

AMESim API Rev 11

User's guide



LMS Imagine.Lab AMESim

How to contact LMS Imagine.Lab



www.lmsintl.com

Web site

www.lmsintl.com/support

Technical support



See here for email addresses for your local office:

Sales, pricing and general information

www.lmsintl.com/lmsworldwide



+33 4 77 23 60 30

Phone



+33 4 77 23 60 31

Fax



LMS Imagine S.A.
7 place des Minimes
42300 Roanne - France

Postal address

AMESim® User's Guides

© Copyright LMS Imagine S.A. 1995-2012

The software described in this documentation is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from LMS Imagine S.A.

Trademarks

AMESim® is a registered trademark of LMS Imagine S.A.

AMESet® is a registered trademark of LMS Imagine S.A.

AMERun® is a registered trademark of LMS Imagine S.A.

AMECustom® is a registered trademark of LMS Imagine S.A.

LMS Imagine.Lab® is a registered trademark of LMS International N.V.

LMS Virtual.Lab Motion® is a registered trademark of LMS International N.V.

SysDM® is a registered trademark of LMS International N.V.

System Synthesis® is a registered trademark of LMS International N.V.

Other product or brand names are trademarks or registered trademarks of their respective holders.

TABLE OF CONTENTS

1.The Circuit API	1
1.1. Limitations	1
1.2. Getting started	1
1.3. Using API functions	3
1.3.1. Listing the functions available in the module	3
1.3.2. Getting function details	4
1.3.3. Creating a script file	4
2.Circuit API Tutorial	7
2.1. Building an AMESim circuit from scratch	7
2.1.1. Creating the system	7
2.1.2. Adding components to the system	9
2.1.3. Setting submodels	13
2.1.4. Connecting components	14
2.1.5. Setting parameter values	16
2.1.6. Creating global parameters	24
2.1.7. Compiling the code	25
2.1.8. Setting run parameters	26
2.1.9. Running a simulation	26
2.1.10. Getting variable values	27
2.2. Managing your scripts	29
2.2.1. Log file and error management	30
2.2.2. One-click complex script creation	34

1. The Circuit API

This manual illustrates the use of the LMS.Imagine.Lab circuit API. It will guide you through the steps required to build your own **AMESim**-based application.

The tutorial uses the circuit API in Python on Windows, but it is possible to get the same result using the C or VBA version of the circuit API on any other supported platform.

Please consult the corresponding reference manuals for details of the differences between the C, Python and VBA circuit APIs.

Limitations
Getting started
Using API functions

1.1.Limitations

Please be aware of the following limitations when using the LMS.Imagine.Lab circuit API.

The circuit API has some limitations in comparison with a model built in **AMESim**:

- You cannot create text or other graphical objects on the sketch.
- Dynamic icons are not supported.
- Interface blocks are not supported
- Parameters which force a recompilation are not supported.
- You cannot modify a supercomponent.
- You cannot define batch parameters.
- You cannot perform linear analysis.
- You cannot use the Export setup or Design Exploration tools.

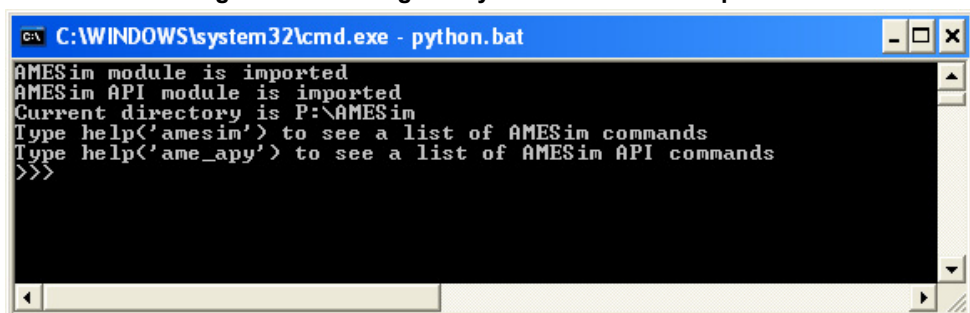
1.2.Getting started

Step 1: Start the Python command interpreter:

1. Start the Python command interpreter. In **AMESim**, use

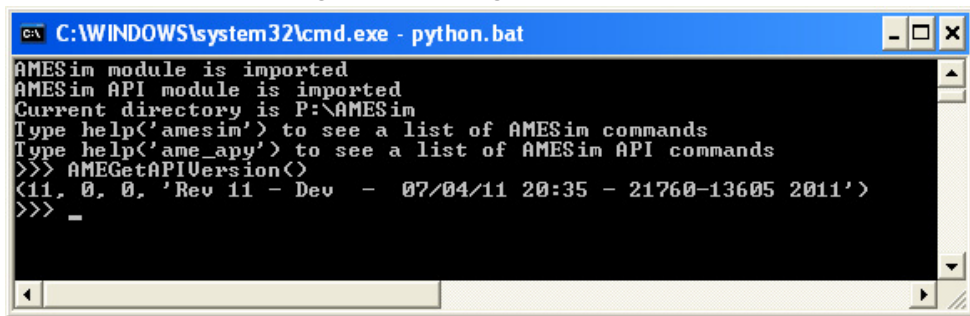
Tools > Python command interpreter, or click .

Figure 1.1: Starting the Python command interpreter



2. Check the circuit API version. Type **AMEGetAPIVersion()**

Figure 1.2: Getting the API Version



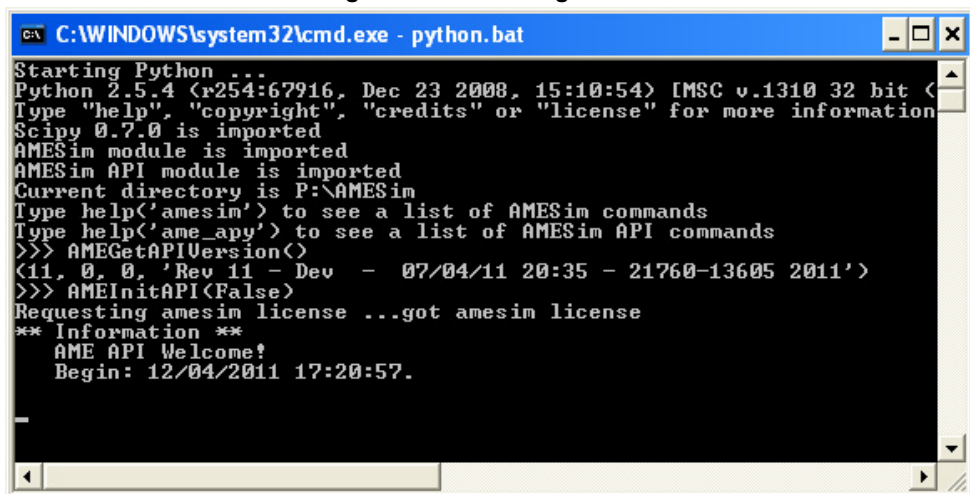
```
C:\WINDOWS\system32\cmd.exe - python.bat
AMESim module is imported
AMESim API module is imported
Current directory is P:\AMESim
Type help('amesim') to see a list of AMESim commands
Type help('ame_apy') to see a list of AMESim API commands
>>> AMEGetAPIVersion()
<11, 0, 0, 'Rev 11 - Dev - 07/04/11 20:35 - 21760-13605 2011'>
>>> _
```

Step 2: Initialize the API

Now we need to initialize the circuit API, by calling the **AMEInitAPI** command. It is necessary to initialize the API before using API functions (except for functions that simply return information, such as **AMEGetAPIVersion**). Otherwise, you will get error messages.

Type **AMEInitAPI(False)**

Figure 1.3: Initializing the API



```
C:\WINDOWS\system32\cmd.exe - python.bat
Starting Python ...
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (
Type "help", "copyright", "credits" or "license" for more information
Scipy 0.7.0 is imported
AMESim module is imported
AMESim API module is imported
Current directory is P:\AMESim
Type help('amesim') to see a list of AMESim commands
Type help('ame_apy') to see a list of AMESim API commands
>>> AMEGetAPIVersion()
<11, 0, 0, 'Rev 11 - Dev - 07/04/11 20:35 - 21760-13605 2011'>
>>> AMEInitAPI(False)
Requesting amesim license ...got amesim license
** Information **
AME API Welcome!
Begin: 12/04/2011 17:20:57.
_
```

Note that we use the optional boolean argument **False**, which indicates that we do not want to index the submodels in the path list. Please consult the *Circuit API* Reference Manual for details about this argument.



Calling **AMEInitAPI** checks out one **AMESim** license token. The license is released after you call **AMECloseAPI**.

1.3.Using API functions

This section describes the basic functions that will be useful for working with the API.



A full list of API functions is given in the HTML documentation.

