OMPython

OpenModelica – Python API

User Manual
Version 1.0

March 2012

Table of Contents

1 About OMPython

OMPython – OpenModelica Python Interface is a free, open source, highly portable Python based interactive session handler for Modelica scripting. It provides the modeler with components for creating a complete Modelica modeling, compilation and simulation environment based on the latest OpenModelica library standard available. OMPython is architectured to combine both the solving strategy and model building. So domain experts (people writing the models) and computational engineers (people writing the solver code) can work on one unified tool that is industrially viable for optimization of Modelica models, while offering a flexible platform for algorithm development and research. OMPython v1.0 is not a standalone package, it depends upon the OpenModelica installation.

OMPython v1.0 is implemented in Python using the OmniORB and OmniORBpy – high performance CORBA ORBs for Python and it supports the Modelica Standard Library version 3.2 that is included in the latest OpenModelica (version 1.8.1) installation.

1.1 Features of OMPython

OMPython provides user friendly features like,

- Interactive session handling, parsing, interpretation of commands and Modelica expressions for evaluation, simulation, plotting, etc.
- Interface to the latest OpenModelica API calls.
- Optimized parser results that give control over every element of the output.
- Helper functions to allow manipulation on Nested dictionaries.
- Easy access to the library and testing of OpenModelica commands.

2 Pre-requisites

- **OpenModelica1.8.1**

  OMPython is bundled with OpenModelica from the nightly build revision 11480 and up.

  http://www.openmodelica.org/index.php/download

- **Python 2.7**

  http://www.python.org/download/

3 Using OMPython API

The third party library of OMPython can be created by changing directory to,

\OpenModelica1.8.x\share\omc\scripts\PythonInterface

and running the command,

```
python setup.py install
```

This will install the OMPython library into your *Python.xy/Lib/site-packages*

Now OMPython can be imported and used within Python.

3.1 Test commands

To test the command outputs, simply import the OMPython library from Python and execute the run() method of OMPython.

The module allows you to iteratively send commands to the OMC server and display their output.

For example:

```
C:\>python
>>> import OMPython
>>> OMPython.run()

>>
```

Full example:

```
C:\>python
>>> import OMPython
>>> OMPython.run()

>>loadModel(Modelica)
True
```

3.2 Import as Library

To use the module from within another python program, simply import OMPython from within the using program.

Make use of the execute() function of the OMPython library to send commands to the OMC server.

For example:

```
answer = OMPython.execute(cmd)
```

Full example:

```
# test.py
import OMPython
cmds =
    ["loadModel(Modelica)",
    "model test end test;",
    "loadFile(\"C:/OpenModelica1.8.1/testmodels/BouncingBall.mo\")",
    "getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)",
    "getElementsInfo(Modelica.Electrical.Analog.Basic.Resistor)",
    "simulate(BouncingBall)",
    "plot(h)"]

for cmd in cmds:
    answer = OMPython.execute(cmd)
    print "\nResult:\n%s"%answer
```

3.2.1 Retrieve results from nested dictionaries

Once the result is available from the OMPython.execute(), the OMPython.get() method can be used to retrieve and use specific values inside the dictionaries by simply querying the result dictionary with a string of nested dictionary names (keys).

The query should define the complete nested structure of the dictionary starting from its root.

Syntax:

```
OMPython.get(dict, "dotted.dict.structure")
```

For example:

```
OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)
")
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
```

Full example:

```
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)
")
```

```
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
print "result of get is \n%s" %inner
```

3.2.2 Set values to nested dictionaries

New dictionaries can be added to the existing nested dictionary by using the, OMPyhton.set() method.

Syntax:

```
OMPython.set(dict,"new.dotted.dict.structure", new_value)
```

Note: new_value can be any of the Python supported datatypes.

For example:

```
OMPython.execute("loadModel(Modelica)")
result=
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)
")
value = OMPython.set(result,"SET2.Elements.Line1.Properties.NEW", 1e-05)
```

The OMPython.set() method does not append dictionaries to the existing nest but creates new ones inside the existing. Design your query such that you don't overwrite the dictionaries if you don’t intend to.

Full example:

```
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
result                                                                  =
OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)
")
inner = OMPython.get(result,'SET2.Elements.Line1.Properties')
print "result of get is \n%s" %inner
value = OMPython.set(result,"SET2.Elements.Line1.Properties.NEW", [1,2,3])
print                    "Result::                    \n%s"                  %
OMPython.get(value,'SET2.Elements.Line1.Properties.NEW')
```

3.2.3 Example

test.py - C:/Test/test.py

File Edit Format Run Options Windows Help

```python
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
result = OMPython.execute("getIconAnnotation(Modelica.Electrical.Analog.Basic.Resistor)")

print result

line3_values = OMPython.get(result,'SET2.Elements.Line3.Properties.Values')
print "\n%s" % line3_values

result = OMPython.set(result,'SET2.Elements.Line3.Properties.Values', "Hello OpenModelica!")

value = OMPython.get(result,'SET2.Elements.Line3.Properties.Values')

print "\n%s" % value

OMPython.execute("quit()")
```

Ln: 17 Col: 0

Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321010356166000

{'SET1': {'Set1': [-100.0, -100.0, 100.0, 100.0, True, 0.1, 2.0, 2.0]}, 'SET2':
{'Elements': {'Line3': {'Properties': {'Values': [False, 0, 'LinePattern.Dot', 0
.25, 3, 'Smooth.None'], 'Subset1': {'Set1': [0, -100], 'Set2': [0, -30]}, 'Set1'
: [0.0, 0.0], 'Set2': [127, 0, 0], 'Set3': ['Arrow.None', 'Arrow.None']}}, 'Line
2': {'Properties': {'Values': [True, 0, 'LinePattern.Solid', 0.25, 3, 'Smooth.No
ne'], 'Subset1': {'Set1': [70, 0], 'Set2': [90, 0]}, 'Set1': [0.0, 0.0], 'Set2':
 [0, 0, 255], 'Set3': ['Arrow.None', 'Arrow.None']}}, 'Line1': {'Properties': {'
Values': [True, 0, 'LinePattern.Solid', 0.25, 3, 'Smooth.None'], 'Subset1': {'Se
t1': [-90, 0], 'Set2': [-70, 0]}, 'Set1': [0.0, 0.0], 'Set2': [0, 0, 255], 'Set3
': ['Arrow.None', 'Arrow.None']}}, 'Text2': {'Properties': {'Values': [True, 0,
'LinePattern.Solid', 'FillPattern.None', 0.25, '"%name"', 0, 'TextAlignment.Cent
er'], 'Subset1': {'Set1': [-152, 87], 'Set2': [148, 47]}, 'Set1': [0.0, 0.0], 'S
et2': [0, 0, 255], 'Set3': [0, 0, 0]}}, 'Text1': {'Properties': {'Values': [True
, 0, 'LinePattern.Solid', 'FillPattern.None', 0.25, 0, 'TextAlignment.Center'],
'Subset1': {'Set1': [-144, -40], 'Set2': [142, -72]}, 'Set1': [0.0, 0.0], 'Set2'
: [0, 0, 0], 'Results': {'"R': '%R"'}}}, 'Rectangle1': {'Properties': {'Values':
 [True, 0, 'LinePattern.Solid', 'FillPattern.Solid', 0.25, 'BorderPattern.None',
 0], 'Subset1': {'Set1': [-70, 30], 'Set2': [70, -30]}, 'Set1': [0.0, 0.0], 'Set
2': [0, 0, 255], 'Set3': [255, 255, 255]}}}}}

[False, 0, 'LinePattern.Dot', 0.25, 3, 'Smooth.None']

Hello OpenModelica!

