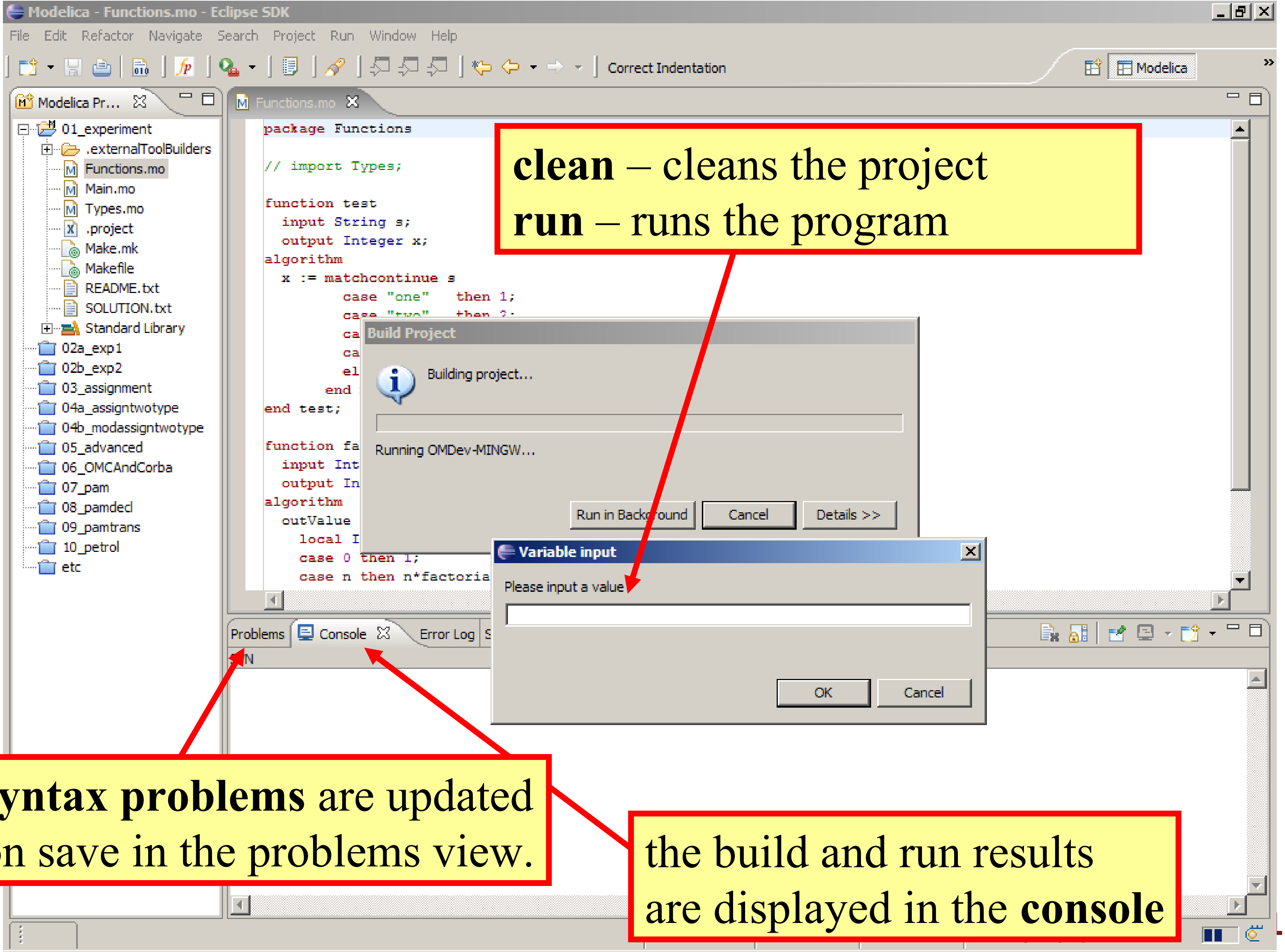


Always use Rebuild Project or the button

(Build Project only works if you modify a source file)

After having pressed the button once, Alt+B can be used

Eclipse – Building a project



```
package Functions

// import Types;

function test
  input String s;
  output Integer x;
algorithm
  x := matchcontinue s
         case "one"   then 1;
         case "two"   then 2;
         ca
         ca
         el
       end
end test;

function fa
  input Int
  output In
algorithm
  outValue
    local I
    case 0 then 1;
    case n then n*factoria
```

Building project...

Running OMDev-MINGW...

Please input a value

clean – cleans the project
run – runs the program

syntax problems are updated on save in the problems view.

the build and run results are displayed in the **console**

OMC Debugging

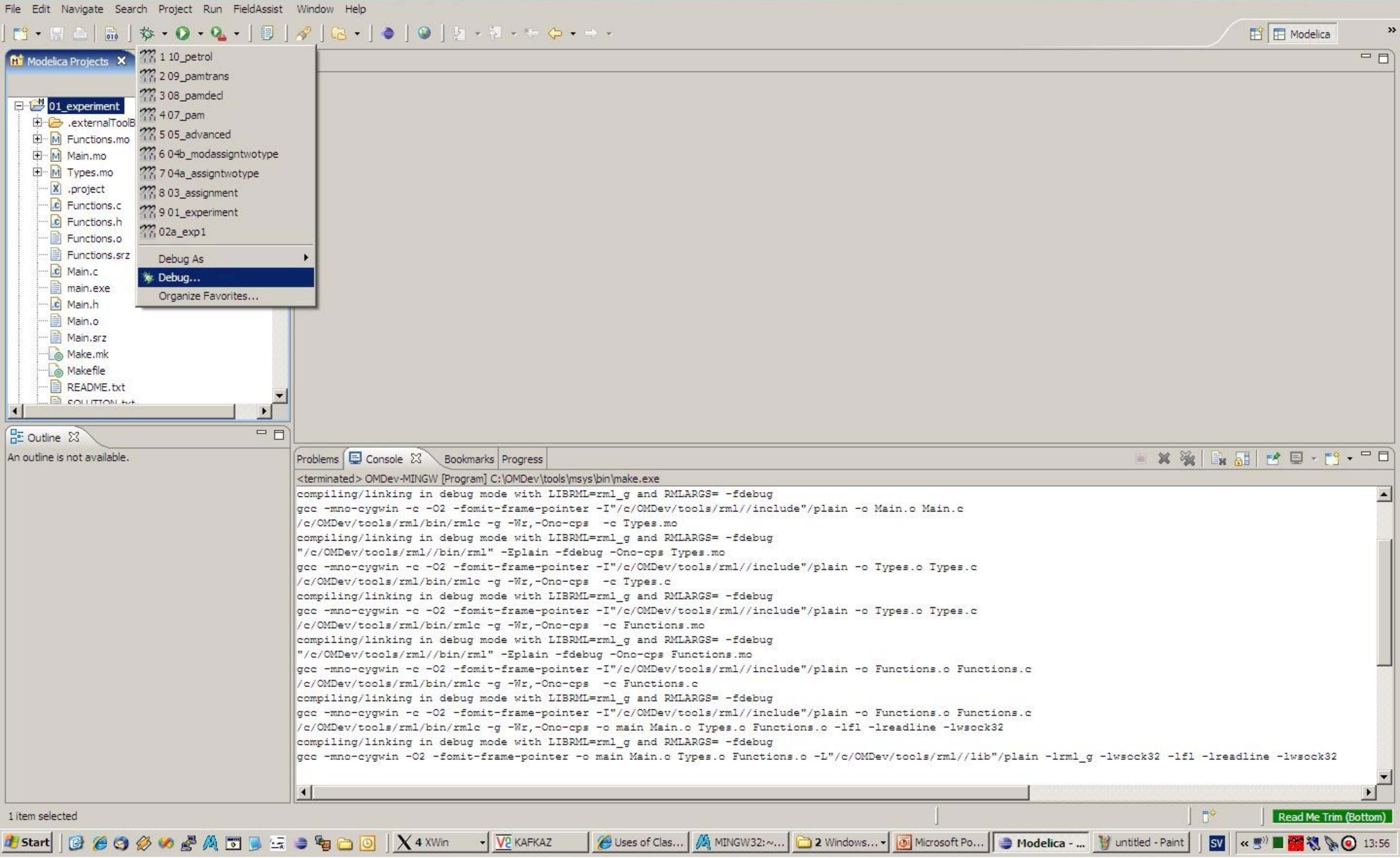
00_simplesim

- 01_experiment
- 02a_exp1, (02b_exp2 optional)
- 03_symbolicderivative
- 04_assignment (optional)