“Listing the functions available in the module”, page 3

“Getting function details”, page 4

“Creating a script file”, page 4

1.3.1. Listing the functions available in the module

You can list the functions available in the `ame_apy` module using the Python command `help`:

`help('ame_apy')`

Figure 1.4: Calling the help

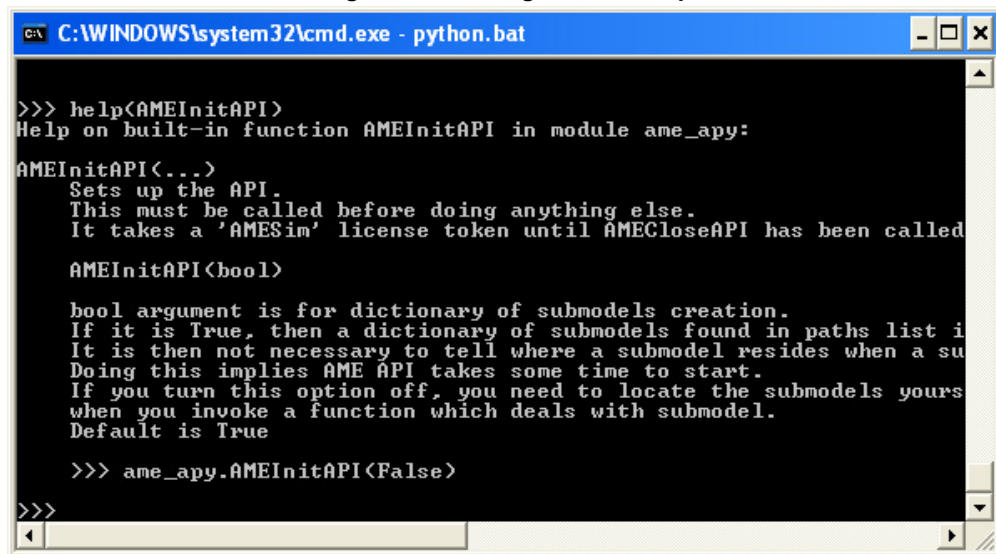
1.3.2. Getting function details

To get help for a specific function in the `ame_apy` module, type:

`help(FUNCTION_NAME)`. For example, for information on **AMEInitAPI**, you would type:

`help(AMEInitAPI)`

Figure 1.5: Getting function help



```
C:\WINDOWS\system32\cmd.exe - python.bat

>>> help(AMEInitAPI)
Help on built-in function AMEInitAPI in module ame_apy:

AMEInitAPI(...)
    Sets up the API.
    This must be called before doing anything else.
    It takes a 'AMESim' license token until AMECloseAPI has been called

    AMEInitAPI(bool)

    bool argument is for dictionary of submodels creation.
    If it is True, then a dictionary of submodels found in paths list i
    It is then not necessary to tell where a submodel resides when a su
    Doing this implies AME API takes some time to start.
    If you turn this option off, you need to locate the submodels yours
    when you invoke a function which deals with submodel.
    Default is True

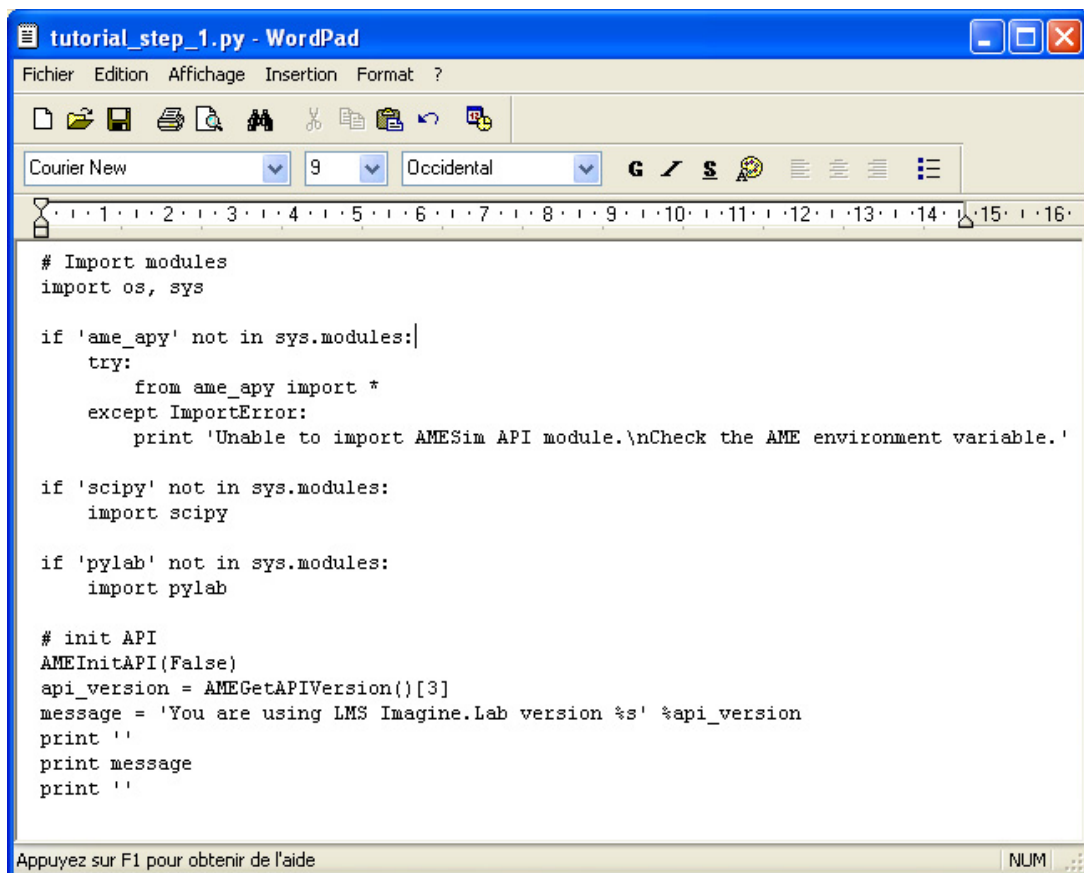
>>> ame_apy.AMEInitAPI(False)

>>>
```

1.3.3. Creating a script file

As well as typing commands in the Python Command Interpreter, you can write scripts. Here we show you how to write a Python script using the **AMESim** circuit API.

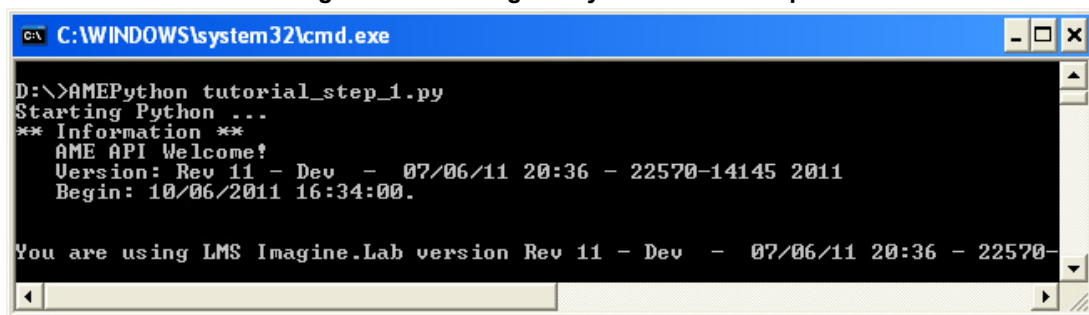
1. Create a file named *tutorial_step_1.py*.
2. Edit the file as follows:.



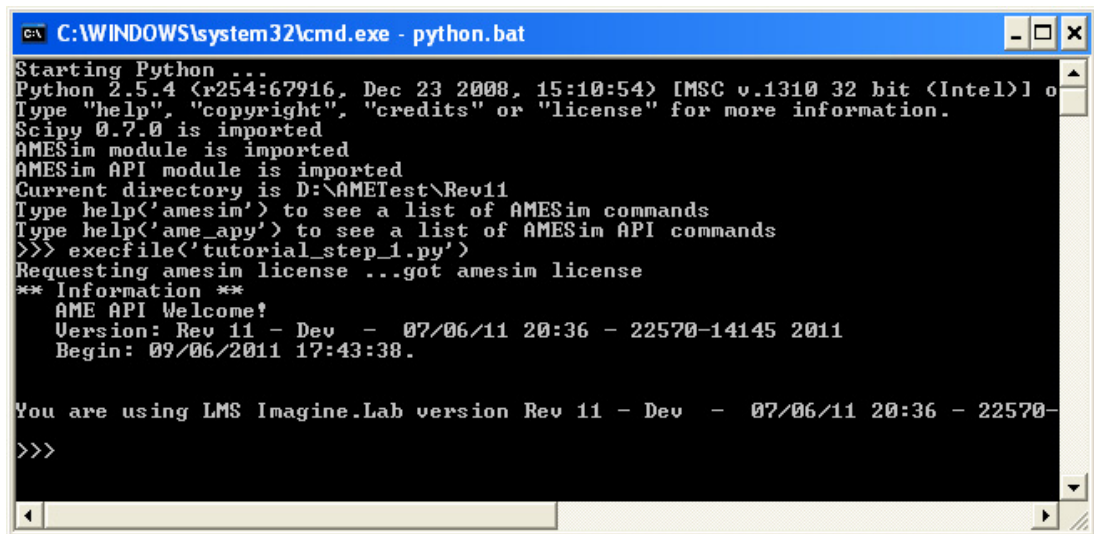
3. Save the file.