OMC has been Shutdown


C:\Test>
```

4 The OMPython API

4.1 Implementation

4.1.1 Client

The OpenModelica-Python API Interface – OMPython, attempts to mimic the OmShell's style of operations.

OMPython is designed to,

- Initialize the CORBA communication
- Send commands to the Omc server via the CORBA interface
- Receive the string results.
- Use the Parser module to format the results
- Return or display the results.

4.1.2 Parser

Since the results of OMC are retrieved in a String format over CORBA, some housekeeping has to be done to make sure the results are usable in Python easily.

The Parser is designed to do the following,

- Analyze the result string for categorical data.
- Group each category under a category name
- Type cast the data within these categories
- Build a suitable data structure to hold these data so that the results can be easily accessed and used.

4.1.2.1 Understanding the Parsed output

Each command in OpenModelica produces a result that can be categorized according to the statistics of the pattern of data presented in the text. Grammar based parsers were found to be tedious to use because of the complexity of the patterns of data.

The parser just type casts the data without "curly brackets" to the appropriate data type and displays it as the result.

For example:

```
>>getVectorizationLimit()
20

>>getNthInheritedClass(Modelica.Electrical.Analog.Basic.Resistor,1)
Modelica.Electrical.Analog.Interfaces.OnePort
```

However, multiple data types packed within a pair of quotations is always treated as a full string.

For example:

```
>>getModelicaPath()
"C:/OpenModelica1.8.0/lib/omlibrary"
```

4.1.2.1.1 The Dictionary data type in Python:

Dictionaries are found to be a useful datatype to pack data with different datatypes. Dictionaries in Python are indexed by Keys unlike the sequences, which are indexed by a range of numbers.

It is best to think of dictionaries as an unordered set of ***key:value*** pairs, with the requirement that the keys are always unique. The common operation on dictionaries is to store a value associated with a key and retrieve the value using the key. This provides us the flexibility of creating keys at runtime and accessing these values using their keys later. All data within the dictionary are stored inside a named dictionary. An empty dictionary is represented by a pair of braces **{}**.

From the reply of the OMC, the complicated result strings are usually the ones found within the curly braces, in order to make a meaningful categorization of the data within these brackets and to avoid the potential complexities due to creating Dynamic variables, we introduce the following notations that can be used within dictionaries,

- SET
- Set
- Subset
- Element
- Results
- Values

4.1.2.1.2 SET

A SET (*note the capital letters*) is used to group data that belong to the first set of balanced curly brackets. According to the needed semantics of the results, a SET can contain ***Sets, Subsets, Elements, Values and Results***. A SET can also be empty, denoted by **{}**. The SETs are named with an increasing index starting from 1(one). This feature was planned to eliminate the need for dynamic variable creation and having duplicate Keys. The SET belongs within the dictionary, result.

For example:

```
>>strtok("abcbdef","b")
{'SET1': {'Values': ['"a","c","def"']}}
```

```
The command strtok tokenizes the string "abcbdef" at every occurrence of b and
produces a SET with values "a", "c", "def".
```

4.1.2.1.3 Set

A set is used to group all data within the a SET that is enclosed within a pair of balanced {}s. A Set can contain only Values and Elements. A set can also be empty, it can be depicted as **{{}}**, the outer brackets compose a SET, the inner brackets are the Set within the SET.

4.1.2.1.4 Subset

A Subset is a two-level deep set that is found within a SET. A subset can contain multiple Sets within its enclosure.

For example:

{*SET1* **{***Subset1***{***Set1***},{***Set2***},{***Set3***}}}**

4.1.2.1.5 Element

Elements are the data which are grouped within a pair of Parenthesis ( ). As observed from the results string, elements have an element name that describes the data within them, so elements can be grouped by their names. Also, many elements have the same names, so they are indexed by increasing numbers starting from 1(one). Elements have the special property of having one or more Sets and Subsets within them. However, they are in turn enclosed within the SET.

For example:

>>getClassAttributes(test.mymodel)

*{'**SET1**': {'**Elements**': {'**rec1**': {'**Properties**': {'**Results**': {'comment': None, 'res*
triction': 'MODEL', 'startLine': 1, 'partial': False, 'name': '"mymodel"', 'enca
psulated': False, 'startColumn': 14, 'readonly': '"writable"', 'endColumn': 69,
'file': '"<interactive>"', 'endLine': 1, 'final': False}}}}}}

The result contains, a SET with a Element named rec1 which has Properties which are Results (see below) of the element.

4.1.2.1.6 Results

Data that is related by the assignment operator "=", within the SETs are denoted as Results. These assignments cannot be assigned to their actual values unless they are related by a Name = Value relationship. So, they form the sub-dictionary called Results within the Element (for example). Then these values can be related by the ***key:value*** pair relationship.

For example:

>>getClassAttributes(test.mymodel)

*{'SET1': {'Elements': {'rec1': {'Properties': {'**Results': {'comment': None, 'res**
triction': 'MODEL', 'startLine': 1, 'partial': False, 'name': '"mymodel"', 'enca
psulated': False, 'startColumn': 14, 'readonly': '"writable"', 'endColumn': 69,
'file': '"<interactive>"', 'endLine': 1, 'final': False}}}}}}

4.1.2.1.7 Values

Data within any or all of SETs, Sets, Elements and Subsets that are not assignments and separated by commas are grouped together into a list called "Values". The Values list may also be empty, due to Python's representation of a null string "" as {}. Although a Null string is still a Null value, sometimes it is possible to observe data grouped into Values to look like Sets within the Values list.

For example:

>>getNthConnection(Modelica.Electrical.Analog.Examples.ChuaCircuit,2)

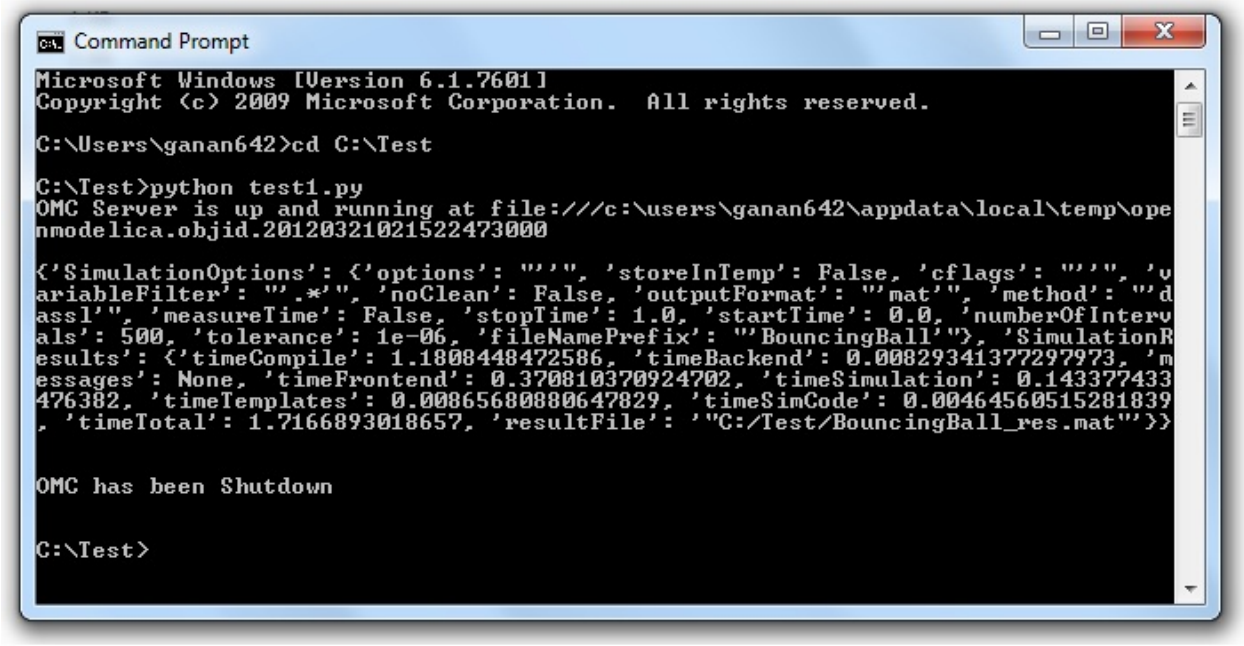
{'SET1': {'Set1': ['G.n', 'Nr.p', {}]}}

4.1.2.1.8 The Simulation results

The simulate() command produces output that has no SET or Set data in it. Instead, for simplicity, it has two dictionaries namely, SimulationResults and SimulationOptions within the result dictionary.

For example:

```
OMPython.execute("simulate(BouncingBall)")
```



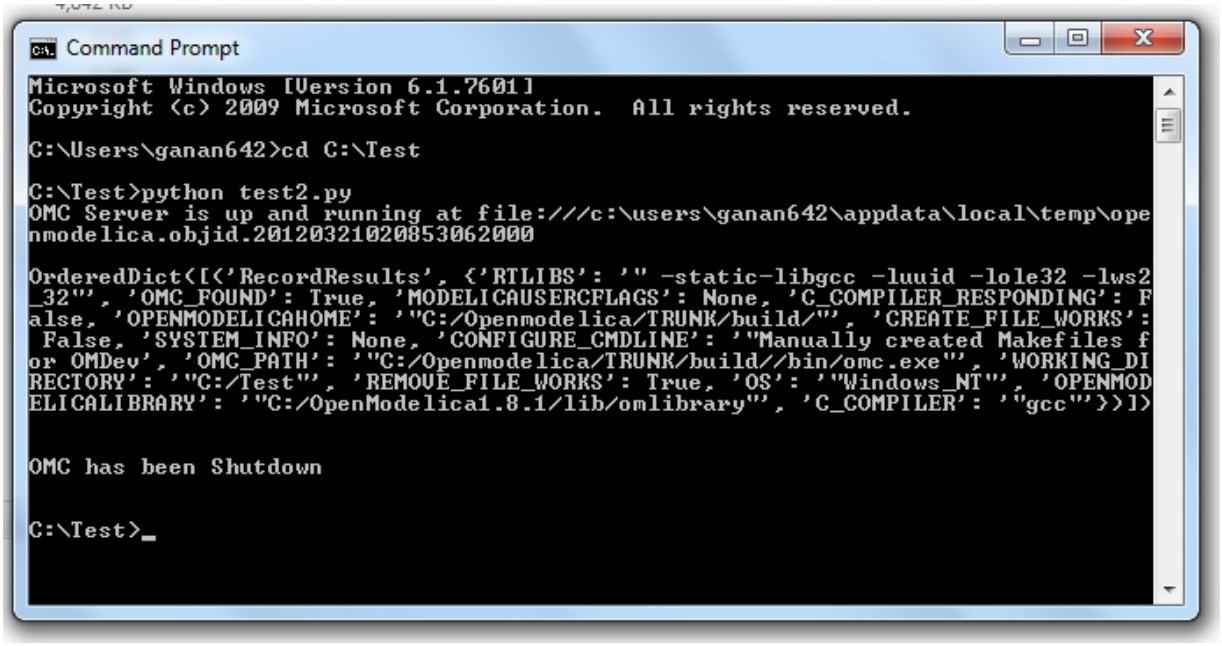
```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test1.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321021522473000

{'SimulationOptions': {'options': "''", 'storeInTemp': False, 'cflags': "''", 'v
ariableFilter': "'.*'", 'noClean': False, 'outputFormat': "'mat'", 'method': "'d
assl'", 'measureTime': False, 'stopTime': 1.0, 'startTime': 0.0, 'numberOfInterv
als': 500, 'tolerance': 1e-06, 'fileNamePrefix': "'BouncingBall'"}, 'SimulationR
esults': {'timeCompile': 1.1808448472586, 'timeBackend': 0.00829341377297973, 'm
essages': None, 'timeFrontend': 0.370810370924702, 'timeSimulation': 0.143377433
476382, 'timeTemplates': 0.00865680880647829, 'timeSimCode': 0.00464560515281839
, 'timeTotal': 1.7166893018657, 'resultFile': '"C:/Test/BouncingBall_res.mat"'}}


OMC has been Shutdown


C:\Test>
```

4.1.2.1.9 Record Constructs

The OpenModelica commands that produce output with Record constructs also do not have SET or Set data within them. The results of the output are packed within the RecordResults dictionary.

For example:

```
OMPython.execute("checkSettings()")
```



```
Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test2.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321020853062000

OrderedDict([('RecordResults', {'RTLIBS': '" -static-libgcc -luuid -lole32 -lws2
_32"', 'OMC_FOUND': True, 'MODELICAUSERCFLAGS': None, 'C_COMPILER_RESPONDING': F
alse, 'OPENMODELICAHOME': '"C:/Openmodelica/TRUNK/build/"', 'CREATE_FILE_WORKS':
 False, 'SYSTEM_INFO': None, 'CONFIGURE_CMDLINE': '"Manually created Makefiles f
or OMDev', 'OMC_PATH': '"C:/Openmodelica/TRUNK/build//bin/omc.exe"', 'WORKING_DI
RECTORY': '"C:/Test"', 'REMOVE_FILE_WORKS': True, 'OS': '"Windows_NT"', 'OPENMOD
ELICALIBRARY': '"C:/OpenModelica1.8.1/lib/omlibrary"', 'C_COMPILER': '"gcc"'})])


OMC has been Shutdown


C:\Test>_
```

5 Examples

5.1 Import as Library

test1.py - C:\Test\test1.py

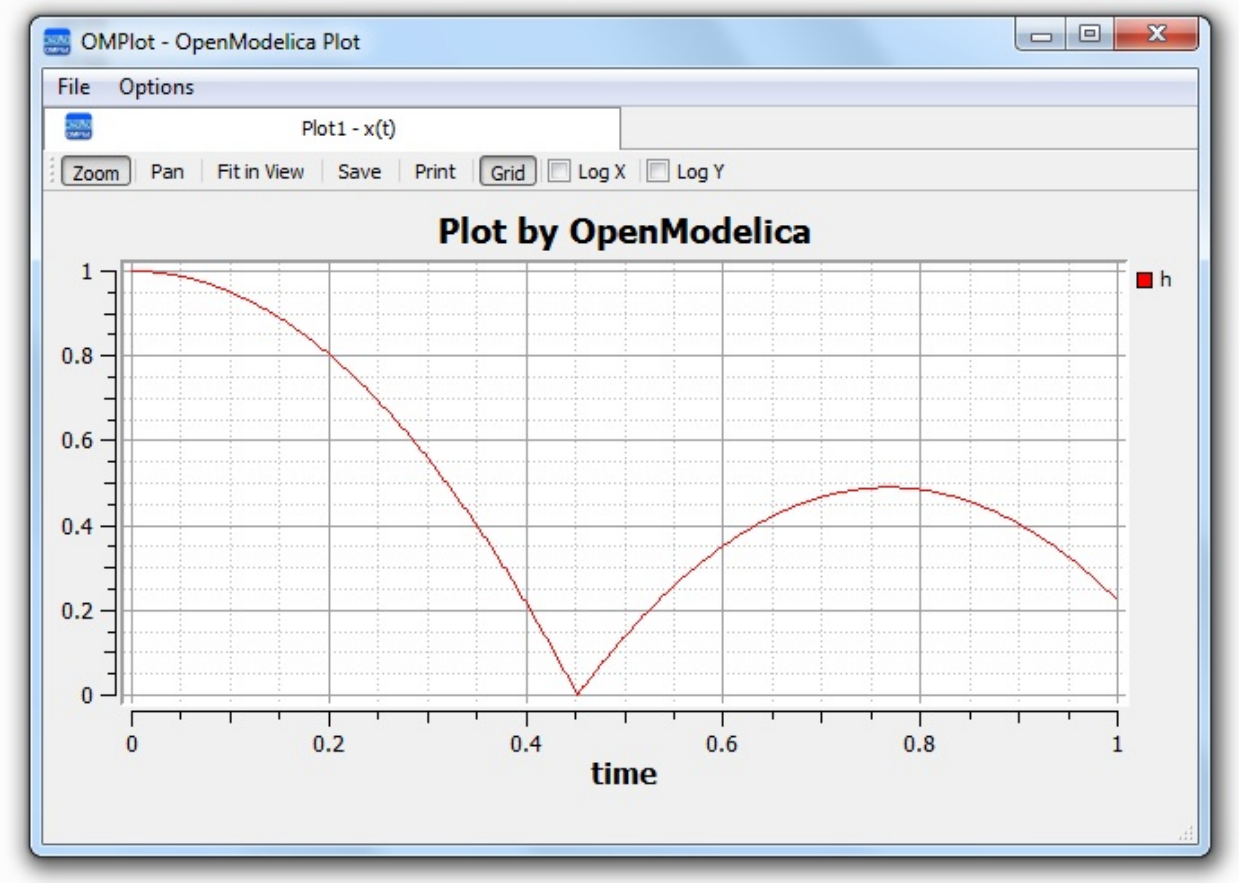
File Edit Format Run Options Windows Help

