

---

# **pymls Documentation**

*Release 1.4.6*

**O. Dazel, M. Gaborit, P. Göransson**

**Jun 07, 2019**



---

## Contents:

---

<b>1</b>	<b>Materials</b>	<b>3</b>
<b>2</b>	<b>API documentation</b>	<b>5</b>
2.1	pymls.solver module . . . . .	5
2.2	pymls.backing module . . . . .	7
2.3	pymls.analysis module . . . . .	7
2.4	pymls.interface package . . . . .	7
2.5	pymls.layers package . . . . .	8
2.6	pymls.media package . . . . .	9
2.7	pymls.utils package . . . . .	13
2.8	Module contents . . . . .	14
<b>3</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Bibliography</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>
	<b>Index</b>	<b>21</b>



pymIs allows to solve acoustic propagation problems through structures with multiple layers using Dazel *et al.*'s method as described in “A stable to model the acoustic response of multilayered structures” (Journal of Applied Physics, 2013, doi: [10.1063/1.4790629](https://doi.org/10.1063/1.4790629) ).



# CHAPTER 1

---

## Materials

---

Materials are described using classes from `pymls.media` module. They can be either created from scratch by instantiating the corresponding class and inserting values into the parameters or reading [YAML](#) files.





## 2.1 pymls.solver module

**exception** `pymls.solver.IncompleteDefinitionError` (*msg='The definition is incomplete and no analysis can be performed'*)

Bases: `Exception`

Exception raised when attempting to solve an incomplete system

Either missing layers definition or no backing will produce such an error.

**class** `pymls.solver.Solver` (*media=None, analyses=None, layers=None, backing=None*)

Bases: `object`

Stores a system to solve and parameters for the analysis.

Performs analysis and gives back raw unmodified/cleaned results. All post-processing should be done *out* of this class.

### Parameters

- **layers** (*list of Layer/StochasticLayer instances*) – The right most layer appears last in the list.
- **backing** (function reference from *pymls.backing*) – Describe the type of backing condition
- **media** (*list of Media subclasses, optional*) – Stores all media used in the system for later reference
- **analyses** (*list of Analysis instances, optional*) – If only one instance is provided with a list, the constructor will wrap it into a list.

### media

*list of Media subclasses, optional*

### layers

*list of Layer/StochasticLayer instances*

**backing**

function reference from *pymls.backing*

**analyses**

*list of Analysis instances, optional*

**resultset**

*list of dict* – Contains the results for all analysis and metadata

**solve(frequencies, angles, n\_draws, prng\_state) : list of dict**

Starts the solving process w/w stochastic parameters.

**check\_is\_complete() : bool**

Check that all required data has been provided and gathers media.

**Methods**

<code>check_is_complete()</code>	Check that all required data has been provided and gathers media.
<code>compute_fields(layer_id, frequency, theta_inc)</code>	Returns the backpropagation matrix from the first interface to layer num.
<code>solve([frequencies, angles, n_draws, prng_state])</code>	Starts the solving process w/w stochastic parameters.

**check\_is\_complete()**

Check that all required data has been provided and gathers media.

**Returns** True if the described is complete and ready to be solved.

**Return type** bool

**Raises** *IncompleteDefinitionError* – If the system is incomplete (missing layer or backing)

**compute\_fields(layer\_id, frequency, theta\_inc)**

Returns the backpropagation matrix from the first interface to layer num. *layer\_id*.

**Parameters**

- **frequency** (*float*) – frequency at which backprop is computed
- **theta\_inc** – angle of incidence
- **layer\_id** (*int*) – id of the layer up to which backpropagate

**Returns** *layer\_func* – Function to get the propagation in the layer

**Return type** callable

**Raises** *ValueError* : – if the id is invalid

**solve(frequencies=None, angles=0, n\_draws=1000, prng\_state=None)**

Starts the solving process w/w stochastic parameters.

The function looks for *StochasticLayer* instances in the *layers* list and flag them. It creates an *Analysis* if all parameters are provided upon call and runs the corresponding solver functions for all registered analysis, gathering results in *resultset*.