Debugging using OMC

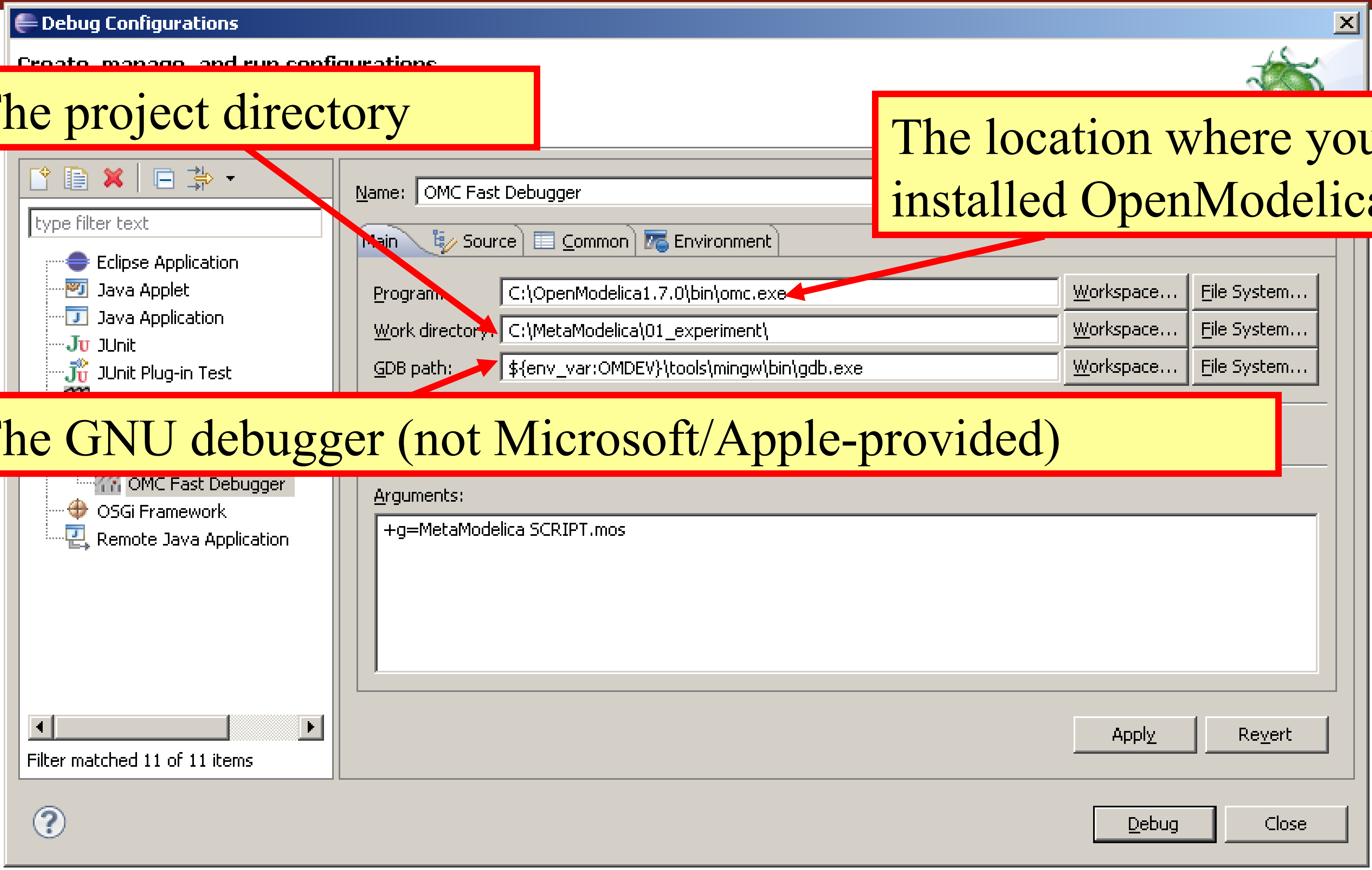
- Very recent work
  - Run target “deps” to build dependencies
  - Then run the debug target
  - Note: Programs that require stdin do not work yet (Eclipse does not send EOF)
- You can print data structures using printAny(x)

Setting the debug configuration



```
<terminated> OMDev-MINGW [Program] C:\OMDev\tools\msys\bin\make.exe
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Main.o Main.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps  -c Types.mo
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
"/c/OMDev/tools/rml//bin/rml" -Eplain -fdebug -Ono-cps Types.mo
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Types.o Types.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps  -c Types.c
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Types.o Types.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps  -c Functions.mo
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
"/c/OMDev/tools/rml//bin/rml" -Eplain -fdebug -Ono-cps Functions.mo
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Functions.o Functions.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps  -c Functions.c
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
gcc -mno-cygwin -c -O2 -fomit-frame-pointer -I"/c/OMDev/tools/rml//include"/plain -o Functions.o Functions.c
/c/OMDev/tools/rml/bin/rmlc -g -Wr,-Ono-cps -o main Main.o Types.o Functions.o -lfl -lreadline -lwsock32
compiling/linking in debug mode with LIBRML=rml_g and RMLARGS= -fdebug
gcc -mno-cygwin -O2 -fomit-frame-pointer -o main Main.o Types.o Functions.o -L"/c/OMDev/tools/rml//lib"/plain -lrml_g -lwsock32 -lfl -lreadline -lwsock32
```

Setup the debugger



Name: OMC Fast Debugger

Program: C:\OpenModelica1.7.0\bin\omc.exe

Work directory: C:\MetaModelica\01_experiment\

GDB path: ${env_var:OMDEV}\tools\mingw\bin\gdb.exe

Arguments:

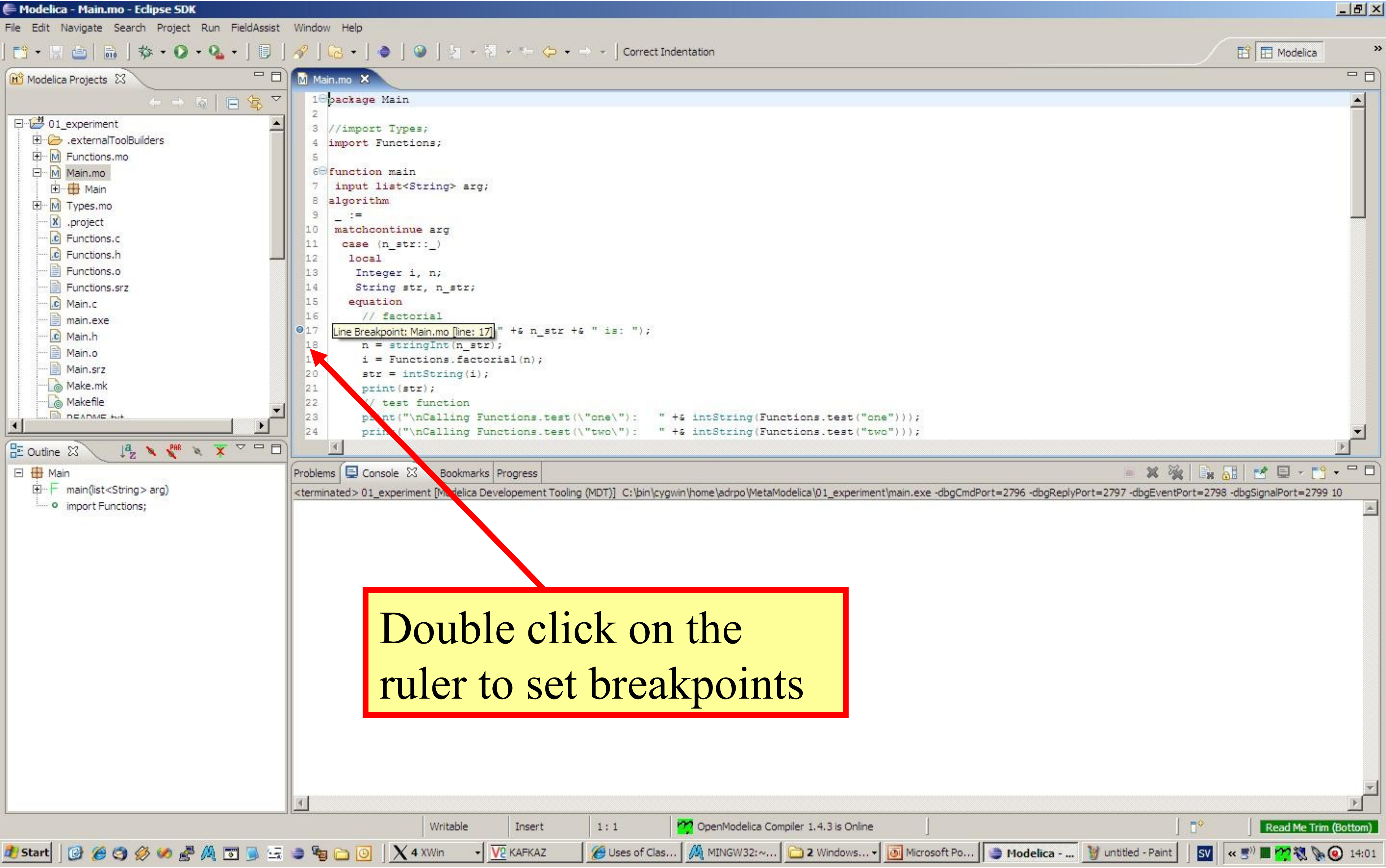
```
+g=MetaModelica SCRIPT.mos
```

The project directory

The location where you installed OpenModelica

The GNU debugger (not Microsoft/Apple-provided)