```python
#test.py
import OMPython

OMPython.execute("loadModel(Modelica)")
OMPython.execute("loadFile(\"C:/OpenModelica1.8.1/testmodels/BouncingBall.mo\")")
result = OMPython.execute("simulate(BouncingBall)")
OMPython.execute("plot(h)")

print result

filename = OMPython.get(result,'SimulationResults.resultFile')
print filename

sim_option_tolerance = OMPython.get(result,'SimulationOptions.tolerance')
print "\nTolerance = %s" % sim_option_tolerance
print "\nType(Tolerance) = %s" % type(sim_option_tolerance)

OMPython.execute("quit()")
```

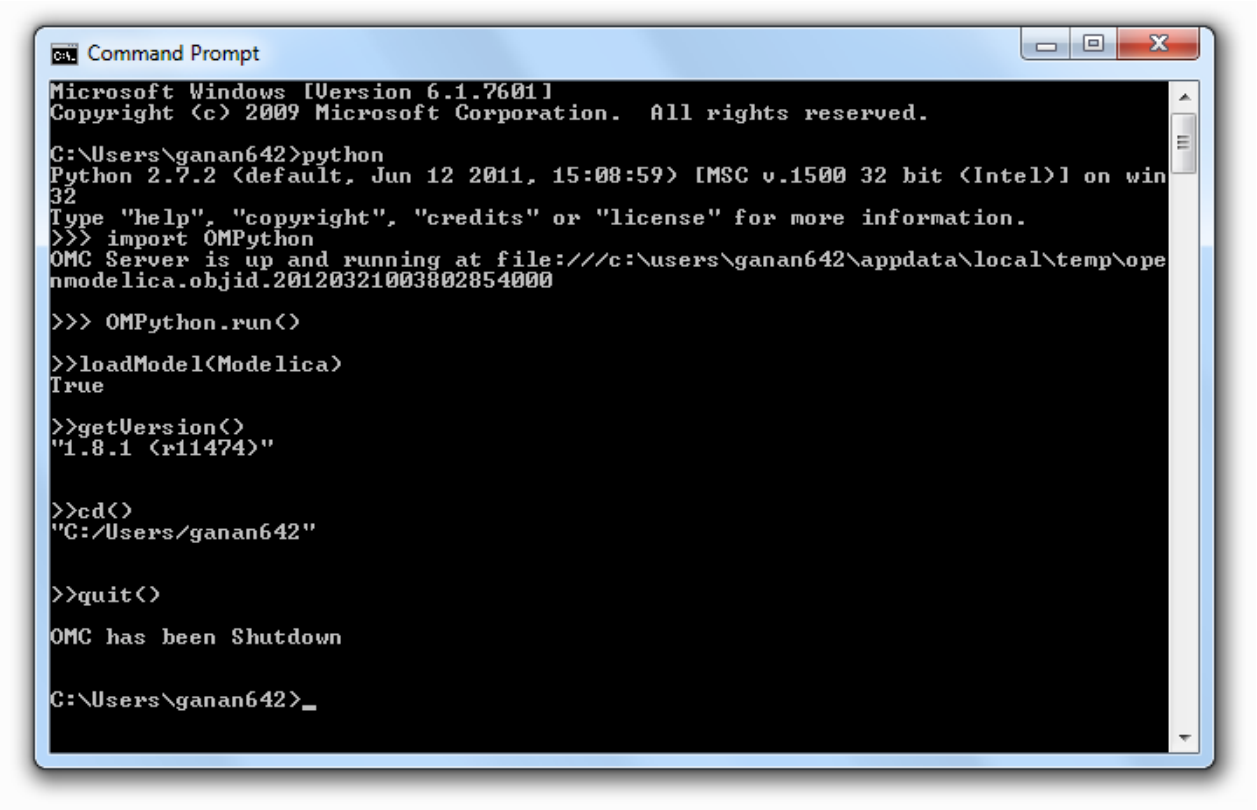
Ln: 17 Col: 0

Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\ganan642>cd C:\Test

C:\Test>python test1.py
OMC Server is up and running at file:///c:\users\ganan642\appdata\local\temp\ope
nmodelica.objid.20120321004143715000

{'SimulationOptions': {'options': "''", 'storeInTemp': False, 'cflags': "''", 'v
ariableFilter': "'.*'", 'noClean': False, 'outputFormat': "'mat'", 'method': "'d
assl'", 'measureTime': False, 'stopTime': 1.0, 'startTime': 0.0, 'numberOfInterv
als': 500, 'tolerance': 1e-06, 'fileNamePrefix': "'BouncingBall'"}, 'SimulationR
esults': {'timeCompile': 1.45518230574149, 'timeBackend': 0.00813173377926215, '
messages': None, 'timeFrontend': 0.369546957259509, 'timeSimulation': 0.09654643
83437193, 'timeTemplates': 0.0088893200355388, 'timeSimCode': 0.0045793933458673
8, 'timeTotal': 1.9429373559315, 'resultFile': '"C:/Test/BouncingBall_res.mat"'}
}
"C:/Test/BouncingBall_res.mat"

Tolerance = 1e-06

Type(Tolerance) = <type 'float'>

OMC has been Shutdown


C:\Test>_
```


OMPlot - OpenModelica Plot
File
Options
Plot1 - x(t)
Zoom
Pan
Fit in View
Save
Print
Grid
Log X
Log Y
Plot by OpenModelica
h
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
time

5.2 Test Commands



6 List of Commands

The following table contains brief descriptions about the commands that are available in the OpenModelica environment.

Name	Description
simulate	Simulate model, optionally setting simulation values. *Inputs*: `TypeName className; Real startTime; Real stopTime; Integer numberOfIntervals; Real outputInterval; String method; Real tolerance; Real fixedStepSize;` *Outputs*: `SimulationResult simRes;`

appendEnvironmentVar	**Interface**
	`function appendEnvironmentVar`
	`  input String var;`
	`  input String value;`
	`  output String result "returns \"error\" if the variable could not be appended";`
	`end appendEnvironmentVar;`

basename	Returns the base name (file part) of a file path. Similar to basename(3), but with the safety of Modelica strings.
	
	Interface
	`function basename`
	`  input String path;`
	`  output String basename;`
	`end basename;`

Cd	change directory to the given path (which may be either relative or absolute) returns the new working directory on success or a message on failure if the given path is the empty string, the function simply returns the current working directory
	
	Interface
	`function cd`
	`  input String newWorkingDirectory = "";`
	`  output String workingDirectory;`
	`end cd;`

checkAllModelsRecursive	**Interface**
	`function checkAllModelsRecursive`
	`  input TypeName className;`
	`  input Boolean checkProtected = false "Checks also protected classes if true";`
	`  output String result;`
	`end checkAllModelsRecursive;`

checkModel	Instantiate model, optimize equations, and report errors.
	
	Interface
	`function checkModel`
	`  input TypeName className;`
	`  output String result;`
	`end checkModel;`

checkSettings	Display some diagnostics
	
	Interface
	`function checkSettings`
	`  output CheckSettingsResult result;`
	`end checkSettings;`

clear	Clears everything: symboltable and variables.
	
	Interface
	`function clear`
	`  output Boolean success;`
	`end clear;`

clearMessages	Clears the error buffer
	
	Interface
	`function clearMessages`
	`  output Boolean success;`
	`end clearMessages;`

clearVariables	Clear all user defined variables.
	