**Parameters**

- **frequencies** (*list, optional*) – list of frequency where to compute the analysis. If it isn't provided and no *Analysis* has been registered before hand, the function will return nothing.
- **angles** (*optional*) – Defaults to 0. Can be a list or anything *Analysis* can parse to an iterable.
- **n\_draws** (*int*) – Number of draws for the stochastic analyses.
- **prng\_state** (*tuple*) – Saved state for Numpy's pseudo random number generator (see *numpy.random.get\_state*)
- **\_numpy.random.get\_state** (.) –

**Returns** **resultset** – Set of all computed results and relevant metadata provided as a dict for easy serialisation.

**Return type** dict or list of dict

## 2.2 pymls.backing module

`pymls.backing.rigid(omega, k_x)`

`pymls.backing.transmission(omega, k_x)`

## 2.3 pymls.analysis module

**class** `pymls.analysis.Analysis` (*name, freqs, angles, enable\_stochastic=False*)

Bases: `object`

## 2.4 pymls.interface package

### 2.4.1 Submodules

#### 2.4.2 pymls.interface.interfaces module

`pymls.interface.interfaces.elastic_fluid_interface(O)`

`pymls.interface.interfaces.elastic_pem_interface(O)`

`pymls.interface.interfaces.fluid_elastic_interface(O)`

`pymls.interface.interfaces.fluid_pem_interface(O)`

`pymls.interface.interfaces.pem_elastic_interface(O)`

`pymls.interface.interfaces.pem_fluid_interface(O)`

#### 2.4.3 pymls.interface.utils module

`pymls.interface.utils.generic_interface(medium_left, medium_right)`

Returns a callable to the interface function corresponding to the two given media.

Note: interface functions are not symmetrical ( `generic_interface(m1, m2) != generic_interface(m2, m1)` ).

`pymls.interface.utils.rigid_interface` (*medium*)

Returns a callable to the rigid backing function corresponding to the given media.

## 2.4.4 Module contents

## 2.5 pymls.layers package

### 2.5.1 Submodules

### 2.5.2 pymls.layers.elastic module

`pymls.layers.elastic.transfert_elastic` (*Omega\_minus, omega, k\_x, medium, d*)

### 2.5.3 pymls.layers.fluid module

`pymls.layers.fluid.transfert_fluid` (*Omega\_minus, omega, k\_x, medium, d*)

### 2.5.4 pymls.layers.layer module

**class** `pymls.layers.layer.Layer` (*medium, thickness, name='Unnamed Layer'*)

Bases: `object`

#### Methods

<b>register</b>	
<b>update_frequency</b>	

**register** (*hook\_name*)

**update\_frequency** (*omega*)

**class** `pymls.layers.layer.StochasticLayer` (*medium, thickness, stochastic\_param, pdf, name='Unnamed Layer'*)

Bases: `pymls.layers.layer.Layer`

#### Methods

<b>register</b>	
<b>reinit</b>	
<b>update_frequency</b>	

**reinit** ()

### 2.5.5 pymls.layers.pem module

`pymls.layers.pem.transfert_pem` (*Omega\_minus, omega, k\_x, medium, d*)

## 2.5.6 pymls.layers.screen module

`pymls.layers.screen.transfert_screen` (*Omega\_minus, omega, k\_x, m, d*)

## 2.5.7 pymls.layers.utils module

`pymls.layers.utils.generic_layer` (*medium*)

## 2.5.8 Module contents

# 2.6 pymls.media package

## 2.6.1 Module contents

`pymls.media.from_yaml` (*filename, force=None*)

Reads medium definition from YAML file *filename*. Raises an IOError if the file is not found, ValueError if medium type is not known and a LookupError if the parameter definition is incomplete.

One may set the optional argument 'force' to a medium type to force loading this one.

### pymls.media.medium

**class** `pymls.media.medium.Medium`

Bases: object

Base class for medium definition and manipulation

All derived classes have a few class parameters explained below.

**EXPECTED\_PARAMS, OPT\_PARAMS**

*list of tuple, class attribute* – List of all expected parameters (resp. optional parameters) for this medium to be well defined. The format is (*param\_name, param\_type*) where *param\_name* is a *str* and *param\_type* a *type*.

**MEDIUM\_TYPE**

*str, class attribute* – State the type of medium, used to determine the real physics hidden in the medium (to distinguish a fluid from and equivalent fluid for instance)

**MODEL**

*str, class attribute* – State which type of propagation model to use.

**name**

*str* – Medium's name (default to Generic Medium)

**omega**

*positive float or -1* – Last circular frequency for which the parameters have been updated

### Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	Computes parameters' value for the given circular frequency

`_compute_missing()`

Computes the required constant parameters missing from the definition

### Notes

Implemented in derived classes

`from_dict(parameters)`

Reads medium definition from a hashmap of params.

Load all possible & recognised parameters and run `_compute_missing` to complete the definition.