Set breakpoints in .mo file

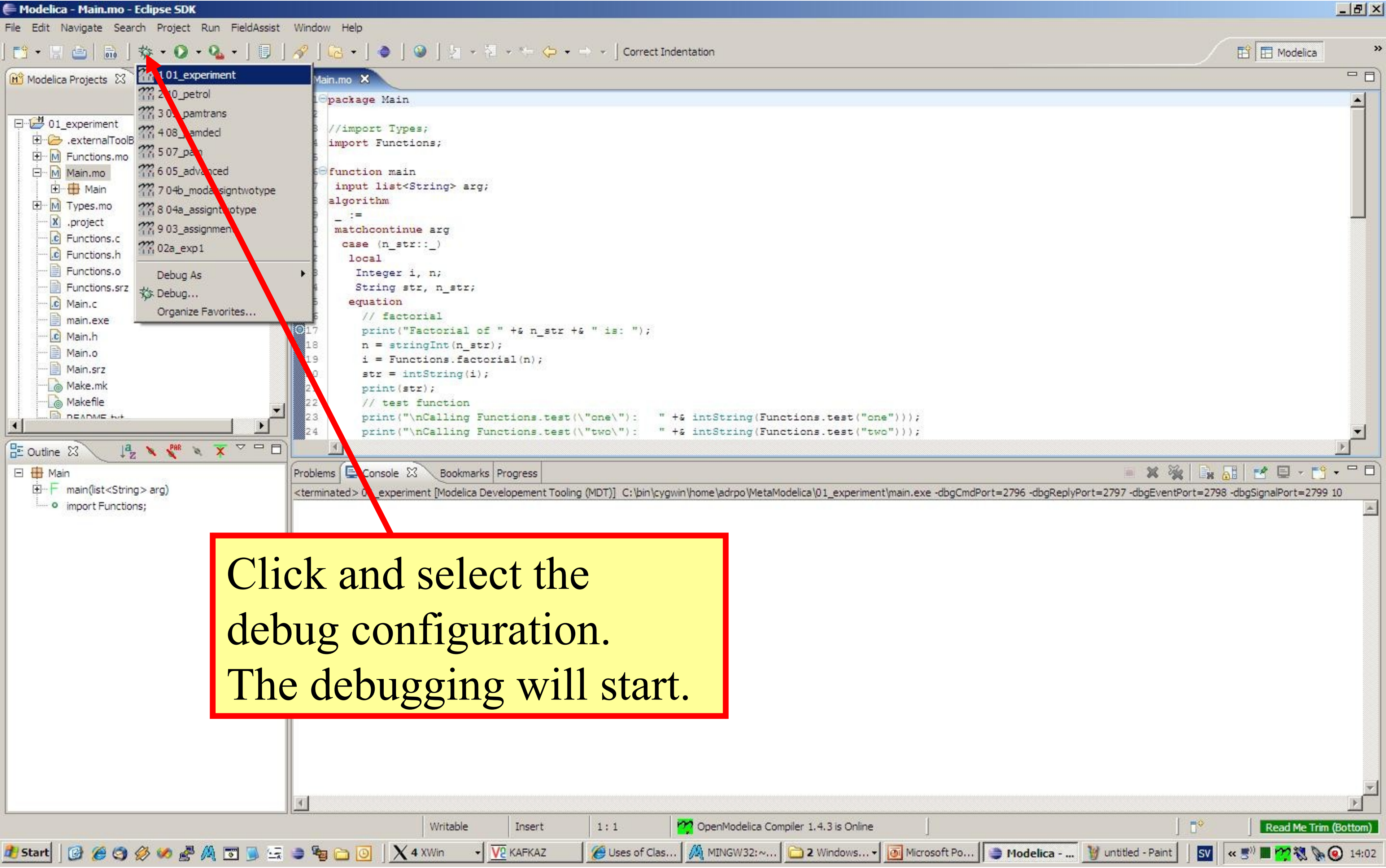


```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
                  " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
     print(str);
     // test function
     print("\nCalling Functions.test(\"one\"):   " +& intString(Functions.test("one")));
     print("\nCalling Functions.test(\"two\"):   " +& intString(Functions.test("two")));
```

```
<terminated> 01_experiment [Modelica Developement Tooling (MDT)] C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=2796 -dbgReplyPort=2797 -dbgEventPort=2798 -dbgSignalPort=2799 10
```

Double click on the ruler to set breakpoints

Run the debug configuration to start debugging

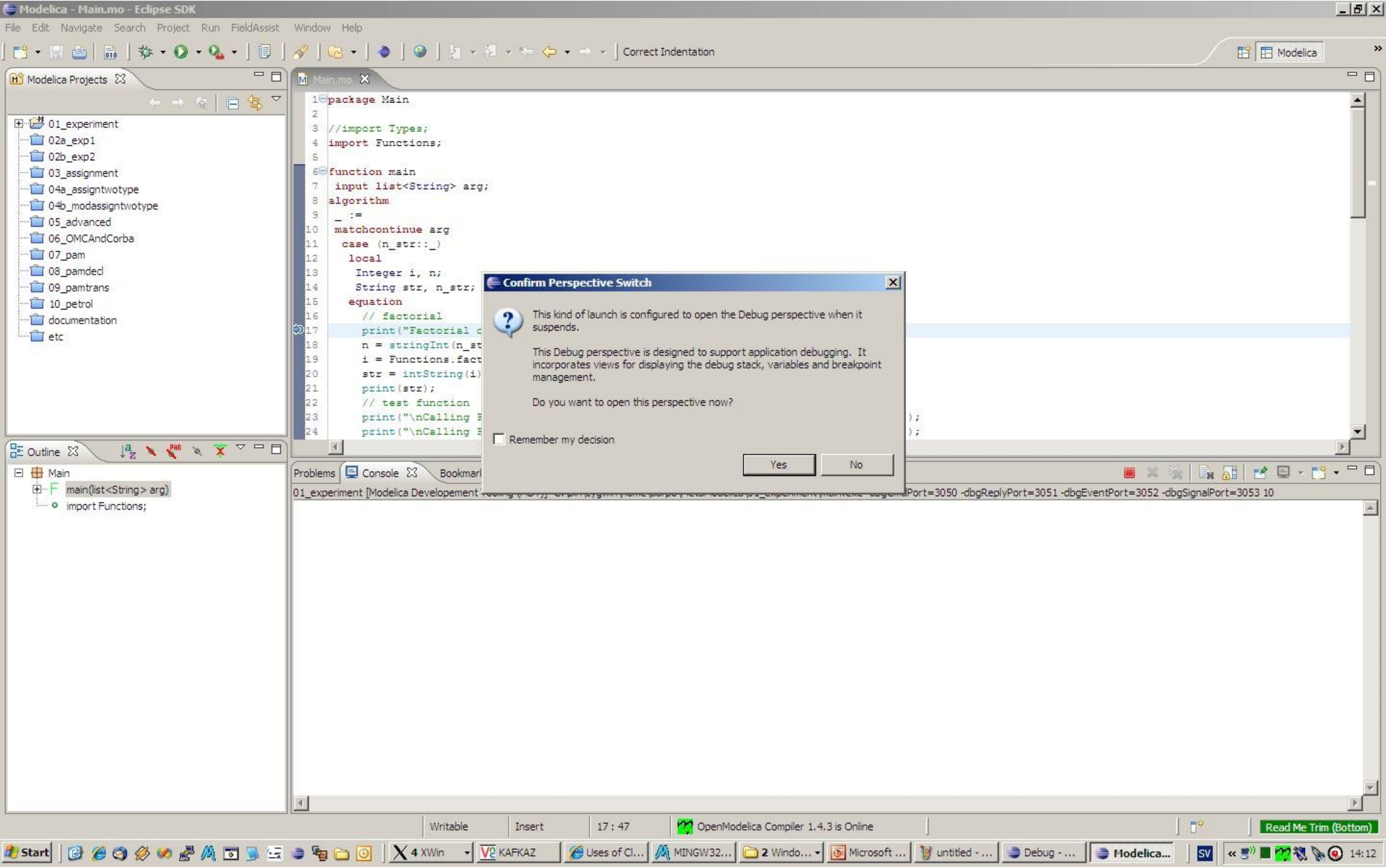


```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
    // factorial
    print("Factorial of " +& n_str +& " is: ");
    n = stringInt(n_str);
    i = Functions.factorial(n);
    str = intString(i);
    print(str);
    // test function
    print("\nCalling Functions.test(\"one\"):   " +& intString(Functions.test("one")));
    print("\nCalling Functions.test(\"two\"):   " +& intString(Functions.test("two")));
```

```
<terminated> 0_experiment [Modelica Developement Tooling (MDT)] C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=2796 -dbgReplyPort=2797 -dbgEventPort=2798 -dbgSignalPort=2799 10
```