	Interface
	`function clearVariables`
	`  output Boolean success;`
	`end clearVariables;`

closeSimulationResultFile	Closes the current simulation result file. Only needed by Windows. Windows cannot handle reading and writing to the same file from different processes. To allow OMEdit to make successful simulation again on the same file we must close the file after reading the Simulation Result Variables. Even OMEdit only use this API for Windows
	
	Interface
	`function closeSimulationResultFile`
	`  output Boolean success;`
	`end closeSimulationResultFile`

codeToString	**Interface**

	`function codeToString`
	`  input Code className;`
	`  output String string;`
	`end codeToString;`

compareSimulationResults	Compare simulation results
	
	Interface
	`function compareSimulationResults`
	`  input String filename;`
	`  input String reffilename;`
	`  input String logfilename;`
	`  input Real refTol;`
	`  input Real absTol;`
	`  input String[:] vars;`
	`  output String result;`
	`end compareSimulationResults;`

deleteFile	Deletes a file with the given name
	
	Interface
	`function deleteFile`
	`  input String fileName;`
	`  output Boolean success;`
	`end deleteFile;`

dirname	Returns the directory name of a file path. Similar to dirname(3), but with the safety of Modelica strings.
	
	Interface
	`function dirname`
	`  input String path;`
	`  output String dirname;`
	`end dirname;`

dumpXMLDAE	**Interface**
	`function dumpXMLDAE`
	`  input TypeName className;`
	`  input String translationLevel = "flat";`
	`  input Boolean addOriginalIncidenceMatrix = false;`
	`  input Boolean addSolvingInfo = false;`
	`  input Boolean addMathMLCode = false;`
	`  input Boolean dumpResiduals = false;`
	`  input String fileNamePrefix = "<default>" "this is the className in string form by default";`
	`  input Boolean storeInTemp = false;`
	`  output String result[2] "Contents, Message/Filename; why is this an array and not 2 output arguments?";`
	`end dumpXMLDAE;`

Echo	echo(false) disables Interactive output, echo(true) enables it again.

	Interface
	`function echo`
	`  input Boolean setEcho;`
	`  output Boolean newEcho;`
	`end echo;`

generateCode	The input is a function name for which C-code is generated and compiled into a dll/so
	
	Interface
	`function generateCode`
	`  input TypeName className;`
	`  output Boolean success;`
	`end generateCode;`

generateHeader	**Interface**
	`function generateHeader`
	`  input String fileName;`
	`  output Boolean success;`
	`end generateHeader;`

generateSeparateCode	**Interface**
	`function generateSeparateCode`
	`  output Boolean success;`
	`end generateSeparateCode;`

getAlgorithmCount	Counts the number of Algorithm sections in a class
	
	Interface
	`function getAlgorithmCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getAlgorithmCount;`

getAlgorithmItemsCount	Counts the number of Algorithm items in a class
	
	Interface
	`function getAlgorithmItemsCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getAlgorithmItemsCount;`

getAnnotationCount	Counts the number of Annotation sections in a class
	
	Interface
	`function getAnnotationCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getAnnotationCount;`

getAnnotationVersion	**Interface**
	`function getAnnotationVersion`
	`  output String annotationVersion;`
	`end getAnnotationVersion;`

getAstAsCorbaString	Print the whole AST on the CORBA format for records, e.g.
	record Absyn.PROGRAM
	classes = ...,
	within_ = ...,
	globalBuildTimes = ...
	end Absyn.PROGRAM;
	
	Interface
	`function getAstAsCorbaString`
	`  input String fileName = "<interactive>";`
	`  output String result "returns the string if fileName is interactive; else it returns ok or error depending on if writing the file succeeded";`
	`end getAstAsCorbaString;`

getClassComment	**Interface**
	`function getClassComment`
	`  input TypeName cl;`
	`  output String comment;`
	`end getClassComment;`

getClassNames	**Interface**
	`function getClassNames`
	`  input TypeName class_ = Code(AllLoadedClasses);`
	`  input Boolean recursive = false;`
	`  input Boolean qualified = false;`
	`  input Boolean sort = false;`
	`  input Boolean builtin = false "List also builtin classes if true";`
	`  input Boolean showProtected = false "List also protected classes if true";`
	`  output TypeName classNames[:];`
	`end getClassNames;`

getClassesInModelicaPath	**Interface**
	`function getClassesInModelicaPath`
	`  output String classesInModelicaPath;`
	`end getClassesInModelicaPath;`

getCompileCommand	**Interface** `function getCompileCommand`

	`  output String compileCommand;` `end getCompileCommand;`

getDocumentationAnnotation	**Interface**
	`function getDocumentationAnnotation`
	`  input TypeName cl;`
	`  output String out[2] "{info,revision}";`
	`end getDocumentationAnnotation;`

getEnvironmentVar	**Interface**
	`function getEnvironmentVar`
	`  input String var;`
	`  output String value "returns empty string on failure";`
	`end getEnvironmentVar;`

getEquationCount	Counts the number of Equation sections in a class
	Interface
	`function getEquationCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getEquationCount;`

getEquationItemsCount	Counts the number of Equation items in a class
	Interface
	`function getEquationItemsCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getEquationItemsCount;`
	ounts the number of Equation items in a class

getErrorString	[file.mo:n:n-n:n:b] Error: message
	Interface
	`function getErrorString`
	`  output String errorString;`
	`end getErrorString;`

getImportCount	Counts the number of Import sections in a class
	Interface
	`function getImportCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getImportCount;`

getInitialAlgorithmCount	Counts the number of Initial Algorithm sections in a class **Interface**

	`function getInitialAlgorithmCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getInitialAlgorithmCount;`

getInitialAlgorithmItemsCount	Counts the number of Initial Algorithm items in a class
	
	Interface
	`function getInitialAlgorithmItemsCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getInitialAlgorithmItemsCount;`

getInitialEquationCount	Counts the number of Initial Equation sections in a class
	
	Interface
	`function getInitialEquationCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getInitialEquationCount;`

getInitialEquationItemsCount	Counts the number of Initial Equation items in a class
	
	Interface
	`function getInitialEquationItemsCount`
	`  input TypeName class_;`
	`  output Integer count;`
	`end getInitialEquationItemsCount;`

getInstallationDirectoryPath	This returns OPENMODELICAHOME if it is set; on some platforms the default path is returned if it is not set.
	
	Interface
	`function getInstallationDirectoryPath`
	`  output String installationDirectoryPath;`
	`end getInstallationDirectoryPath;`

getLanguageStandard	**Interface**
	`function getLanguageStandard`
	`  output String outVersion;`
	`end getLanguageStandard;`

getMessagesString	see getErrorString()
	
	Interface
	`function getMessagesString`
	`  output String messagesString;`
	`end getMessagesString;`

getMessagesStringInter	{{[file.mo:n:n-n:n:b] Error: message, TRANSLATION, Error, code}}

nal	**Interface**
	`function getMessagesStringInternal`
	`  output ErrorMessage[:] messagesString;`
	`end getMessagesStringInternal;`

getModelicaPath	See loadModel() for a description of what the MODELICAPATH is used for.
	
	Interface
	`function getModelicaPath`
	`  output String modelicaPath;`
	`end getModelicaPath;`

getNoSimplify	**Interface**
	`function getNoSimplify`
	`  output Boolean noSimplify;`
	`end getNoSimplify;`

getNthAlgorithm	Returns the Nth Algorithm section
	
	Interface
	`function getNthAlgorithm`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthAlgorithm;`

getNthAlgorithmItem	Returns the Nth Algorithm Item
	
	Interface
	`function getNthAlgorithmItem`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthAlgorithmItem;`

getNthAnnotationString	Returns the Nth Annotation section as string
	
	Interface
	`function getNthAnnotationString`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthAnnotationString;`

getNthEquation	Returns the Nth Equation section
	
	Interface
	`function getNthEquation`
	`  input TypeName class_;`
	`  input Integer index;`