**Parameters** `parameters` (*dict*) – Linking parameter names and values.

**Raises** `LookupError` – If the parameter definition is incomplete (missing parameters listed in `EXPECTED_PARAMS`).

`update_frequency(omega)`

Computes parameters' value for the given circular frequency

### Notes

Implemented in derived classes

## pymls.media.fluid

**class** `pymls.media.fluid.Fluid`

Bases: `pymls.media.medium.Medium`

Represent a fluid medium

**rho**

*float* – density

**c**

*float* – sound speed

### Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	For a fluid, does nothing.

`update_frequency(omega)`

For a fluid, does nothing.

## pymls.media.eqf

**class** `pymls.media.eqf.EqFluidJCA`

Bases: `pymls.media.medium.Medium`

Represents an equivalent fluid medium with the Johnson-Champoux-Allard approach

The Johnson-Champoux-Allard model is described in references [1], [2] and [3].

**phi**

*float* – Porosity

**sigma**

*float* – Flow resistivity

**alpha**

*float* – Tortuosity

**Lambda\_prime**

*float* – Thermal characteristic length

**Lambda**

*float* – Viscous characteristic length

## Notes

Populates the instance with the following attributes (and a few others left undocumented)

*rho\_eq\_til* Frequency dependent equivalent density

*alpha\_til* Frequency dependent Johnson’s tortuosity

*alpha\_prime\_til* Frequency dependent Champoux-Allard’s tortuosity

*K\_eq\_til* Frequency dependent equivalent compressibility

*c\_eq\_til* Frequency dependent equivalent speed of sound

## References

## Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	Computes the JCA parameters (see Notes on the class).

`_compute_missing()`

Computes the required constant parameters missing from the definition

For a JCA equivalent fluid, the shear modulus  $N$  is computed.

`update_frequency(omega)`

Computes the JCA parameters (see Notes on the class).

**Parameters** `omega` – Circular frequency of interest

## pymls.media.air

`class pymls.media.air.Air`

Bases: `pymls.media.medium.Medium`

## Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(*args, **kwargs)</code>	Computes parameters' value for the given circular frequency

`from_yaml`  

**update\_frequency** (*\*args, \*\*kwargs*)  
 Computes parameters' value for the given circular frequency

### Notes

Implemented in derived classes

## pymls.media.elastic

**class** `pymls.media.elastic.Elastic`  
 Bases: `pymls.media.medium.Medium`

### Methods

<code>from_dict(*a, **kw)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	Computes parameters' value for the given circular frequency

**\_compute\_missing** ()  
 Computes the required constant parameters missing from the definition

### Notes

Implemented in derived classes

**from\_dict** (*\*a, \*\*kw*)  
 Reads medium definition from a hashmap of params.  
 Load all possible & recognised parameters and run `_compute_missing` to complete the definition.

**Parameters** `parameters` (*dict*) – Linking parameter names and values.

**Raises** `LookupError` – If the parameter definition is incomplete (missing parameters listed in `EXPECTED_PARAMS`).

**update\_frequency** (*omega*)  
 Computes parameters' value for the given circular frequency

### Notes

Implemented in derived classes



## pymls.media.pem

**class** pymls.media.pem.**PEM**

Bases: *pymls.media.eqf.EqFluidJCA*

### Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	Computes the JCA parameters (see Notes on the class).

`_compute_missing()`

Computes the required constant parameters missing from the definition

For a JCA equivalent fluid, the shear modulus  $N$  is computed.

`update_frequency(omega)`

Computes the JCA parameters (see Notes on the class).

**Parameters** `omega` – Circular frequency of interest

## pymls.media.screen

**class** pymls.media.screen.**Screen**

Bases: *pymls.media.pem.PEM*

### Methods

<code>from_dict(parameters)</code>	Reads medium definition from a hashmap of params.
<code>update_frequency(omega)</code>	Computes the JCA parameters (see Notes on the class).

`_compute_missing()`

Computes the required constant parameters missing from the definition

For a JCA equivalent fluid, the shear modulus  $N$  is computed.

`update_frequency(omega)`

Computes the JCA parameters (see Notes on the class).