You can start your file from a command line prompt, by typing **AMEPython tutorial_step_1.py**:

Figure 1.6: Running the Python tutorial script



If you start the file from within Python, type `execfile('tutorial_step_1.py')`



```
C:\WINDOWS\system32\cmd.exe - python.bat
Starting Python ...
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] o
Type "help", "copyright", "credits" or "license" for more information.
Scipy 0.7.0 is imported
AMESim module is imported
AMESim API module is imported
Current directory is D:\AMETest\Rev11
Type help('amesim') to see a list of AMESim commands
Type help('ame_apy') to see a list of AMESim API commands
>>> execfile('tutorial_step_1.py')
Requesting amesim license ...got amesim license
*** Information ***
AME API Welcome!
Version: Rev 11 - Dev - 07/06/11 20:36 - 22570-14145 2011
Begin: 07/06/2011 17:43:38.

You are using LMS Imagine.Lab version Rev 11 - Dev - 07/06/11 20:36 - 22570-
>>>
```

We are now ready to create an **AMESim** system using the circuit API. This process is detailed in the next chapter.

2. Circuit API Tutorial

2.1. Building an AMESim circuit from scratch

“Creating the system”, page 7

“Adding components to the system”, page 9

“Setting submodels”, page 13

“Connecting components”, page 14

“Setting parameter values”, page 16

“Creating global parameters”, page 24

“Compiling the code”, page 25

“Setting run parameters”, page 26

“Running a simulation”, page 26

“Getting variable values”, page 27

We will start by creating a new file named *tutorial_step_2.py*, containing the following lines:

```
AMEInitAPI(False)
AMECreateCircuit('FlatTwin')
AMECloseCircuit(True)
AMECloseAPI()
```



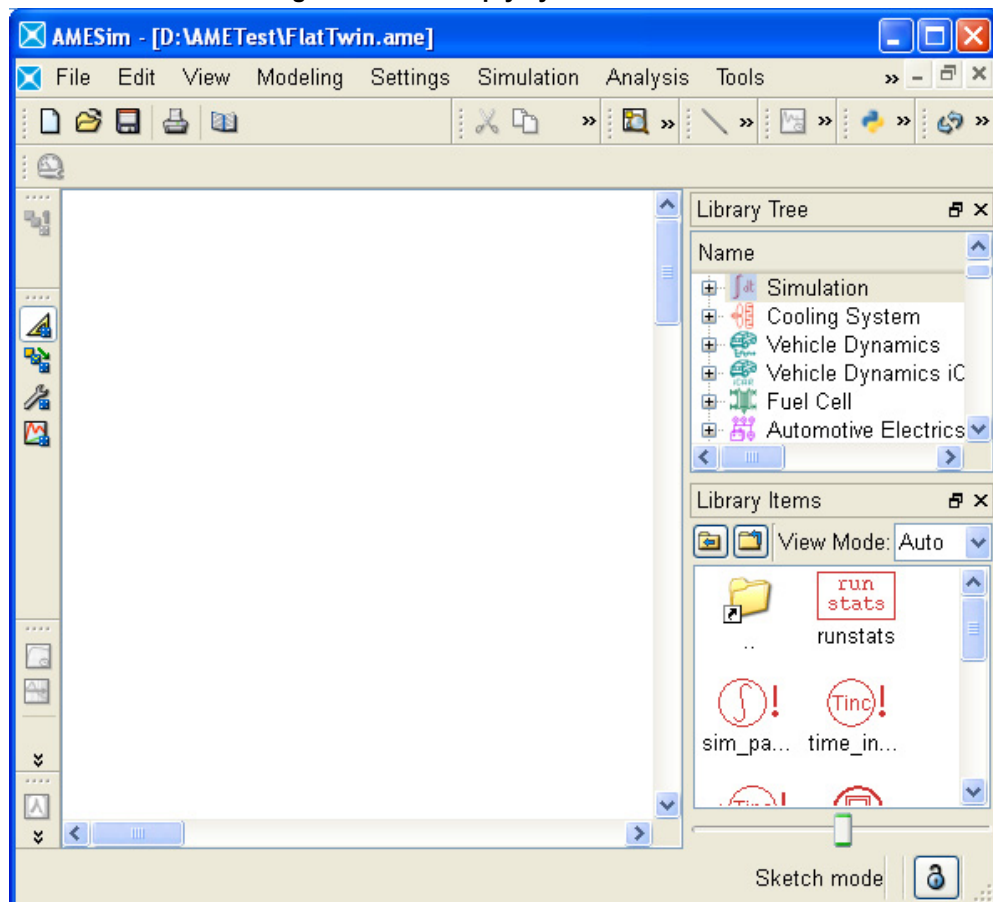
Adding (**True**) to the **AMECloseCircuit** line enables the circuit to be saved before it is closed. It would also be possible to insert **AMESaveCircuit** before **AMECloseCircuit** to achieve the same result in two steps.

2.1.1. Creating the system

The file above will create a new system by calling the **AMECreateCircuit** function. It then saves and closes it by calling the **AMECloseCircuit** function. You can check the details of these two functions using the help command in a Python Command Interpreter.

Execute the script. It creates an empty **AMESim** file named *FlatTwin.ame*. You can open it in **AMESim**:

Figure 2.1: The empty system in AMESim



You can change the working directory for your model using in the Python Command Interpreter as follows:

```
>>> import os  
  
>>> os.chdir('D:/AMETest')
```

The above example('D' would change the working directory to 'AMETest' on the 'D' drive.

You can check your current working directory as follows:

```
>>> os.getcwd()
```

2.1.2. Adding components to the system

We will now add components to the sketch using the **AMEAddComponent** function. We could add the components to the sketch without giving them any geometrical details, but since we will open the created sketch in **AMESim**, we do not want to create an ugly sketch so we will provide the component positions and rotation details.

- Components are rotated using **AMERotateComponent**
- Components are moved using **AMEMoveComponent**.

Insert the following lines after the command that creates the circuit and before the command that closes the circuit:

```

# Add components

# Add "mass_friction1port" component
AMEAddComponent('mass_friction1port', 'mass_friction1port', (40, 99))

# Add "springdamper01" component
AMEAddComponent('springdamper01', 'springdamper01')

# Rotate and flip the component
AMERotateComponent('springdamper01')
AMEMoveComponent('springdamper01', (89, 99))

# Add "simple_crank" component
AMEAddComponent('simple_crank', 'simple_crank')

# Rotate and flip the component
AMERotateComponent('simple_crank')
AMEFlipComponent('simple_crank')
AMEMoveComponent('simple_crank', (138, 103))

# Add "rconnector" component
AMEAddComponent('rconnector', 'rconnector')

# Rotate and flip the component
AMERotateComponent('rconnector', 2)
AMEMoveComponent('rconnector', (176, 149))

# Add "simple_crank_2" component
AMEAddComponent('simple_crank', 'simple_crank_2')

# Rotate and flip the component
AMERotateComponent('simple_crank_2')
AMEMoveComponent('simple_crank_2', (229, 103))

# Add "springdamper01_2" component
AMEAddComponent('springdamper01', 'springdamper01_2')

# Rotate and flip the component
AMERotateComponent('springdamper01_2')
AMEMoveComponent('springdamper01_2', (287, 99))

# Add "rload01" component
AMEAddComponent('rload01', 'rload01')

# Rotate and flip the component
AMERotateComponent('rload01')
AMEMoveComponent('rload01', (187, 174))

# Add "mass_friction1port_2" component
AMEAddComponent('mass_friction1port', 'mass_friction1port_2')

# Rotate and flip the component
AMERotateComponent('mass_friction1port_2', 2)
AMEMoveComponent('mass_friction1port_2', (336, 99))