	`  output String result;` `end getNthEquation;`

getNthEquationItem	Returns the Nth Equation Item
	
	Interface
	`function getNthEquationItem`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthEquationItem;`

getNthImport	Returns the Nth Import as string
	
	Interface
	`function getNthImport`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String out[3] "{\"Path\",\"Id\",\"Kind\"}";`
	`end getNthImport;`

getNthInitialAlgorithm	Returns the Nth Initial Algorithm section
	
	Interface
	`function getNthInitialAlgorithm`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthInitialAlgorithm;`

getNthInitialAlgorithmItem	Returns the Nth Initial Algorithm Item
	
	Interface
	`function getNthInitialAlgorithmItem`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthInitialAlgorithmItem;`

getNthInitialEquation	Returns the Nth Initial Equation section
	
	Interface
	`function getNthInitialEquation`
	`  input TypeName class_;`
	`  input Integer index;`
	`  output String result;`
	`end getNthInitialEquation;`

getNthInitialEquationItem	Returns the Nth Initial Equation Item **Interface** `function getNthInitialEquationItem`

	input TypeName class_; input Integer index; output String result; end getNthInitialEquationItem; ns the Nth Initial Equation Item
getOrderConnections	**Interface** function getOrderConnections output Boolean orderConnections; end getOrderConnections;
getPackages	**Interface** function getPackages input TypeName class_ = Code(AllLoadedClasses); output TypeName classNames[:]; end getPackages;
getPlotSilent	**Interface** function getPlotSilent output Boolean plotSilent; end getPlotSilent;
getSettings	**Interface** function getSettings output String settings; end getSettings;
getShowAnnotations	**Interface** function getShowAnnotations output Boolean show; end getShowAnnotations;
getSourceFile	**Interface** function getSourceFile input TypeName class_; output String filename "empty on failure"; end getSourceFile;
getTempDirectoryPath	**Interface** function getTempDirectoryPath output String tempDirectoryPath; end getTempDirectoryPath;
getVectorizationLimit	**Interface**

	`function getVectorizationLimit`
	`  output Integer vectorizationLimit;`
	`end getVectorizationLimit;`

getVersion	Returns the version of the Modelica compiler
	
	Interface
	`function getVersion`
	`  input TypeName cl = Code(OpenModelica);`
	`  output String version;`
	`end getVersion;`

help	Display the OpenModelica help text
	
	Interface
	`function help`
	`  output String helpText;`
	`end help;`

iconv	The iconv() function converts one multibyte characters from one character set to another.
	See man (3) iconv for more information.
	
	Interface
	`function iconv`
	`  input String string;`
	`  input String from;`
	`  input String to = "UTF-8";`
	`  output String result;`
	`end iconv;`

importFMU	Imports the Functional Mockup Unit
	Example command:
	importFMU("A.fmu");
	
	Interface
	`function importFMU`
	`  input String filename "the fmu file name";`
	`  input String workdir = "./" "The output directory for imported FMU files. <default> will put the files to current working directory.";`
	`  output Boolean success "Returns true on success";`
	`end importFMU;`

instantiate Model	Instantiate model, resulting in a .mof file of flattened Modelica.
	
	Interface
	`function instantiateModel`
	`  input TypeName className;`
	`  output String result;`
	`end instantiateModel;`

isModel	Returns true if the given class has restriction model.
	
	Interface
	`function isModel`
	`  input TypeName cl;`
	`  output Boolean b;`
	`end isModel;`

isPackage	Returns true if the given class is a package.
	
	Interface
	`function isPackage`
	`  input TypeName cl;`
	`  output Boolean b;`
	`end isPackage;`

isPartial	Returns true if the given class is partial.
	
	Interface
	`function isPartial`
	`  input TypeName cl;`
	`  output Boolean b;`
	`end isPartial;`

list	Pretty-prints a class definition.
	
	Syntax
	`list(Modelica.Math.sin)`
	`list(Modelica.Math.sin,interfaceOnly=true)`
	
	Interface
	`function list`
	`  input TypeName class_ = Code(AllLoadedClasses);`
	`  input Boolean interfaceOnly = false;`
	`  input Boolean shortOnly = false "only short class definitions";`
	`  output String contents;`
	`end list;`

listVariables	Lists the names of the active variables in the scripting environment.
	
	Interface
	`function listVariables`
	`  output TypeName variables[:];`
	`end listVariables;`

loadFile	load file (*.mo) and merge it with the loaded AST
	
	Interface
	`function loadFile`
	`  input String fileName;`

	`  output Boolean success;` `end loadFile;`

loadFileInteractive	**Interface**
	`function loadFileInteractive`
	`  input String filename;`
	`  output TypeName names[:];`
	`end loadFileInteractive;`

loadFileInteractiveQualified	**Interface**
	`function loadFileInteractiveQualified`
	`  input String filename;`
	`  output TypeName names[:];`
	`end loadFileInteractiveQualified;`

loadModel	Loads a Modelica library.
	
	Syntax
	`loadModel``(Modelica)`
	`loadModel``(Modelica,{"3.2"})`
	
	Interface
	`function loadModel`
	`  input TypeName className;`
	`  input String[:] priorityVersion = {"default"};`
	`  output Boolean success;`
	`end loadModel;`

loadString	Parses the data and merges the resulting AST with the loaded AST.
	If a filename is given, it is used to provide error-messages as if the string was read in binary format from a file with the same name.
	The file is converted to UTF-8 from the given character set.
	
	Interface
	`function loadString`
	`  input String data;`
	`  input String filename = "<interactive>";`
	`  input String encoding = "UTF-8";`
	`  output Boolean success;`
	`end loadString;`

parseFile	**Interface**
	`function parseFile`
	`  input String filename;`
	`  output TypeName names[:];`
	`end parseFile;`

parseString	**Interface**
	`function parseString`
	`  input String data;`
	`  input String filename = "<interactive>";`
	`  output TypeName names[:];`
	`end parseString;`

plot	Launches a plot window using OMPlot. Returns true on success.
	Don't require sendData support.
	
	Example command sequences:
	simulate(A);plot({x,y,z});
	simulate(A);plot(x, externalWindow=true);
	simulate(A,fileNamePrefix="B");simulate(C);plot(z,"B.mat",legend=false);
	
	Interface
	`function plot`
	`  input VariableNames vars "The variables you want to plot";`
	`  input Boolean externalWindow = false "Opens the plot in a new plot window";`
	`  input String fileName = "<default>" "The filename containing the variables. <default> will read the last simulation result";`
	`  input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`  input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`  input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`
	`  input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`  input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`  input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";`
	`  input String yLabel = "" "This text will be used as the vertical label in the diagram.";`
	`  input Real xRange[2] = {0.0,0.0} "Determines the horizontal interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  output Boolean success "Returns true on success";`
	`  output String[:] result "Returns list i.e {\"_omc_PlotResult\",\"<fileName>\",\"<title>\",\"<legend>\",\"<grid>\",\"<PlotType>\",\"<logX>\",\"<logY>\",\"<xLabel>\",\"<yLabel>\",\"<xRange>\",\"<yRange>\",\"<PlotVariables>\"}";`
	`end plot;`

plot2	Uses the Java-based plot window (ptplot.jar) to launch a plot, similar to the plot() command. This command accepts fewer options, but works even when OpenModelica was not compiled with sendData support.

	Example command sequences:
	simulate(A);plot2({x,y});
	simulate(A,fileNamePrefix="B");simulate(C);plot2(x,"B.mat");
	
	Interface
	`function plot2`
	`  input VariableNames vars;`
	`  input String fileName = "<default>";`
	`  output Boolean success "Returns true on success";`
	`end plot2;`

plotAll	Works in the same way as plot(), but does not accept any variable names as input. Instead, all variables are part of the plot window.
	
	Example command sequences:
	simulate(A);plotAll();
	simulate(A);plotAll(externalWindow=true);
	simulate(A,fileNamePrefix="B");simulate(C);plotAll(x,"B.mat");
	
	Interface
	`function plotAll`
	`  input Boolean externalWindow = false "Opens the plot in a new plot window";`
	`  input String fileName = "<default>" "The filename containing the variables. <default> will read the last simulation result";`
	`  input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`  input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`  input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`
	`  input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`  input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`  input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";`
	`  input String yLabel = "" "This text will be used as the vertical label in the diagram.";`
	`  input Real xRange[2] = {0.0,0.0} "Determines the horizontal interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  output Boolean success "Returns true on success";`
	`  output String[:] result "Returns list i.e {\"_omc_PlotResult\",\"<fileName>\",\"<title>\",\"<legend>\",\"<grid>\",\"<PlotType>\",\"<logX>\",\"<logY>\",\"<xLabel>\",\"<yLabel>\",\"<xRange>\",\"<yRange>\",\"<PlotVariables>\"}";`
	`end plotAll;`

plotParametric	Launches a plotParametric window using OMPlot. Returns true on success.
	Don't require sendData support.
	