Click and select the debug configuration. The debugging will start.

Eclipse will ask to switch to debugging perspective



```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial c
     n = stringInt(n_st
     i = Functions.fact
     str = intString(i)
     print(str);
     // test function
     print("\nCalling F
     print("\nCalling F
```

Confirm Perspective Switch

This kind of launch is configured to open the Debug perspective when it suspends.

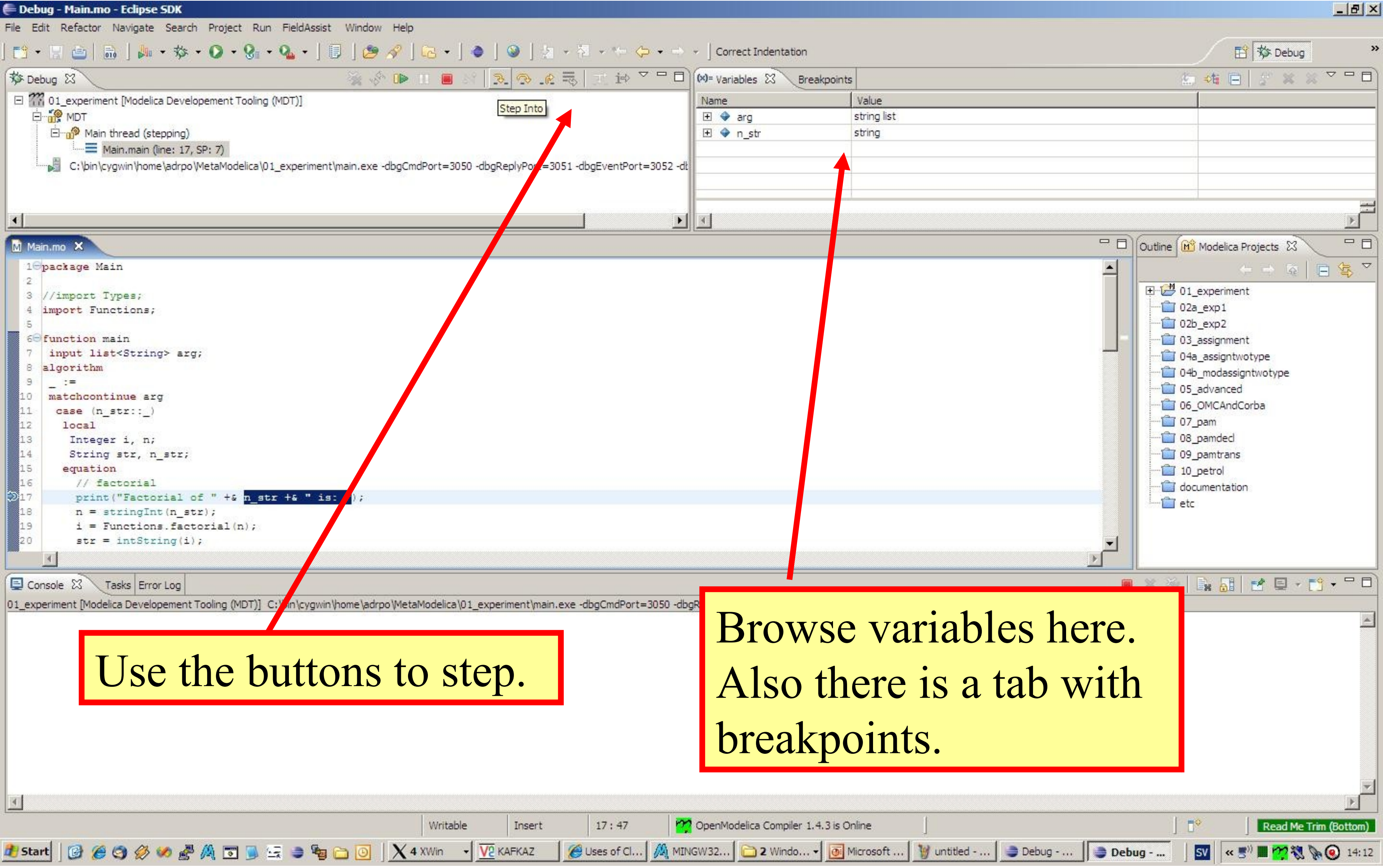
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.

Do you want to open this perspective now?

☐ Remember my decision

```
01_experiment [Modelica Developement Port=3050 -dbgReplyPort=3051 -dbgEventPort=3052 -dbgSignalPort=3053 10
```

Debugging perspective



Debug

```
01_experiment [Modelica Developement Tooling (MDT)]
  MDT
    Main thread (stepping)
      Main.main (line: 17, SP: 7)
  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgReplyPort=3051 -dbgEventPort=3052 -db
```

Name	Value
arg	string list
n_str	string

Main.mo

```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial of " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
```

Modelica Projects

- 01_experiment
- 02a_exp1
- 02b_exp2
- 03_assignment
- 04a_assigntwotype
- 04b_modassigntwotype
- 05_advanced
- 06_OMCAndCorba
- 07_pam
- 08_pamdecl
- 09_pamtrans
- 10_petrol
- documentation
- etc