```

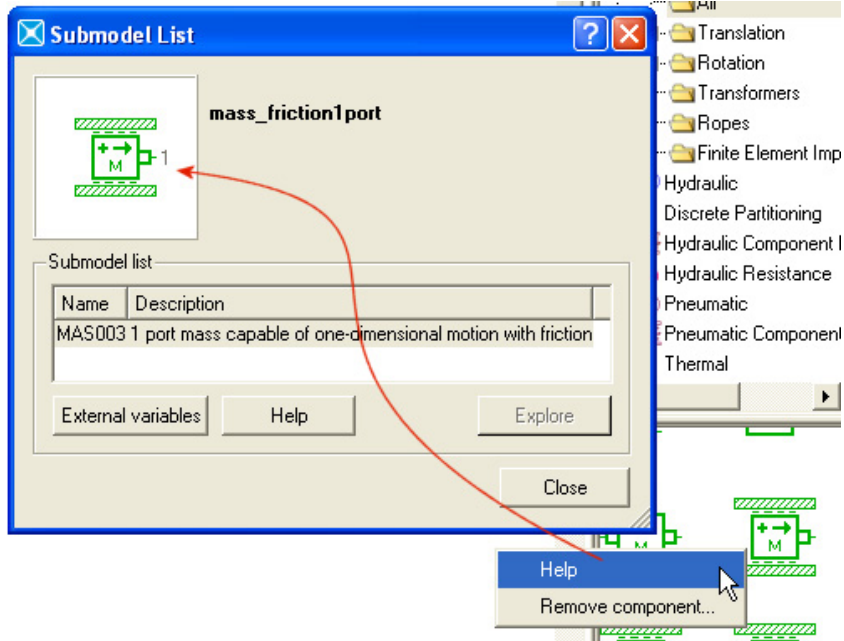
Here, the first argument of **AMEAddComponent** is the component icon name.

```
# Add "mass_friction1port" component
AMEAddComponent('mass_friction1port', 'mass_friction1port', (40, 99))
```

You can obtain component names from the [AMESim](#) category sidebar as follows:

1. Locate the component you require.
2. Right-click the component and select *Help*.

Figure 2.2: Obtaining a component name



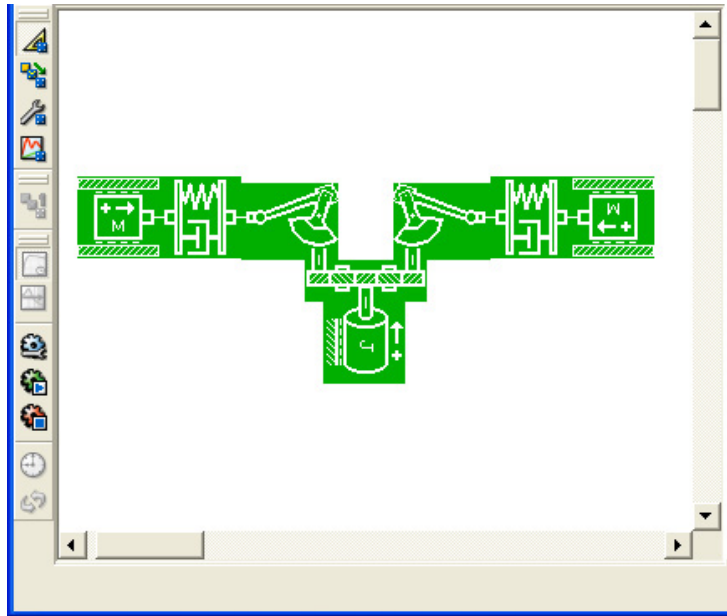
The icon is named *mass_friction1port*. The submodel it is compatible with is *MAS003*.

Note also that when adding a component to a circuit, we must give it a name (the second argument in the function) that is unique in the entire circuit. If you try to use the same name twice, you will get an error. This name identifies the component.

Launch your script. It produces the *FlatTwin.ame* system, now containing the components specified.

Open the system in [AMESim](#) again:

Figure 2.3: The updated system in AMESim



All the components are in reverse video as their submodels have not been set, and they therefore cannot be connected.

2.1.3. Setting submodels

You use the **AMEChangeSubmodel** command to set component submodels. Add the following lines to your script to set the submodels:

```
# Set submodels

# Set "MAS003" submodel to "mass_friction1port" component
AMEChangeSubmodel('mass_friction1port', 'MAS003', '$AME/libmec/
submodels')

# Set "SD0000A" submodel to "springdamper01" component
AMEChangeSubmodel('springdamper01', 'SD0000A', '$AME/libmec/
submodels')

# Set "CRANK0" submodel to "simple_crank" component
AMEChangeSubmodel('simple_crank', 'CRANK0', '$AME/libmec/
submodels')

# Set "RCON00" submodel to "rconnector" component
AMEChangeSubmodel('rconnector', 'RCON00', '$AME/libmec/
submodels')

# Set "CRANK0" submodel to "simple_crank_2" component
AMEChangeSubmodel('simple_crank_2', 'CRANK0', '$AME/libmec/
submodels')

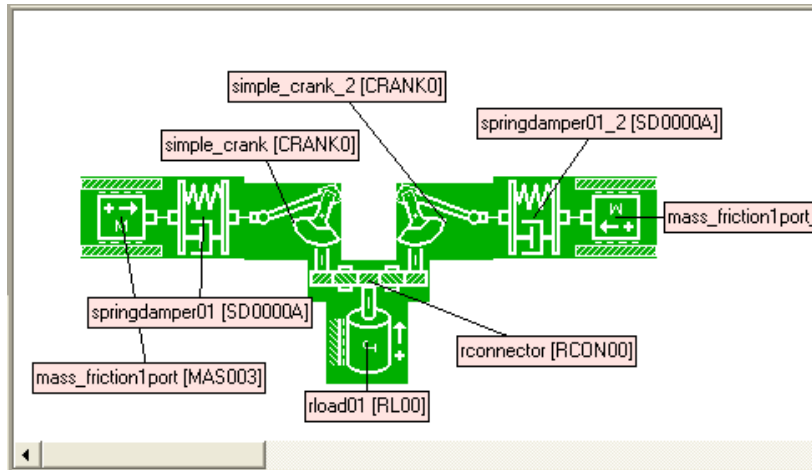
# Set "SD0000A" submodel to "springdamper01_2" component
AMEChangeSubmodel('springdamper01_2', 'SD0000A', '$AME/libmec/
submodels')

# Set "RL00" submodel to "rload01" component
AMEChangeSubmodel('rload01', 'RL00', '$AME/libmec/submodels')

# Set "MAS003" submodel to "mass_friction1port_2" component
AMEChangeSubmodel('mass_friction1port_2', 'MAS003', '$AME/libmec/
submodels')
```

Launch your script and then open the circuit produced in [AMESim](#). Right-click and select **Labels > Show component labels**:

Figure 2.4: The circuit with submodels set

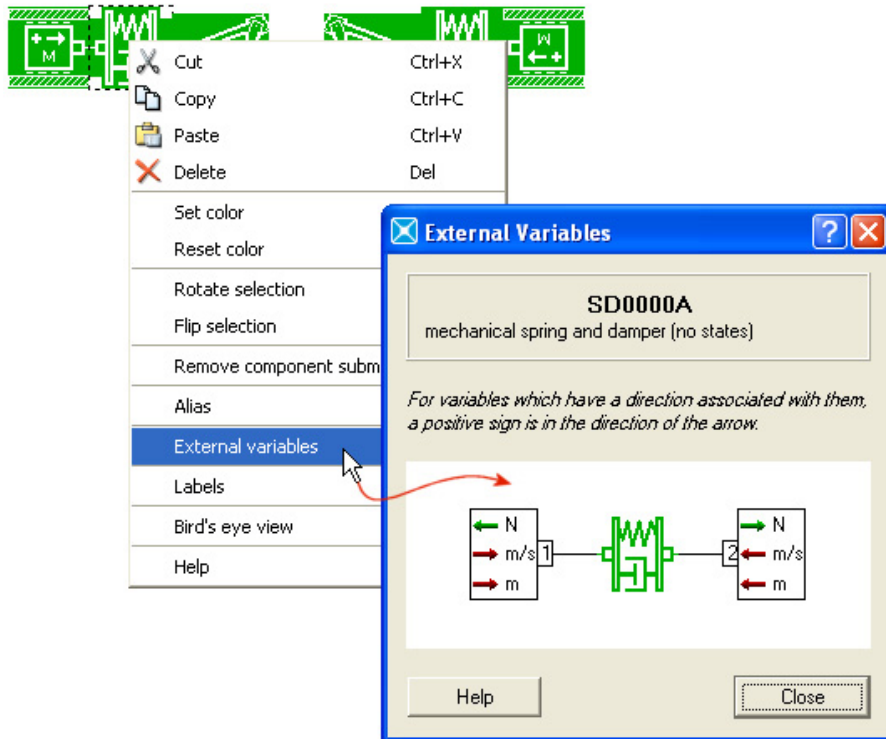


Again, all the components are in reverse video, this time because the components are not yet connected.

2.1.4. Connecting components

In order to connect two components, you require the port numbers. To obtain the port numbers, right-click a component and select *External variables*:

Figure 2.5: External Variables



Here we can see the component port numbers. However, when you connect the ports using the script, you require the port *indexes*. The conversion to obtain the indexes is *port index = port number - 1*.