	Example command sequences:
	simulate(A);plotParametric2(x,y);
	simulate(A);plotParametric2(x,y, externalWindow=true);
	
	Interface
	`function plotParametric`
	`  input VariableName xVariable;`
	`  input VariableName yVariable;`
	`  input Boolean externalWindow = false "Opens the plot in a new plot window";`
	`  input String fileName = "<default>" "The filename containing the variables. <default> will read the last simulation result";`
	`  input String title = "Plot by OpenModelica" "This text will be used as the diagram title.";`
	`  input Boolean legend = true "Determines whether or not the variable legend is shown.";`
	`  input Boolean grid = true "Determines whether or not a grid is shown in the diagram.";`
	`  input Boolean logX = false "Determines whether or not the horizontal axis is logarithmically scaled.";`
	`  input Boolean logY = false "Determines whether or not the vertical axis is logarithmically scaled.";`
	`  input String xLabel = "time" "This text will be used as the horizontal label in the diagram.";`
	`  input String yLabel = "" "This text will be used as the vertical label in the diagram.";`
	`  input Real xRange[2] = {0.0,0.0} "Determines the horizontal interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  input Real yRange[2] = {0.0,0.0} "Determines the vertical interval that is visible in the diagram. {0,0} will select a suitable range.";`
	`  output Boolean success "Returns true on success";`
	`  output String[:] result "Returns list i.e {\"_omc_PlotResult\",\"<fileName>\",\"<title>\",\"<legend>\",\"<grid>\",\"<PlotType>\",\"<logX>\",\"<logY>\",\"<xLabel>\",\"<yLabel>\",\"<xRange>\",\"<yRange>\",\"<PlotVariables>\"}";`
	`end plotParametric;`

plotParametric2	Plots the y-variables as a function of the x-variable.
	
	Example command sequences:
	simulate(A);plotParametric2(x,y);
	simulate(A,fileNamePrefix="B");simulate(C);plotParametric2(x,{y1,y2,y3},"B.mat");
	
	Interface
	`function plotParametric2`
	`  input VariableName xVariable;`
	`  input VariableNames yVariables;`

	`  input String fileName = "<default>";`
	`  output Boolean success "Returns true on success";`
	`end plotParametric2;`

readFile	The contents of the given file are returned.
	Note that if the function fails, the error message is returned as a string instead of multiple output or similar.
	
	Interface
	`function readFile`
	`  input String fileName;`
	`  output String contents;`
	`end readFile;`

readFileNoNumeric	Returns the contents of the file, with anything resembling a (real) number stripped out, and at the end adding:
	Filter count from number domain: n.
	This should probably be changed to multiple outputs; the filtered string and an integer.
	Does anyone use this API call?
	
	Interface
	`function readFileNoNumeric`
	`  input String fileName;`
	`  output String contents;`
	`end readFileNoNumeric;`

readFilePostprocessLineDirective	Searches lines for the #modelicaLine directive. If it is found, all lines up until the next #modelicaLine or #endModelicaLine are put on a single file, following a #line linenumber "filename" line.
	This causes GCC to output an executable that we can set breakpoints in and debug.
	Note: You could use a stack to keep track of start/end of #modelicaLine and match them up. But this is not really desirable since that will cause extra breakpoints for the same line (you would get breakpoints before and after each case if you break on a match-expression, etc).
	
	Interface
	`function readFilePostprocessLineDirective`
	`  input String fileName;`
	`  output String out;`
	`end readFilePostprocessLineDirective;`

readFileShowLineNumbers	Prefixes each line in the file with <n>:, where n is the line number.
	Note: Scales O(n^2)
	
	Interface
	`function readFileShowLineNumbers`
	`  input String fileName;`
	`  output String out;`
	`end readFileShowLineNumbers;`

readSimulationResult	Reads a result file, returning a matrix corresponding to the variables and size given.
	
	Interface
	`function readSimulationResult`
	`  input String filename;`
	`  input VariableNames variables;`
	`  input Integer size = 0 "0=read any size... If the size is not the same as the result-file, this function fails";`
	`  output Real result[:,:];`
	`end readSimulationResult;`

readSimulationResultSize	The number of intervals that are present in the output file
	
	Interface
	`function readSimulationResultSize`
	`  input String fileName;`
	`  output Integer sz;`
	`end readSimulationResultSize;`

readSimulationResultVars	Returns the variables in the simulation file; you can use val() and plot() commands using these names
	
	Interface
	`function readSimulationResultVars`
	`  input String fileName;`
	`  output String[:] vars;`
	`end readSimulationResultVars;`

Regex	Sets the error buffer and returns -1 if the regex does not compile.
	
	The returned result is the same as POSIX regex():
	The first value is the complete matched string
	The rest are the substrings that you wanted.
	For example:
	regex(lorem," \([A-Za-z]*\) \([A-Za-z]*\) ",maxMatches=3)
	=> {" ipsum dolor ","ipsum","dolor"}
	This means if you have n groups, you want maxMatches=n+1
	
	Interface
	`function regex`
	`  input String str;`
	`  input String re;`
	`  input Integer maxMatches = 1 "The maximum number of matches that will be returned";`
	`  input Boolean extended = true "Use POSIX extended or regular syntax";`
	`  input Boolean caseInsensitive = false;`
	`  output Integer numMatches "-1 is an error, 0 means no match, else returns a number 1..maxMatches";`
	`  output String matchedSubstrings[maxMatches] "unmatched strings are returned as empty";`

	`end regex;`
`regexBool`	Returns true if the string matches the regular expression

`regularFileExists`	The contents of the given file are returned.
	Note that if the function fails, the error message is returned as a string instead of multiple output or similar.
	
	Interface
	`function regularFileExists`
	`  input String fileName;`
	`  output Boolean exists;`
	`end regularFileExists;`

`reopenStandardStream`	**Interface**
	`function reopenStandardStream`
	`  input StandardStream _stream;`
	`  input String filename;`
	`  output Boolean success;`
	`end reopenStandardStream;`

`runScript`	Runs the mos-script specified by the filename.
	
	Interface
	`function runScript`
	`  input String fileName "*.mos";`
	`  output String result;`
	`end runScript;`

`Save`	**Interface**
	`function save`
	`  input TypeName className;`
	`  output Boolean success;`
	`end save;`

`saveAll`	Save the entire loaded AST to file
	
	Interface
	`function saveAll`
	`  input String fileName;`
	`  output Boolean success;`
	`end saveAll;`

`saveModel`	Save class definition in a file.
	
	Interface
	`function saveModel`
	`  input String fileName;`
	`  input TypeName className;`
	`  output Boolean success;`

	`end saveModel;`

saveTotalModel	Save total class definition into file of a class.
	