Console

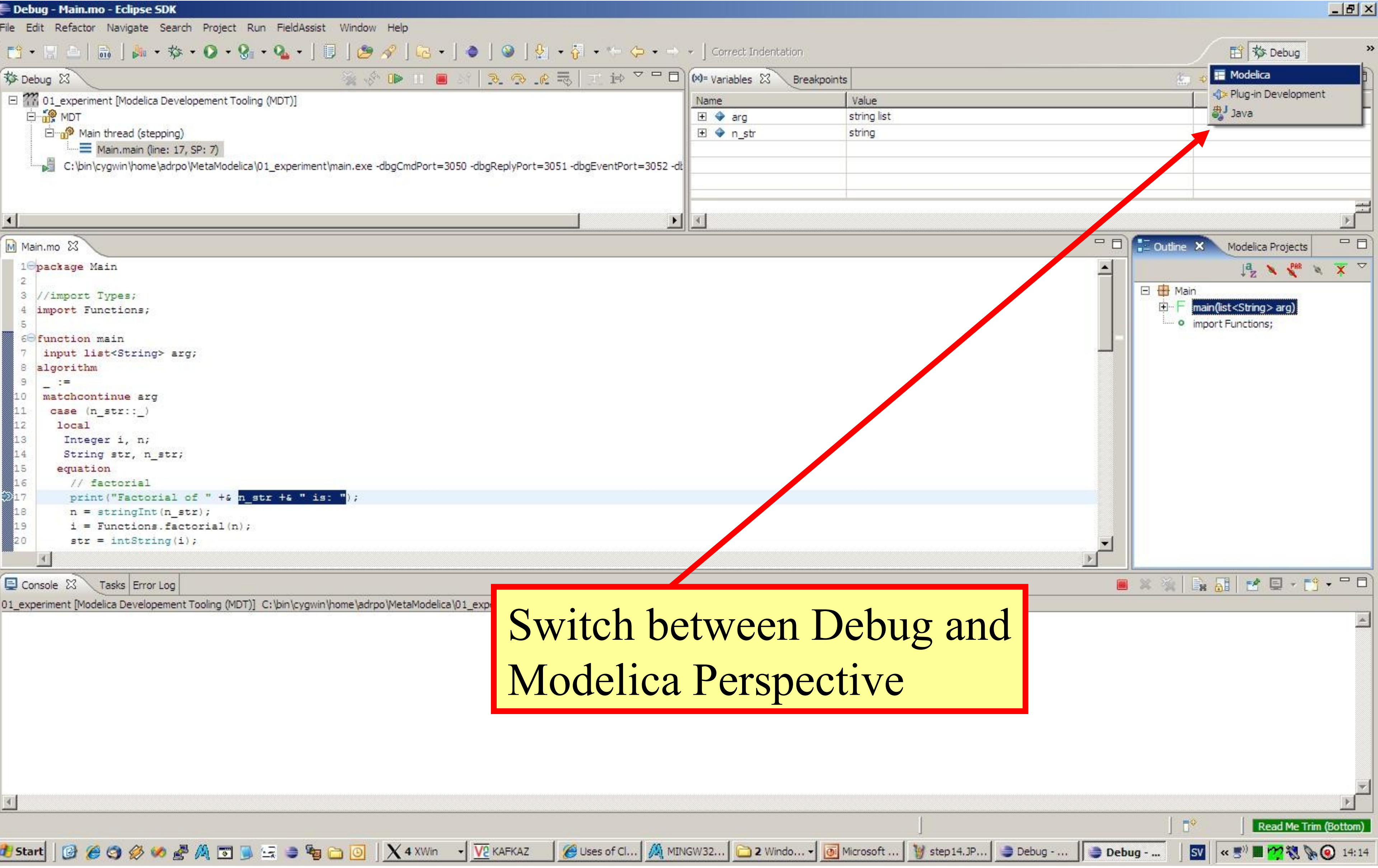
```
01_experiment [Modelica Developement Tooling (MDT)] C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgR
```

Writable | Insert | 17 : 47 | OpenModelica Compiler 1.4.3 is Online | Read Me Trim (Bottom)

Use the buttons to step.

Browse variables here. Also there is a tab with breakpoints.

Switching perspectives



```
01_experiment [Modelica Developement Tooling (MDT)]
  MDT
    Main thread (stepping)
      Main.main (line: 17, SP: 7)
  C:\bin\cygwin\home\adrpo\MetaModelica\01_experiment\main.exe -dbgCmdPort=3050 -dbgReplyPort=3051 -dbgEventPort=3052 -dt
```

Name	Value
arg	string list
n_str	string

```
package Main

//import Types;
import Functions;

function main
 input list<String> arg;
algorithm
 _ :=
 matchcontinue arg
  case (n_str::_)
   local
    Integer i, n;
    String str, n_str;
   equation
     // factorial
     print("Factorial of " +& n_str +& " is: ");
     n = stringInt(n_str);
     i = Functions.factorial(n);
     str = intString(i);
```

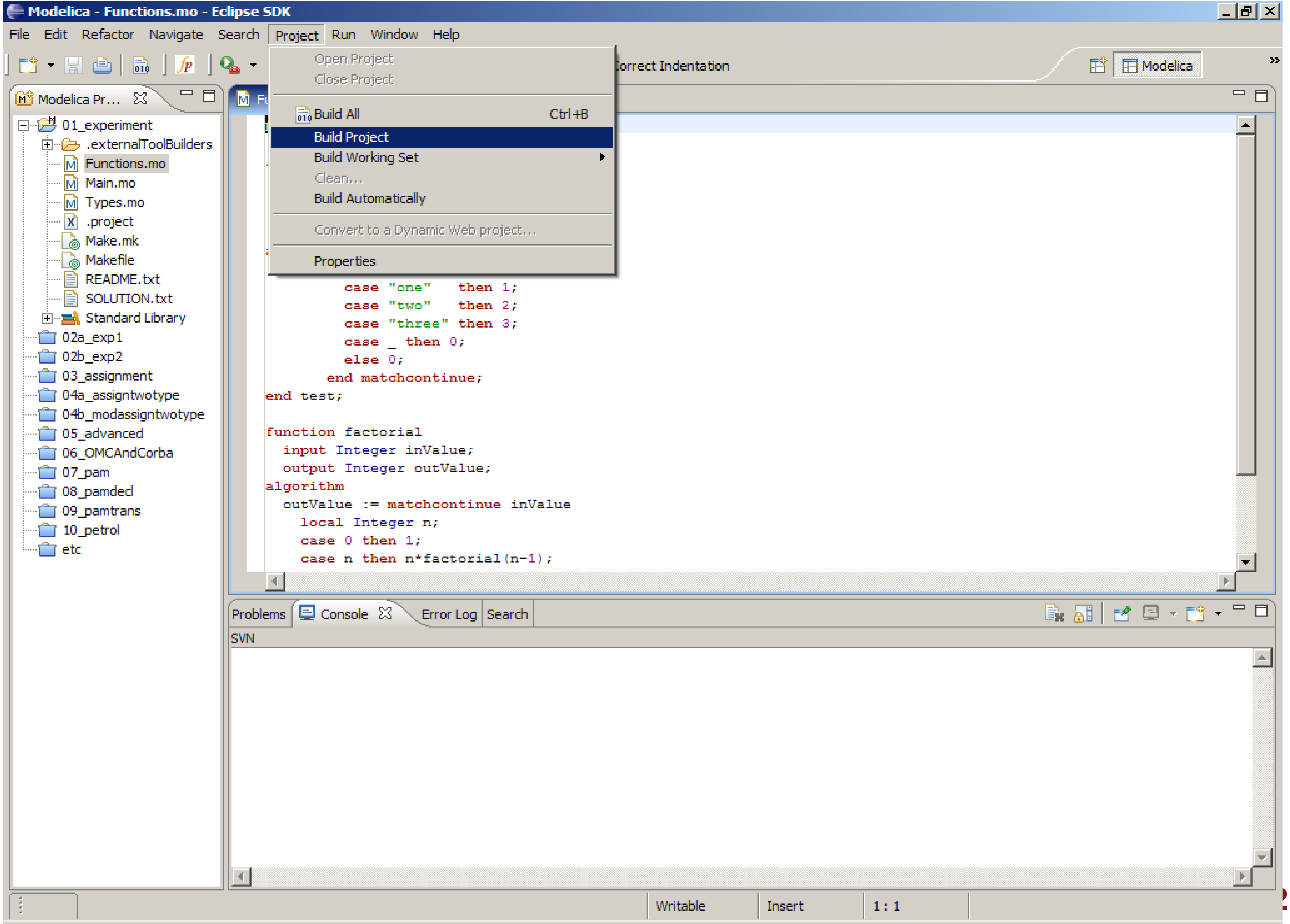
```
01_experiment [Modelica Developement Tooling (MDT)]  C:\bin\cygwin\home\adrpo\MetaModelica\01_exp
```