Modify your script to perform the connection by adding the following lines:

```

# Do connections

# Connect "mass_friction1port" and "springdamper01"
AMEConnectTwoPorts('mass_friction1port', 0, 'springdamper01', 0)

# Connect "springdamper01" and "simple_crank"
AMEConnectTwoPorts('springdamper01', 1, 'simple_crank', 1)

# Connect "simple_crank" and "rconnector"
AMEConnectTwoPorts('simple_crank', 0, 'rconnector', 1)

# Connect "rconnector" and "simple_crank_2"
AMEConnectTwoPorts('rconnector', 0, 'simple_crank_2', 0)

# Connect "rconnector" and "rload01"
AMEConnectTwoPorts('rconnector', 2, 'rload01', 0)

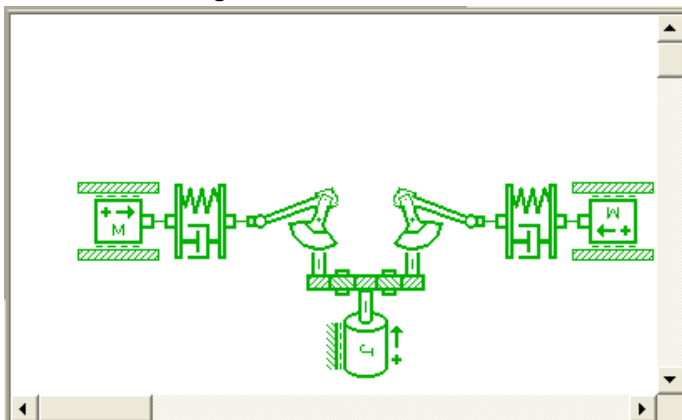
# Connect "simple_crank_2" and "springdamper01_2"
AMEConnectTwoPorts('simple_crank_2', 1, 'springdamper01_2', 0)

# Connect "springdamper01_2" and "mass_friction1port_2"
AMEConnectTwoPorts('springdamper01_2', 1, 'mass_friction1port_2', 0)

```

Launch your script again, and then open the system in [AMESim](#):

Figure 2.6: Ports connected



The components are no longer in reverse video, they are connected.

2.1.5. Setting parameter values

Next we will assign parameters values to component submodels using the **AMESetParameterValue** function.

We will set the following values:

Alias	Submodel	Title	Name	Value
mass_friction1port	MAS003-1	velocity at port 1[m/s]	v1	0
		displacement at port 1 [m]	x1	0
		mass [kg]	mass	MASS
		coefficient of viscous friction [N/(m/s)]	coefv	0
		coefficient of windage [N/(m/s)**2]	wind	0
		coulomb friction force [N]	coul	0
		stiction force [N]	stict	0
		inclination (+90 port 1 lowest, -90 port 1 highest) [degree]	angle	0

Alias	Submodel	Title	Name	Value
mass_friction1port	MAS003-2	velocity at port 1[m/s]	v1	0
		displacement at port 1 [m]	x1	0
		mass [kg]	mass	MASS
		coefficient of viscous friction [N/(m/s)]	coefv	0
		coefficient of windage [N/(m/s)**2]	wind	0
		coulomb friction force [N]	coul	0
		stiction force [N]	stict	0
		inclination (+90 port 1 lowest, -90 port 1 highest) [degree]	angle	0
rload01	RL00-1	shaft speed [rev/min]	omega	4000
		moment of inertia [kgm**2]	J	10
		coefficient of viscous friction [Nm/(rev/min)]	vis	1
		coulomb friction torque [Nm]	coul	0
		stiction torque [Nm]	stict	0

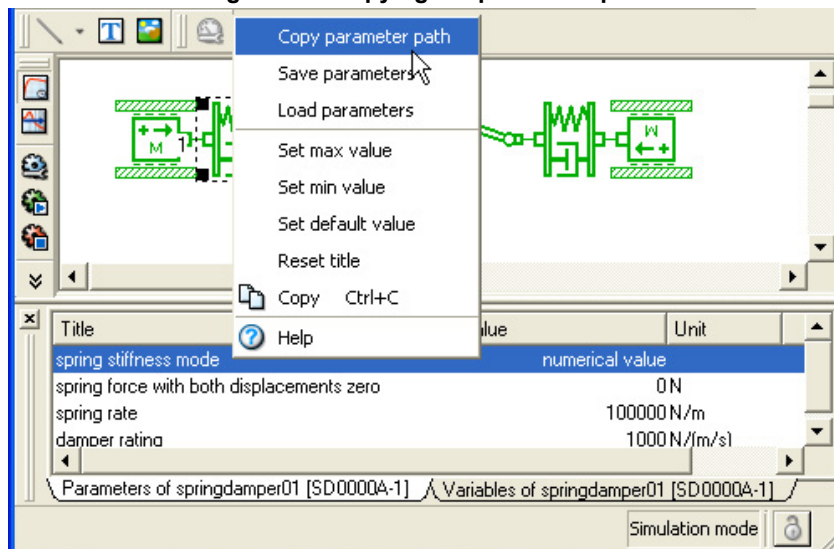
Alias	Submodel	Title	Name	Value
simple_crank	CRANK0-1	angular position of crank [degree]	theta	ANG
		radius of crank [mm]	rad	R
		length of connecting rod [mm]	length	L
		offset for displacement [mm]	offset	0
simple_crank_2	CRANK0-2	angular position of crank [degree]	theta	ANG+OFF
		radius of crank [mm]	rad	R
		length of connecting rod [mm]	length	L
		offset for displacement [mm]	offset	0
spring_damper01	SD0000A-1	spring force with both displacements zero [N]	force0	0
		spring rate [N/m]	srate	100000
		damper rating [N/(m/s)]	cdamp	1000
		material shear modulus [N/m**2]	G	8.57e+10
		spring diameter [mm]	sdiam	20
		wire diameter [mm]	wdiam	2
		spring stiffness mode	stiffmode	numerical value
		number of active coils	ncoils	10

Alias	Submodel	Title	Name	Value
spring_damper01_2	SD0000A-2	spring force with both displacements zero [N]	force0	0
		spring rate [N/m]	srate	100000
		damper rating [N/(m/s)]	cdamp	1000
		material shear modulus [N/m**2]	G	8.57e+10
		spring diameter [mm]	sdiam	20
		wire diameter [mm]	wdiam	2
		spring stiffness mode	stiffmode	numerical value
		number of active coils	ncoils	10

Obtaining parameter and variable names

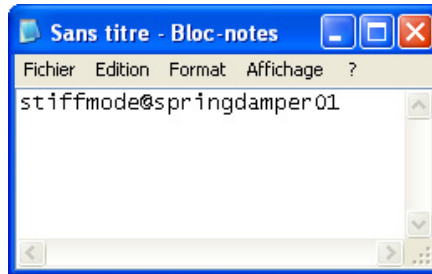
To obtain the name of a parameter or variable from an existing **AMESim** system, right-click and select *Copy parameter path* or *Copy variable path*:

Figure 2.7: Copying the parameter path



Once you have selected either of these functions, you can paste the path into any text editor to display the result:

Figure 2.8: The parameter path

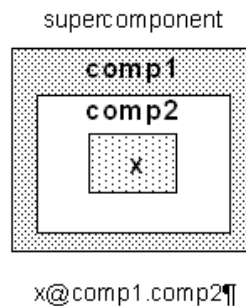


Here we can see the parameter path for the parameter named *stiffmode* on the *springdamper01*. In the code this will be used as follows when setting the parameters:

```
AMESetParameterValue('stiffmode@springdamper01', '1')
```

This information will be necessary when setting the parameters.

Note that in the case of a supercomponent, the path is as follows:



Editing the code

The code which sets these parameters is quite long, but simple:

Set parameter values

Set 'mass_friction1port' parameters

AMESetParameterValue('v1@mass_friction1port', '0')
AMESetParameterValue('x1@mass_friction1port', '0')
AMESetParameterValue('mass@mass_friction1port', 'MASS')
AMESetParameterValue('coefv@mass_friction1port', '0')
AMESetParameterValue('wind@mass_friction1port', '0')
AMESetParameterValue('coul@mass_friction1port', '0')
AMESetParameterValue('stict@mass_friction1port', '0')
AMESetParameterValue('angle@mass_friction1port', '0')

Set 'springdamper01' parameters

AMESetParameterValue('force0@springdamper01', '0')
AMESetParameterValue('srate@springdamper01', '1e5')
AMESetParameterValue('cdamp@springdamper01', '1e3')
AMESetParameterValue('G@springdamper01', '8.57e10')
AMESetParameterValue('sdiam@springdamper01', '20')
AMESetParameterValue('wdiam@springdamper01', '2')
AMESetParameterValue('stiffmode@springdamper01', '1')
AMESetParameterValue('ncoils@springdamper01', '10')

Set 'simple_crank' parameters

AMESetParameterValue('theta@simple_crank', 'ANG')
AMESetParameterValue('rad@simple_crank', 'R')
AMESetParameterValue('length@simple_crank', 'L')
AMESetParameterValue('offset@simple_crank', '0')

Set 'rconnector' parameters

No parameter

Set 'simple_crank_2' parameters

AMESetParameterValue('theta@simple_crank_2', 'ANG+OFF')
AMESetParameterValue('rad@simple_crank_2', 'R')
AMESetParameterValue('length@simple_crank_2', 'L')
AMESetParameterValue('offset@simple_crank_2', '0')