	***Inputs*:** `String fileName; TypeName className`
	
	***Outputs*:** `Boolean res;`
	
	Interface
	`function saveTotalModel`
	`  input String fileName;`
	`  input TypeName className;`
	`  output Boolean success;`
	`end saveTotalModel;`

saveTotalSCode	**Interface**
	`function saveTotalSCode`
	`  input String fileName;`
	`  input TypeName className;`
	`  output Boolean success;`
	`end saveTotalSCode;`

setAnnotationVersion	**Interface**
	`function setAnnotationVersion`
	`  input String annotationVersion;`
	`  output Boolean success;`
	`end setAnnotationVersion;`

setCXXCompiler	**Interface**
	`function setCXXCompiler`
	`  input String compiler;`
	`  output Boolean success;`
	`end setCXXCompiler;`

setClassComment	**Interface**
	`function setClassComment`
	`  input TypeName class_;`
	`  input String filename;`
	`  output Boolean success;`
	`end setClassComment;`

setCommandLineOptions	The input is a regular command-line flag given to OMC, e.g. +d=failtrace or +g=MetaModelica
	
	Interface
	`function setCommandLineOptions`
	`  input String option;`

	`  output Boolean success;` `end setCommandLineOptions;`

setCompileCommand	**Interface**
	`function setCompileCommand`
	`  input String compileCommand;`
	`  output Boolean success;`
	`end setCompileCommand;`

setCompiler	**Interface**
	`function setCompiler`
	`  input String compiler;`
	`  output Boolean success;`
	`end setCompiler;`

setCompilerFlags	**Interface**
	`function setCompilerFlags`
	`  input String compilerFlags;`
	`  output Boolean success;`
	`end setCompilerFlags;`

setCompilerPath	**Interface**
	`function setCompilerPath`
	`  input String compilerPath;`
	`  output Boolean success;`
	`end setCompilerPath;`

setDebugFlags	example input: failtrace,-noevalfunc
	
	Interface
	`function setDebugFlags`
	`  input String debugFlags;`
	`  output Boolean success;`
	`end setDebugFlags;`

setEnvironmentVar	**Interface**
	`function setEnvironmentVar`
	`  input String var;`
	`  input String value;`
	`  output Boolean success;`
	`end setEnvironmentVar;`

setIndexReductionMethod	example input: dummyDerivative
	
	Interface
	`function setIndexReductionMethod`
	`  input String method;`

	`  output Boolean success;` `end setIndexReductionMethod;`

setInstallationDirectoryPath	Sets the OPENMODELICAHOME environment variable. Use this method instead of setEnvironmentVar
	
	Interface
	`function setInstallationDirectoryPath`
	`  input String installationDirectoryPath;`
	`  output Boolean success;`
	`end setInstallationDirectoryPath;`

setLanguageStandard	**Interface**
	`function setLanguageStandard`
	`  input String inVersion;`
	`  output Boolean success;`
	`end setLanguageStandard;`

setLinker	**Interface**
	`function setLinker`
	`  input String linker;`
	`  output Boolean success;`
	`end setLinker;`

setLinkerFlags	**Interface**
	`function setLinkerFlags`
	`  input String linkerFlags;`
	`  output Boolean success;`
	`end setLinkerFlags;`

setModelicaPath	See loadModel() for a description of what the MODELICAPATH is used for.
	
	Interface
	`function setModelicaPath`
	`  input String modelicaPath;`
	`  output Boolean success;`
	`end setModelicaPath;`

setNoSimplify	**Interface**
	`function setNoSimplify`
	`  input Boolean noSimplify;`
	`  output Boolean success;`
	`end setNoSimplify;`

setOrderConnections	**Interface**
	`function setOrderConnections`
	`  input Boolean orderConnections;`

	`  output Boolean success;` `end setOrderConnections;`

setPastOptModules	example input: lateInline,inlineArrayEqn,removeSimpleEquations
	
	Interface
	`function setPastOptModules`
	`  input String modules;`
	`  output Boolean success;`
	`end setPastOptModules;`

setPlotCommand	**Interface**
	`function setPlotCommand`
	`  input String plotCommand;`
	`  output Boolean success;`
	`end setPlotCommand;`

setPlotSilent	**Interface**
	`function setPlotSilent`
	`  input Boolean silent;`
	`  output Boolean success;`
	`end setPlotSilent;`

setPreOptModules	example input: removeFinalParameters,removeSimpleEquations,expandDerOperator
	
	Interface
	`function setPreOptModules`
	`  input String modules;`
	`  output Boolean success;`
	`end setPreOptModules;`

setShowAnnotations	**Interface**
	`function setShowAnnotations`
	`  input Boolean show;`
	`  output Boolean success;`
	`end setShowAnnotations;`

setSourceFile	**Interface**
	`function setSourceFile`
	`  input TypeName class_;`
	`  input String filename;`
	`  output Boolean success;`
	`end setSourceFile;`

setTempDirectoryPath	**Interface**
	`function setTempDirectoryPath`
	`  input String tempDirectoryPath;`

	`  output Boolean success;` `end setTempDirectoryPath;`

setVectorizationLimit	**Interface**
	`function setVectorizationLimit`
	`  input Integer vectorizationLimit;`
	`  output Boolean success;`
	`end setVectorizationLimit;`

solveLinearSystem	Solve A*X = B, using dgesv or lp_solve (if any variable in X is integer)
	Returns for solver dgesv: info>0: Singular for element i. info<0: Bad input.
	
	Interface
	`function solveLinearSystem`
	`  input Real[size(B, 1),size(B, 1)] A;`
	`  input Real[:] B;`
	`  input LinearSystemSolver solver = LinearSystemSolver.dgesv;`
	`  input Integer[:] isInt = {-1} "list of indices that are integers";`
	`  output Real[size(B, 1)] X;`
	`  output Integer info;`
	`end solveLinearSystem;`

strictRMLCheck	Checks if any loaded function
	
	Interface
	`function strictRMLCheck`
	`  output String message "empty if there was no problem";`
	`end strictRMLCheck;`

stringReplace	**Interface**
	`function stringReplace`
	`  input String str;`
	`  input String source;`
	`  input String target;`
	`  output String res;`
	`end stringReplace;`

Strtok	Splits the strings at the places given by the token, for example:
	strtok("abcbdef","b") => {"a","c","def"}
	
	Interface
	`function strtok`
	`  input String string;`
	`  input String token;`
	`  output String[:] strings;`
	`end strtok;`

System	Similar to system(3). Executes the given command in the system shell.

	Interface
	`function system`
	`  input String callStr "String to call: bash -c $callStr";`
	`  output Integer retval "Return value of the system call; usually 0 on success";`
	`end system;`

translateGraphics	**Interface**
	`function translateGraphics`
	`  input TypeName className;`
	`  output String result;`
	`end translateGraphics;`

typeNameString	**Interface**
	`function typeNameString`
	`  input TypeName cl;`
	`  output String out;`
	`end typeNameString;`

typeNameStrings	**Interface**
	`function typeNameStrings`
	`  input TypeName cl;`
	`  output String out[:];`
	`end typeNameStrings;`

typeOf	**Interface**
	`function typeOf`
	`  input VariableName variableName;`
	`  output String result;`
	`end typeOf;`

uriToFilename	Handles modelica:// and file:// URI's. The result is an absolute path on the local system. The result depends on the current MODELICAPATH. Returns the empty string on failure.
	
	Interface
	`function uriToFilename`
	`  input String uri;`
	`  output String filename;`
	`end uriToFilename;`

Val	Works on the filename pointed to by the scripting variable currentSimulationResult. The result is the value of the variable at a certain time point. For parameters, any time may be given. For variables the startTime<=time<=stopTime needs to hold. On error, nan (Not a Number) is returned and the error buffer contains the message.

	Interface
	`function val`
	`  input VariableName var;`
	`  input Real time;`
	`  output Real valAtTime;`
	`end val;`

verifyCompiler	**Interface**
	`function verifyCompiler`
	`  output Boolean compilerWorks;`
	`end verifyCompiler;`

visualize	Uses the 3D visualization package, SimpleVisual.mo, to visualize the model. See chapter 3.4 (3D Animation) of the OpenModelica System Documentation for more details.
	Writes the visulizations objects into the file "model_name.visualize"
	Don't require sendData support.
	
	Example command sequence:
	
	simulate(A,outputFormat="mat");visualize(A);visualize(A,"B.mat");visualize(A,"B.mat", true);
	
	Interface
	`function visualize`
	`  input TypeName className;`
	`  input Boolean externalWindow = false "Opens the visualize in a new window";`
	`  input String fileName = "<default>" "The filename containing the variables. <default> will read the last simulation result";`
	`  output Boolean success "Returns true on success";`
	`end visualize;`

writeFile	Write the data to file. Returns true on success.
	
	Interface
	`function writeFile`
	`  input String fileName;`
	`  input String data;`
	`  input Boolean append = false;`
	`  output Boolean success;`
	`end writeFile;`

6.1 Additional resources

For a list of OMC APIs with their syntax and examples, read the document

http://www.openmodelica.org/download/OMC_API-HowTo.pdf

The new API function calls are constantly updated at,

http://build.openmodelica.org/Documentation/OpenModelica.Scripting.html