Switch between Debug and Modelica Perspective

Eclipse Setup – Creating the projects

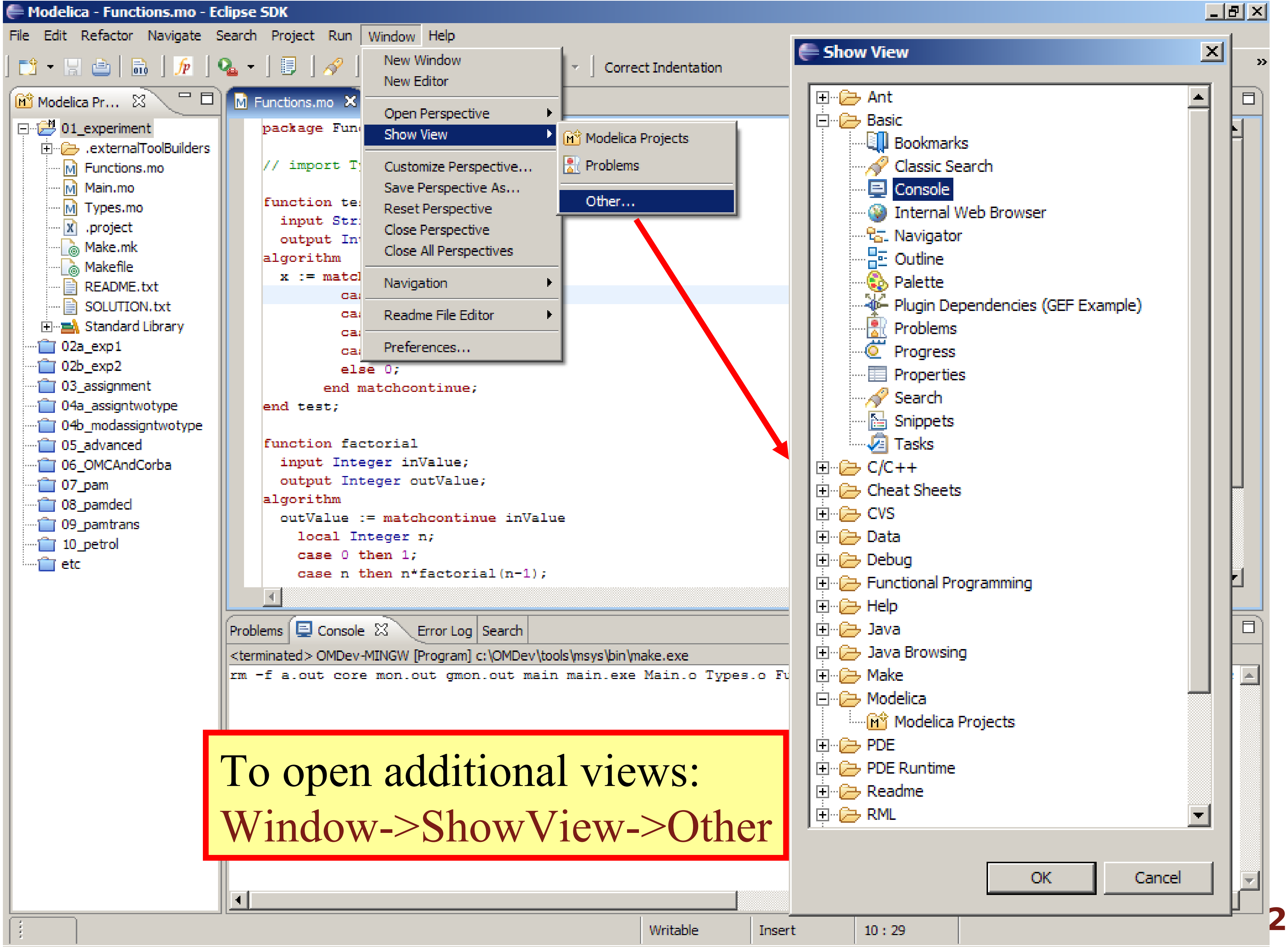
- repeat the procedure for all exercises
  - 01_experiment
  - 02a_exp1, 02b_exp2
  - 03_symbolicderivative
  - 04_assignment
  - 05a_assigntwotype
  - 05b_modassigntwotype
  - 06_advanced
  - 07_OMCAndCorba
- leave open only the project you are working on! close all the others

Eclipse – Building a project



```
         case "one"   then 1;
         case "two"   then 2;
         case "three" then 3;
         case _ then 0;
         else 0;
       end matchcontinue;
end test;

function factorial
  input Integer inValue;
  output Integer outValue;
algorithm
  outValue := matchcontinue inValue
    local Integer n;
    case 0 then 1;
    case n then n*factorial(n-1);
```

Eclipse – Opening views



```
package Fun

// import T

function te
  input Str
  output In
algorithm
  x := matc
         ca
         ca
         ca
         ca
         else 0;
      end matchcontinue;
end test;

function factorial
  input Integer inValue;
  output Integer outValue;
algorithm
  outValue := matchcontinue inValue
    local Integer n;
    case 0 then 1;
    case n then n*factorial(n-1);
```

```
<terminated> OMDev-MINGW [Program] c:\OMDev\tools\msys\bin\make.exe
rm -f a.out core mon.out gmon.out main main.exe Main.o Types.o Fu
```

To open additional views:
Window->ShowView->Other

Exercises

- Each exercise is in a different Eclipse project
- All exercises have :
  - **`README.txt`** - information about the exercise
  - **`SOLUTION.txt`** – the solution of the exercise (if the exercise has some implementation assignment)
  - **`program.txt`** – input program to the exercise, edit if needed (for the exercises which have an input)
- Consult the **`MetaModelica Programming Guide`** (1.0 Draft or 2.0) and the slides if you need additional information during the exercises.
- Of course, feel free to ask us any type of questions, it is faster and better!

Exercises - 01_experiment

- See **`README.txt`** in the Eclipse project
- In this exercise you experiment with
  - Types
  - Constants
  - Functions
- Assignment
  - Write functions in **`Functions.mo`** to display the constants defined in **`Types.mo`**.
  - Search for **`// your code here`** in **`Main.mo`** and **`Functions.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 02a_exp1

- See **`README.txt`** in the Eclipse project
- In this exercise you will add new constructs to the exp1 language and deal with their evaluation.
- Assignment - add new constructs to the language
  - a power operator (`^`)
  - a factorial operator (`!`)
  - search for **`// your code here`** within **`Exp1.mo`**
- Note
  - The parser/lexer are ready, but give parser errors for the new operators until they are added in **`Exp1.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 02b_exp2

- See **`README.txt`** in the Eclipse project

- In this exercise you will explore a different way to model the **`exp1`** language using different **`Exp`** trees.
- Explore the **`Exp2.mo`** file and compare it with **`Exp1.mo`** file.

Exercises – 03_symbolicderivative

- See **`README.txt`** in the Eclipse project
- Assignment:
  - add rules to derive ‘-’, ‘*’, sine, cosine and power expressions
  - add rules to simplify ‘-’, sine, cosine and power expressions
  - search for **`// your code here`** within **`SymbolicDerivative.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 04_assignment

- See **`README.txt`** in the Eclipse project
- Assignment - add functions to print:
  - the assignments present in the current program before the actual evaluation
  - the environment after it was augmented with the assignments
  - search for **`// your code here`** within **`Assignment.mo`**
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

Exercises – 05a_assigntwotype

- See `README.txt` in the Eclipse project
- Assignment - add functions to print:
  - add a new `String` type which can hold only integers as strings to the current `Exp` node
  - add cases to evaluate expressions/assignments of the form `"2" + 1 + "1" + 1.0` in the `eval` function
  - search for `// your code here` within `AssignTwoType.mo`
- Compare your solution with the `SOLUTION.txt` you find in the Eclipse project

Exercises – 05b_modassigntwotype

- See **`README.txt`** in the Eclipse project
- In this exercise you will explore a different way to structure your code within different packages.
- The code from **`05a_assigntwotype`** is now split over 4 packages.
- Compare the 05a/b projects.

Exercises – 06_advanced (I)

- See **`README.txt`** in the Eclipse project
- In this exercise you experiment with
  - polymorphic types
  - constants
  - higher order functions
- Assignment 1
  - Write a polymorphic function that orders a list of any type.
  - The function has as input a list and a compare function between the objects of that list.
  - Write the comparison functions for **`Integers`**, **`Strings`** and **`Reals`**.
  - Test your function on the **`Types.intList`**

Exercises – 06_advanced (II)

- See **`README.txt`** in the Eclipse project
- Assignment 2
  - Write a polymorphic map function that applies a function over a list and returns a new list with the result.
  - Write three functions that transform from:
    - integer to real
    - integer to string
    - real to string
  - Use your map function and the two transformation functions to transform the **`Types.intList`** to a list of reals and a list of string, then apply the ordering function from Assignment 1 on the newly created lists

Exercises – 06_advanced (III)

- See **`README.txt`** in the Eclipse project
- Assignment 3
  - Write a polymorphic map function that applies a print function over a list (of Strings) and prints the it.
  - Use the transformer functions from real->string and integer->string from Assignment 2 to transform the real list or the integer list to a string list for printing.
- Compare your solution with the **`SOLUTION.txt`** you find in the Eclipse project

OMC Overview

We are Switching to OMC Overview now!