Set 'springdamper01_2' parameters

AMESetParameterValue('force0@springdamper01_2', '0')

```

AMESetParameterValue('srate@springdamper01_2', '1e5')
AMESetParameterValue('cdamp@springdamper01_2', '1e3')
AMESetParameterValue('G@springdamper01_2', '8.57e10')
AMESetParameterValue('sdiam@springdamper01_2', '20')
AMESetParameterValue('wdiam@springdamper01_2', '2')
AMESetParameterValue('stiffmode@springdamper01_2', '1')
AMESetParameterValue('ncoils@springdamper01_2', '10')

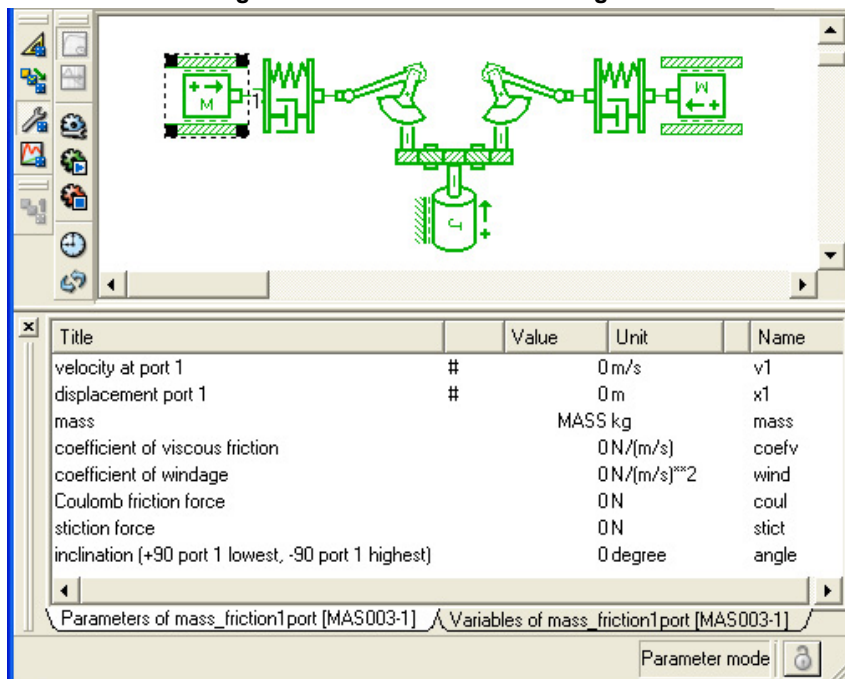
# Set 'load01' parameters
AMESetParameterValue('omega@rload01', '4000')
AMESetParameterValue('J@rload01', '10')
AMESetParameterValue('vis@rload01', '1')
AMESetParameterValue('coul@rload01', '0')
AMESetParameterValue('stict@rload01', '0')

# Set 'mass_friction1port_2' parameters
AMESetParameterValue('v1@mass_friction1port_2', '0')
AMESetParameterValue('x1@mass_friction1port_2', '0')
AMESetParameterValue('mass@mass_friction1port_2', 'MASS')
AMESetParameterValue('coefv@mass_friction1port_2', '0')
AMESetParameterValue('wind@mass_friction1port_2', '0')
AMESetParameterValue('coul@mass_friction1port_2', '0')
AMESetParameterValue('stict@mass_friction1port_2', '0')
AMESetParameterValue('angle@mass_friction1port_2', '0')

```

Copy and paste this code into your **tutorial_step_2.py** file and run the script. If you open your model in **AMESim** you will see that the parameter values have been assigned to the submodels:

Figure 2.9: Parameter values assigned



You will have noticed that some of the parameter values are not numerical values, but text: **MASS**, **ANG**, **OFF**, **R**, **L**. These are *Global parameters*. Global parameters are commonly used

in [AMESim](#) when several parameters need the same value.

The next step is to create these Global parameters.

2.1.6. Creating global parameters

Global parameters are created using the function **AMEAddGlobalParameter**. You should consult the documentation for this function by typing **help ('AMEAddGlobalParameter')**.

We will create the following Global parameters:

NAME	TITLE	VALUE	UNIT
MASS	mass of piston	0.5	kg
BORE	bore of piston	60	mm
R	radius of crank	30	mm
ANG	initial angular position of crank	30	degree
OFF	difference in piston angles	180	degree
L	length of connecting rod	40	mm

We suggest you set meaningful default, minimum and maximum values for your Global parameters, according to usage type.

Add the code to your script:

Set Global parameter values

```
AMEAddGlobalParameter('MASS', 'mass of piston', 'ame_real_parameter',  
'0.5', '0.1', '0.5', '2', 'kg')  
AMEAddGlobalParameter('BORE', 'bore of piston', 'ame_real_parameter', '60',  
'20', '60', '100', 'mm')  
AMEAddGlobalParameter('R', 'radius of crank', 'ame_real_parameter', '30',  
'10', '30', '50', 'mm')  
AMEAddGlobalParameter('ANG', 'initial angular position of piston 1',  
'ame_real_parameter', '0', '0', '0', '360', 'degree')  
AMEAddGlobalParameter('OFF', 'difference in piston angles',  
'ame_real_parameter', '180', '0', '0', '360', 'degree')  
AMEAddGlobalParameter('L', 'length of connecting rod',  
'ame_real_parameter', '40', '25', '40', '110', 'mm')
```

Launch your script and then open the circuit in [AMESim](#). Switch to Parameter mode and check the parameter values and Global parameter values you created:

Figure 2.10: Global Parameters set for a component

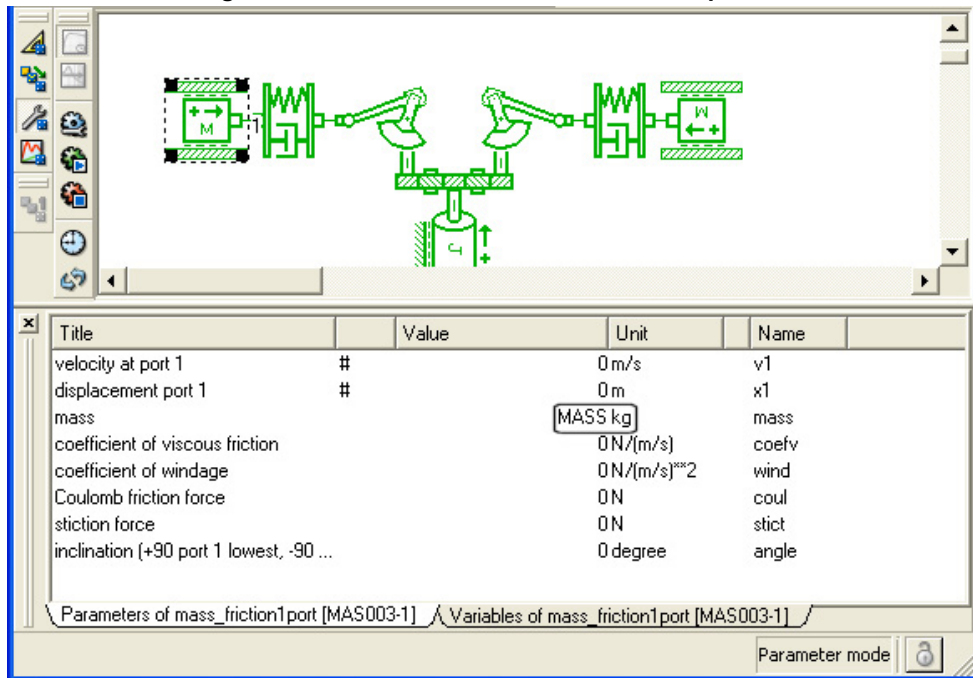
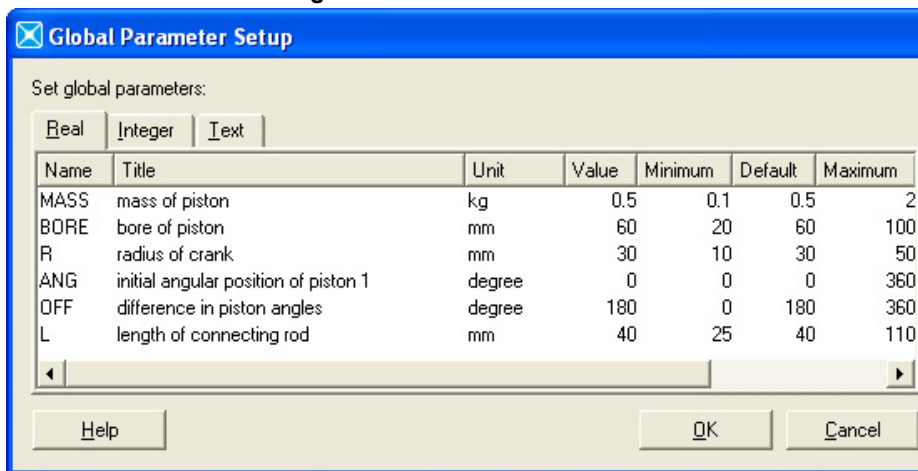


Figure 2.11: Global Parameters



Note that when you opened the circuit in **AMESim** and switched to *Parameter* mode, the system was compiled. To avoid this, you can configure your script to perform the compilation.

2.1.7. Compiling the code

You can compile the code for your circuit with the **AMEGenerateCode** function.



Compilation will fail if there are any unconnected ports in your circuit, or if a component or line does not have an assigned submodel.

The following compilation functions are also available:

- **AMEGetCompilers**: list available compilers.
 - **AMESetCompiler**: set your favorite compiler.
 - **AMESetDebugCompilation**: switch between debug or release compilation modes.
- Add the following line to your script in order to compile the circuit. You can add this line before the run parameter value settings:

```
# Generate code
AMEGenerateCode()
```

2.1.8. Setting run parameters

In this tutorial, we will just change the final simulation time (to 0.1s) and the print interval (to 1e-3s). We will leave other run parameters at their default values.

Add the following lines to your script to set the run parameters:

```
# Set run parameters
AMESetRunParameter('stop_time_s', '0.1')
AMESetRunParameter('interval_s', '1e-3')
```

2.1.9. Running a simulation

To start a simulation from your script, you use the function **AMERunSimulation**.

```
# Start simulation
print "
print 'Running "FlatTwin" system simulation...'

AMERunSimulation()
```

Add this function to your script and then run it. The script produces the *FlatTwin.ame* system with a simulation run. You can open it in [AMESim](#) to check the output.

2.1.10. Getting variable values

In the *FlatTwin* system, we are interested in visualizing piston displacement. To achieve this, we want to plot the `x1@mass_friction1port` and `x1@mass_friction1port_2` variables.



We will use the pylab module provided with [LMS Imagine.Lab AMESim](#) to plot these variable values.

Look at the code below.



You must import the scipy and pylab modules using the import command:

Import scipy and pylab modules

import scipy

import pylab

```

# Get & plot values
print 'Plotting "x1@mass_friction1port" and "x1@mass_friction1port_2"...'

# get variable values
res_1 = AMEGetVariableValues('x1@mass_friction1port')
res_2 = AMEGetVariableValues('x1@mass_friction1port_2')

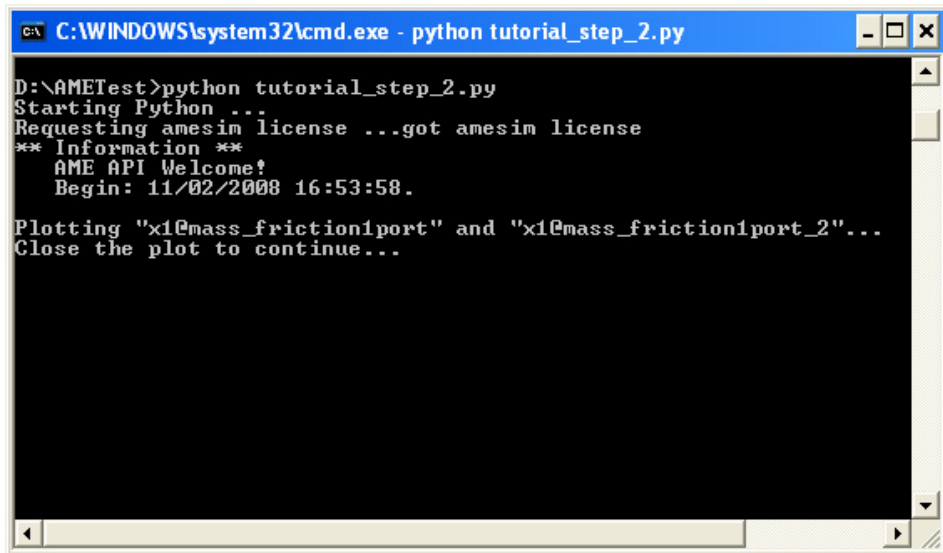
# Format values
x_1 = scipy.array(res_1)
x_2 =scipy.array(res_2)
# get variable title (these two variables have the same title)
variable_title = AMEGetVariableInfos('x1@mass_friction1port')[2]

# Plot variables
pylab.plot(x_1[:,0], x_1[:,1], label='x1@mass_friction1')
pylab.plot(x_2[:,0], x_2[:,1], label='x1@mass_friction1_2')
pylab.legend(loc='upper left')
pylab.xlabel('Time')
pylab.ylabel(variable_title)
pylab.grid(True)
print 'Close the plot to continue...'
pylab.show()

```

Add the code to your script and launch it. The selected variables are plotted:

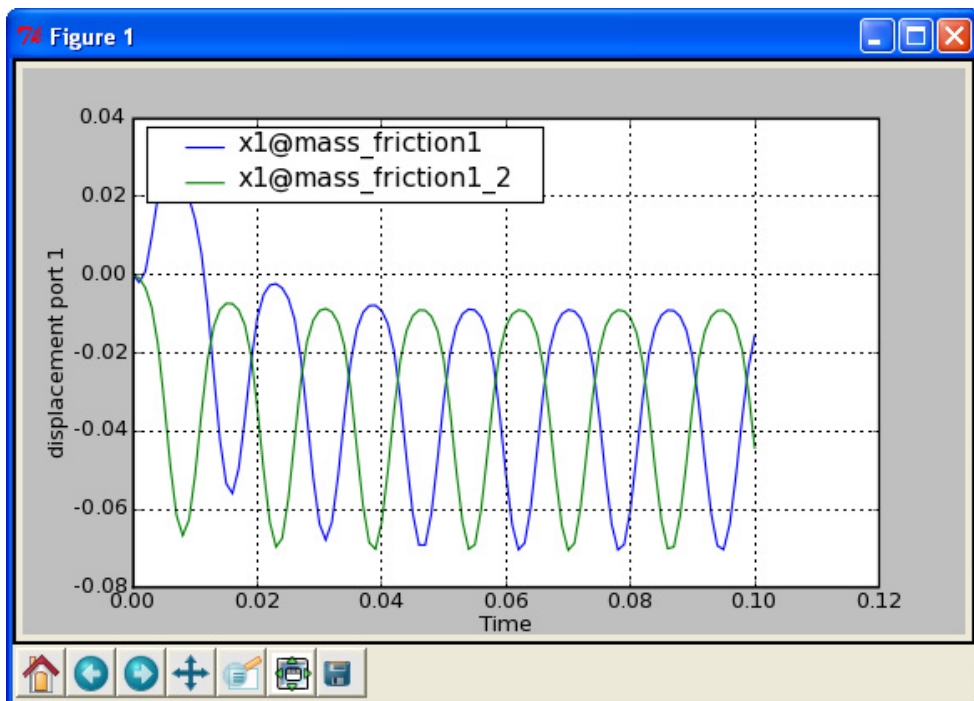
Figure 2.12: The plot



```
C:\WINDOWS\system32\cmd.exe - python tutorial_step_2.py

D:\AMETest>python tutorial_step_2.py
Starting Python ...
Requesting amesim license ...got amesim license
** Information **
AME API Welcome!
Begin: 11/02/2008 16:53:58.

Plotting "x1@mass_friction1port" and "x1@mass_friction1port_2"...
Close the plot to continue...
```



When you close the plot, the script terminates, and your *FlatTwin.ame* system is complete.

2.2.Managing your scripts

“Log file and error management”, page 30

“One-click complex script creation”, page 34

This section presents ways to manage your scripts.

2.2.1. Log file and error management

Sometimes you will want to know what's happening when you run your script. This is useful for debugging purposes. The **AMESim** API can help you trace what is happening, or can produce a log file.

In this section we will load an out of date system, upgrade it, and save it under a new name. At the same time, we will see how to log what your script is doing, and how to deal with errors.

Logging what your script is doing

To turn this function on, use the **AMESSubscribeToLogFile** function. To turn this function off, use **AMEUnSubscribeToLogFile**.

1. Copy *CentrifugalPump_out_of_date.ame* and *tutorial_step_3.py* to your working directory from the API folder of the **AMESim** tutorial directory.
2. Edit *tutorial_step_3.py* to enable the log file function.

```
# Add log file subscription
AMESSubscribeToLogFile('my_first_log.log')
```

3. Launch your script. It will produce a log file (in this case, named *my_first_log.log*) with the following information:

```
** Information **
AME API Welcome!
Begin: 08/02/2008 16:55:40.

** Warning **
"CentrifugalPump_out_of_date" circuit submodels may need updating.
Check submodels may be necessary.

** Information **
End: 08/02/2008 16:55:45
AME API Bye!.
```

There are four message levels:

- Information
- Warning
- Error
- Fatal

By default, log files receive messages from the *Warning* to *Fatal* levels (except for initialization

and closure messages).

You can change the filter level to catch more or fewer messages from the API.

Filtering messages from the API

We will now change the message filter information level to catch what the *Check Submodels* tool does when it upgrades a system.