OpenModelica Development Toolkit (OMDev)

- OMDev is a pre-packaged pre-compiled kit containing all tools needed for OpenModelica development.
  - Just unpack and set some environment variables. (Windows)
  - apt-get build-dep openmodelica (Ubuntu/Debian Linux)
  - port install depends:openmodelica
- MetaModelica Compiler (MMC) – for developing OMC
- OpenModelica Compiler (OMC) – for browsing support
- Eclipse plugin MDT – (Modelica Development Tooling), e.g. for compiler (OMC) development
- Pre-compiled Corba (MICO or omniORB) for tool communication
- Packaged Gnu compiler (GCC; Mingw version for Windows)
- Emacs mode
- Online (web) Subversion for version handling
- Online (web) Codebeamer for bug reporting and management
- Automatic regression testing using a test suite
- Unit testing using the bootstrapped OpenModelica Compiler
- Interactive MetaModelica debugger

OpenModelica Environment Architecture

Eclipse Plugin
Editor/Browser

Graphical Model
Editor/Browser

Interactive
session handler

Emacs
Editor/Browser

Textual
Model Editor

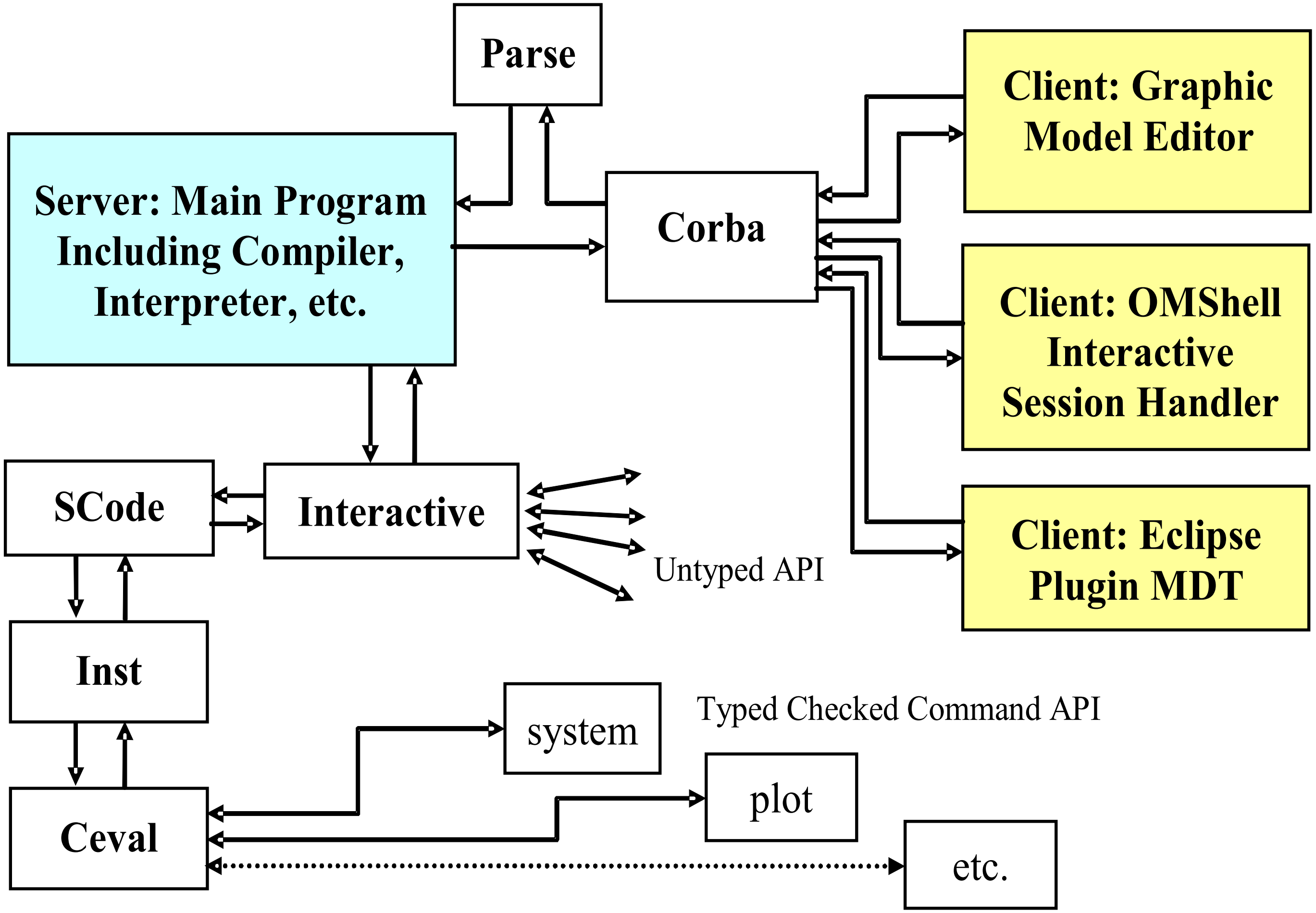
DrModelica
OMNoteBook
Model Editor

Execution

Modelica
Compiler

Modelica
Debugger

OpenModelica Client-Server Architecture



Parse

Server: Main Program Including Compiler, Interpreter, etc.

Corba

Client: Graphic Model Editor

Client: OMShell Interactive Session Handler

Client: Eclipse Plugin MDT

SCode

Interactive

Untyped API

Inst

Ceval

system

Typed Checked Command API

plot

etc.

Released in OpenModelica 1.7.0

- OpenModelica Compiler/Interpreter – OMC
- Interactive session handler – OMShell
- OpenModelica Notebook with DrModelica and DrControl – OMNotebook
- OpenModelica Eclipse plugin MDT
- OMEdit connection editor
- MetaModelica Debugger
  - New version in OpenModelica 1.8.0

OpenModelica Compiler/Interpreter

- OpenModelica 1.7.1
- Currently implemented in 194 000 lines of MetaModelica (excl. generated code, external C)
- Includes code generation, BLT-transformation, index reduction, connection to DASSL, etc.
- Most of the Modelica 3.1 language including classes, functions, inheritance, modifications, import, etc.
- Hybrid/Discrete event support

Corba Client-Server API

- Simple text-based (string) communication in Modelica Syntax
- API supporting model structure query and update

Example Calls:
Calls fulfill the normal Modelica function call syntax.:

```
saveModel("MyResistorFile.mo",MyResistor)
```

will save the model MyResistor into the file “MyResistorFile.mo”.

For creating new models it is most practical to send a model, e.g.:

```
model Foo   end Foo;
```

or, e.g.,

```
connector Port   end Port;
```

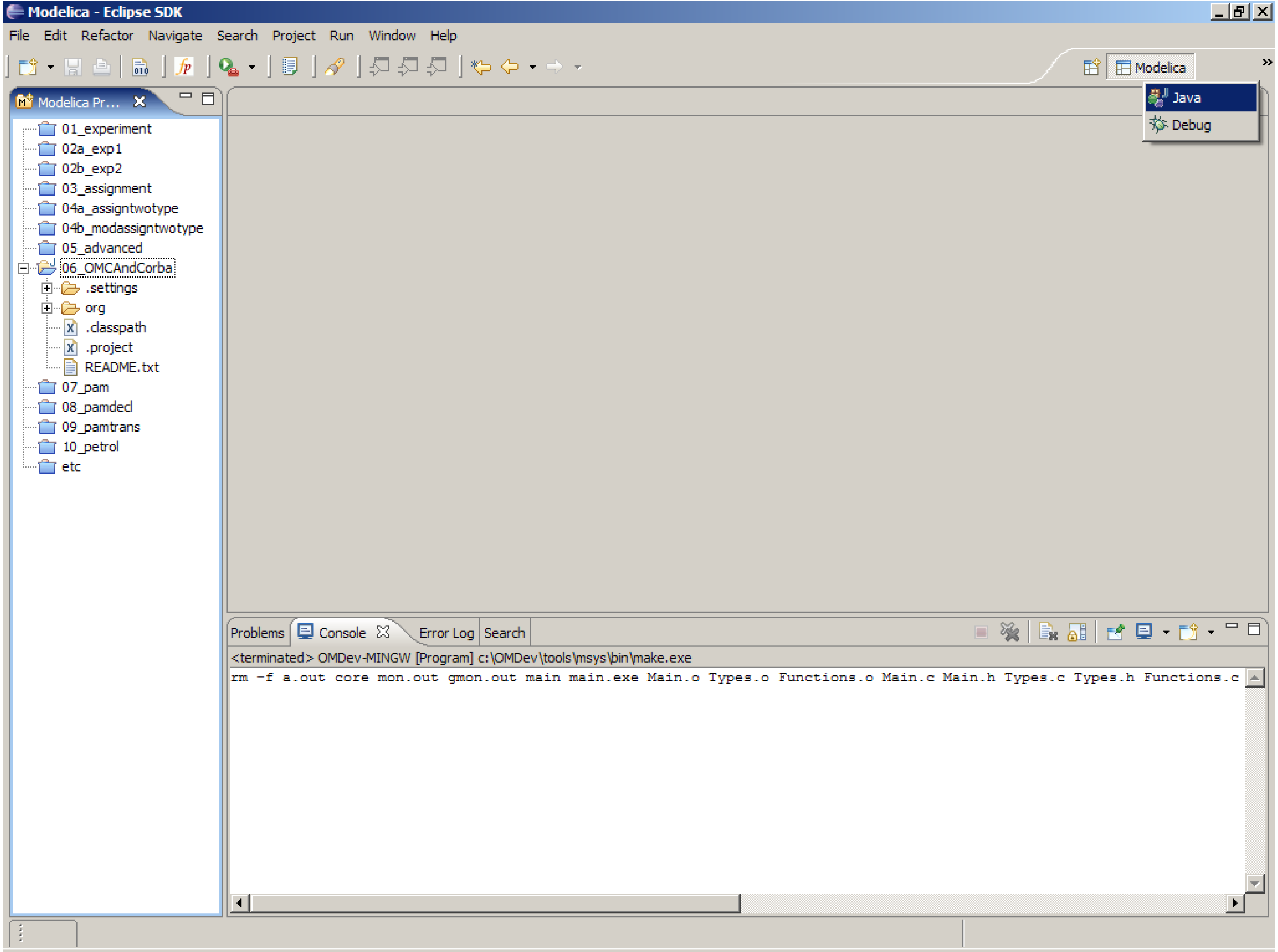
Some of the Corba API functions

`saveModel(A1<string>,A2<cref>)`	Saves the model (A2) in a file given by a string (A1). This call is also in typed API.
`loadFile(A1<string>)`	Loads all models in the file. Also in typed API. Returns list of names of top level classes in the loaded files.
`loadModel(A1<cref>)`	Loads the model (A1) by looking up the correct file to load in `$MODELICAPATH`. Loads all models in that file into the symbol table.
`deleteClass(A1<cref>)`	Deletes the class from the symbol table.
`addComponent(A1<ident>,A2<cref>, A3<cref>,annotate=<expr>)`	Adds a component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
`deleteComponent(A1<ident>, A2<cref>)`	Deletes a component (A1) within a class (A2).
`updateComponent(A1<ident>, A2<cref>, A3<cref>,annotate=<expr>)`	Updates an already existing component with name (A1), type (A2), and class (A3) as arguments. Optional annotations are given with the named argument `annotate`.
`addClassAnnotation(A1<cref>, annotate=<expr>)`	Adds annotation given by A2(in the form `annotate= classmod(...))` to the model definition referenced by A1. Should be used to add Icon Diagram and Documentation annotations.
`getComponents(A1<cref>)`	Returns a list of the component declarations within class A1: `{{Atype,varidA,”commentA”},{Btype,varidB,”commentB”}, {...}}`
`getComponentAnnotations``(A1<cref>)`	Returns a list `{...}` of all annotations of all components in A1, in the same order as the components, one annotation per component.
`getComponentCount(A1<cref>)`	Returns the number (as a string) of components in a class, e.g return `“2”` if there are 2 components.
`getNthComponent(A1<cref>,A2<int>)`	Returns the belonging class, component name and type name of the nth component of a class, e.g. `“A.B.C,R2,Resistor”`, where the first component is numbered 1.

Platforms

- All OpenModelica GUI tools (OMShell, OMNotebook, ...) are developed on top of the Qt4 GUI library, portable between Windows, Linux, Mac
- Both compilers (OMC, MMC) are portable between the three platforms and compiled nightly
  - Windows – main release platform
  - Linux – main development platform
  - Mac – available

Eclipse – Switching to Java Perspective

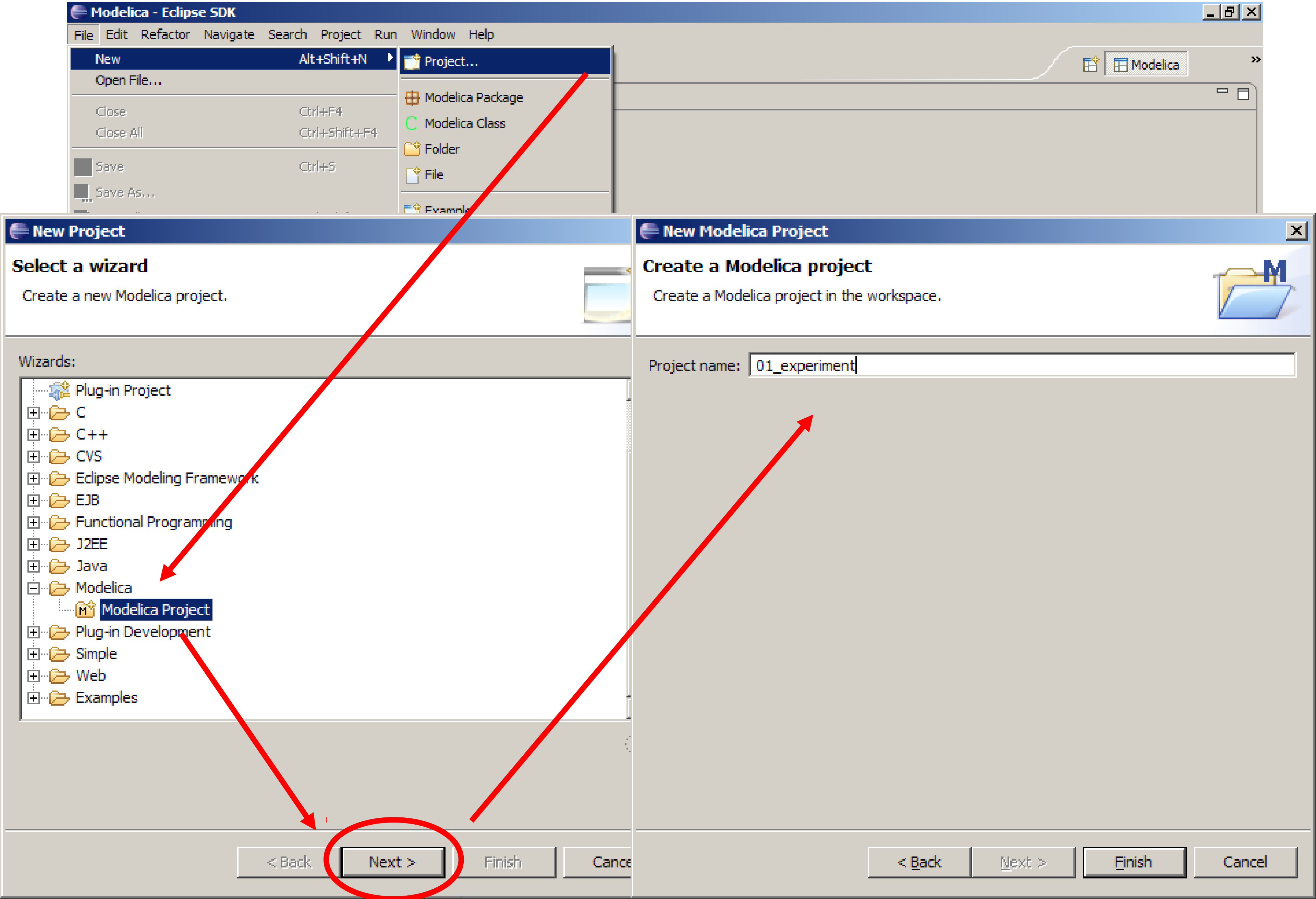


```
<terminated> OMDev-MINGW [Program] c:\OMDev\tools\msys\bin\make.exe
rm -f a.out core mon.out gmon.out main main.exe Main.o Types.o Functions.o Main.c Main.h Types.c Types.h Functions.c
```

Exercises – 07_OMCAndCorba

- See **`README.txt`** in the Eclipse project
- In this exercise you will send commands to the OMC Compiler (**`omc.exe`**) via CORBA
- OMCProxy.java has functionality for
  - starting the omc process if is not already started
    - the starting is a bit different for Windows/Linux
  - sending commands to OMC
  - logging facilities
- If you need clients in C++ or Python check
  - http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html
  - **developer pages**

Eclipse Setup – Creating the projects



New Project

Select a wizard

Create a new Modelica project.

New Modelica Project

Create a Modelica project

Create a Modelica project in the workspace.

Project name: 01_experiment

End

Thank you!

Administrative Question:

What would you like to implement tomorrow in the OpenModelica Compiler?