Edit the *tutorial_step_3.py* to add the following lines:

```
# Check circuit submodels if necessary
if AMEDoesNeedCheckCircuitSubmodels():
    print 'Checking "CentrifugalPump_out_of_date" circuit submodels...'
    try:
        # Redirect the check submodels report into 'testFriction.log' file
        AMESetMessageFilter('ame_info')
        AMECheckCircuitSubmodels()
    except:
        print 'Unable to check "CentrifugalPump_out_of_date".'
```

Launch your script. The log file now contains details about the circuit upgrade:

**** Information ****

AME API Welcome!

Begin: 04/02/2008 08:57:21.

**** Warning ****

**"CentrifugalPump_out_of_date" circuit submodels may need update.
Check submodels may be necessary.**

**** Information ****

"CentrifugalPump_out_of_date" circuit check submodels report:

PQSG1

*** Checked OK.**

UD00

*** 2 differences found:**

**- number of integer stores
circuit: 1**

submodel file: 2

**- number of integer parameters
circuit: 0**

submodel file: 2

UD00

*** Updated OK.**

*** Checked OK.**

FXYA2

*** Checked OK.**

SPLT0

*** Checked OK.**

SGN10

*** Checked OK.**

TK000

*** Checked OK.**

PQSG2

*** Checked OK.**

VOR00

*** 5 differences found:**

**- number of internal variables
circuit: 1**

submodel file: 5

**- internal variable 1 type
circuit: activity variable**

submodel file: basic variable

**- internal variable 1 title
circuit: activity of hydraulic dissipation (orif)**

submodel file: flow coefficient (Cq)
- internal variable 1 unit
circuit: J
submodel file: null
- real parameter 7 title
circuit: critical flow number laminar->turbulent
submodel file: critical flow number (laminar -> turbulent)

VOR00

* Updated OK.

* Checked OK.

FX00

* Checked OK.

SIN0

* Checked OK.

TK000

* Checked OK.

JUN3P

* Checked OK.

LAG1

* Checked OK.

FP01

* Checked OK.

**** Information ****

There is a doubt about the value of 3 parameters:

- nstages@signal03
- iscyclic@signal03
- lamc@variableorifice.

**** Information ****

A backup of the original circuit has been created: 'E:/CentrifugalPump_out_of_date.ame.bak420_1'.

**** Information ****

"CentrifugalPump_out_of_date" circuit submodels have been updated.

**** Information ****

End: 04/02/2008 08:57:25

AME API Bye!.

2.2.2. One-click complex script creation

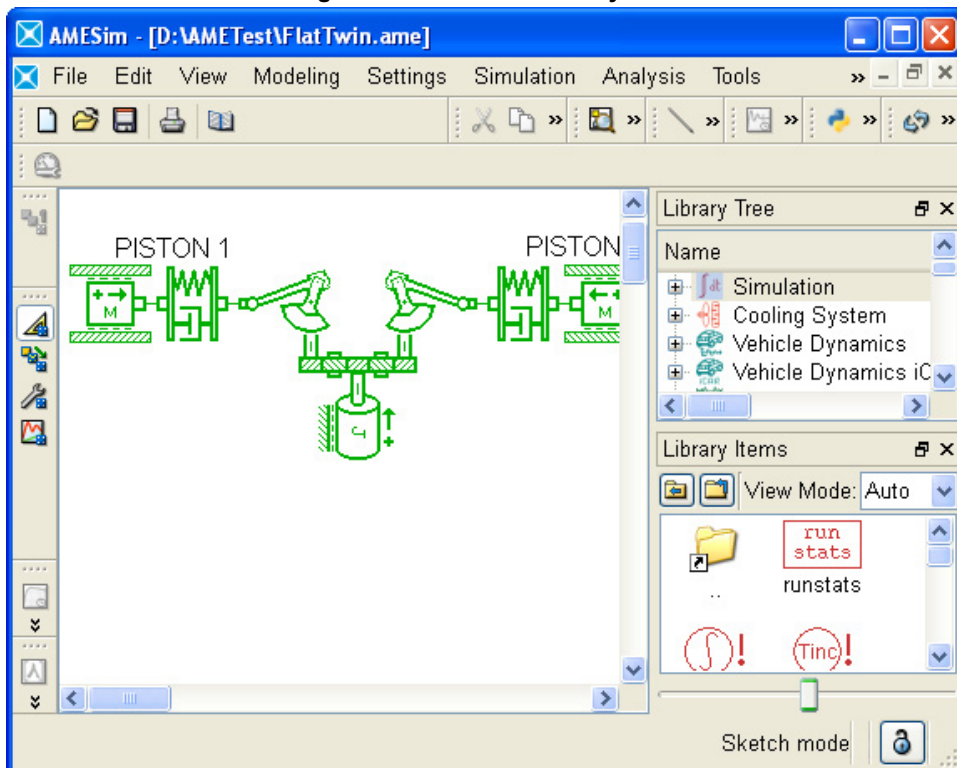
As we have seen, creating a complex system from scratch can be a long and arduous task.

In many cases, it is easier to use the [AMESim](#) graphical user interface to build and test your system, and then create the script to produce the same system and finally customize it.

You can indeed do this!

1. Copy the `$AME/demo/Tutorials/AMESimTutorials/FlatTwin.ame` system to your local working directory and open it with [AMESim](#).

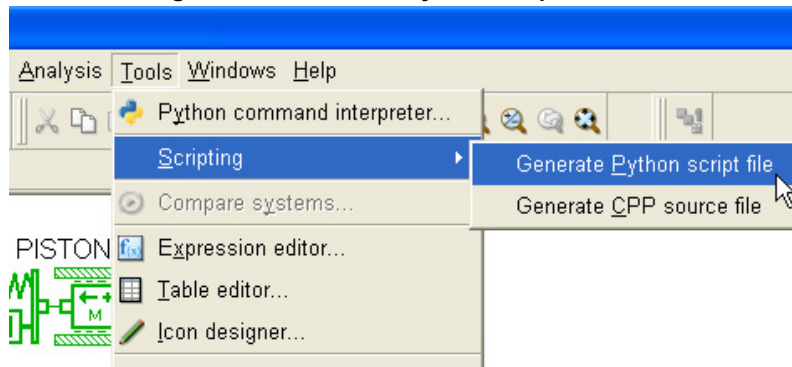
Figure 2.13: FlatTwin.ame system



2. Generate a ready-to-be-launched `tutorial_step_4.py` script file by using **Tools > Scripting > Generate Python script file**.

You can also produce a ready-to-be-compiled Cpp file using **Tools > Scripting > Generate Cpp source file**.

Figure 2.14: One-click Python Script creation



Doing this on a very complex system will save you a lot of time. You can then edit the script produced and add whatever commands you require. Now you can generate a model with the script and check the result by opening it in **AMESim** as we did at the beginning of the chapter.



If you pass a system created with **AMESim** (v4.3 or earlier) through the script generation tool, it will create a script which works correctly, but with missing component rotation instructions. Consequently, when you run your script, the system produce will be very ugly if you open it in **AMESim**. You can modify it by hand to add the suitable missing **AMERotateComponent** commands to the script.

To get a script containing all geometrical instructions, your system must be created with **AMESim** Rev 7A or higher.

Upgrading an old system will not solve this problem.

Index

A

AMEAddComponent	9
AMEAddGlobalParameter	24
AMECloseAPI	2
AMECloseCircuit	7
AMECreateCircuit	7
AMEGenerateCode	25
AMEGetAPIVersion	1
AMEGetCompilers	26
AMEInitAPI	2
AMEMoveComponent	9
AMERotateComponent	9
AMERunSimulation	26
AMERunSimulation()	26
AMESetCompiler	26
AMESetDebugCompilation	26
AMESetParameterValue	16
AMESetRunParameter	26
AMESubscribeToLogFile	30
API, AMEAddComponent	9
API, AMEAddGlobalParameter	24
API, AMECloseCircuit	7
API, AMECreateCircuit	7
API, AMEGenerateCode	25
API, AMEGetCompilers	26
API, AMEMoveComponent	9
API, AMERotateComponent	9
API, AMERunSimulation	26
API, AMESetCompiler	26
API, AMESetDebugCompilation	26
API, AMESetParameterValue	16
API, AMESetRunParameter	26
API, AMESubscribeToLogFile	30
API, building an AMESim circuit from scratch	7
API, compiling the code	25
API, connecting components	14
API, creating a script file	4
API, creating global parameters	24
API, creating the system	7
API, filtering messages	31
API, getting function details	4
API, getting variable values	27
API, log file and error management	30
API, managing your scripts	29
API, obtaining parameter and variable names	20
API, one-click complex script creation	34
API, pylab	27
API, running a simulation	26
API, scipy	27
API, setting parameter values	16
API, setting run parameters	26

B	
Building	7
C	
Circuit API	1
F	
Filtering messages from the API	31
H	
help('ame_apy')	3
P	
pylab	27
Python Command Interpreter	1
S	
scipy	27