**Parameters** `omega` – Circular frequency of interest

## 2.7 pymls.utils package

### 2.7.1 Submodules

### 2.7.2 pymls.utils.hdf5\_export module

`pymls.utils.hdf5_export.hdf5_export(filename, S)`

Traverse the Solver  $S$ 's resultset and outputs in to a HDF5 file

### 2.7.3 pymls.utils.indicators module

`pymls.utils.indicators.TL_from_T(T)`  
 Compute the Transmission Loss from the Transmission coefficient

`pymls.utils.indicators.alpha_from_R(R)`  
 Compute the absorption coefficient from the Reflexion coefficient

### 2.7.4 pymls.utils.yaml\_loader module

**class** `pymls.utils.yaml_loader.YamlLoader`

Bases: `object`

Load a multilayer definition from a yaml file

#### Methods

<code>extract_from_yaml</code>	
<code>from_file</code>	
<code>parse_yaml</code>	
<code>yaml_is_valid</code>	

`EXPECTED_FIELDS = {'analysis': {'type': [<class 'list'>, <class 'dict'>], 'item_keys'`

`KEYS_ANALYSIS = {'frequency': ['type', 'value'], 'range': ['type', 'start', 'end', ']`

`MAP_BACKING = {'rigid': <function rigid at 0x7ff707811400>, 'transmission': <function`

`extract_from_yaml (yaml=None)`

`from_file (filename)`

`parse_yaml ()`

`yaml_is_valid()`

### 2.7.5 Module contents

## 2.8 Module contents

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



---

## Bibliography

---

- [1] 4. (1) Johnson *et al.*, “Theory of dynamic permeability and tortuosity in fluid-saturated porous media”, *Journal of fluid mechanics* 176(1), 1987.
- [2] 25. Champoux & J.-F. Allard, “Dynamic tortuosity and bulk modulus in air-saturated porous media”, *Journal of Applied Physics*, 1975, DOI: 10.1063/1.349482.
- [3] J.-F. Allard & N. Atalla, “Propagation of Sound in porous media: modelling sound absorbing materials”, Wiley, 2009, ISBN: 978-0-470-74661-5.



**p**

pymls, 14  
pymls.analysis, 7  
pymls.backing, 7  
pymls.interface, 8  
pymls.interface.interfaces, 7  
pymls.interface.utils, 7  
pymls.layers, 9  
pymls.layers.elastic, 8  
pymls.layers.fluid, 8  
pymls.layers.layer, 8  
pymls.layers.pem, 8  
pymls.layers.screen, 9  
pymls.layers.utils, 9  
pymls.media, 9  
pymls.media.air, 11  
pymls.media.elastic, 12  
pymls.media.eqf, 10  
pymls.media.fluid, 10  
pymls.media.medium, 9  
pymls.media.pem, 13  
pymls.media.screen, 13  
pymls.solver, 5  
pymls.utils, 14  
pymls.utils.hdf5\_export, 13  
pymls.utils.indicators, 14  
pymls.utils.yaml\_loader, 14





## Symbols

`_compute_missing()` (pymls.media.elastic.Elastic method), 12

`_compute_missing()` (pymls.media.eqf.EqFluidJCA method), 11

`_compute_missing()` (pymls.media.medium.Medium method), 10

`_compute_missing()` (pymls.media.pem.PEM method), 13

`_compute_missing()` (pymls.media.screen.Screen method), 13

## A

Air (class in pymls.media.air), 11

alpha (pymls.media.eqf.EqFluidJCA attribute), 11

alpha\_from\_R() (in module pymls.utils.indicators), 14

analyses (pymls.solver.Solver attribute), 6

Analysis (class in pymls.analysis), 7

## B

backing (pymls.solver.Solver attribute), 5

## C

c (pymls.media.fluid.Fluid attribute), 10

check\_is\_complete() (pymls.solver.Solver method), 6

compute\_fields() (pymls.solver.Solver method), 6

## E

Elastic (class in pymls.media.elastic), 12

elastic\_fluid\_interface() (in module pymls.interface.interfaces), 7

elastic\_pem\_interface() (in module pymls.interface.interfaces), 7

EqFluidJCA (class in pymls.media.eqf), 10

EXPECTED\_FIELDS (pymls.utils.yaml\_loader.YamlLoader attribute), 14

extract\_from\_yaml() (pymls.utils.yaml\_loader.YamlLoader method), 14

## F

Fluid (class in pymls.media.fluid), 10

fluid\_elastic\_interface() (in module pymls.interface.interfaces), 7

fluid\_pem\_interface() (in module pymls.interface.interfaces), 7

from\_dict() (pymls.media.elastic.Elastic method), 12

from\_dict() (pymls.media.medium.Medium method), 10

from\_file() (pymls.utils.yaml\_loader.YamlLoader method), 14

from\_yaml() (in module pymls.media), 9

## G

generic\_interface() (in module pymls.interface.utils), 7

generic\_layer() (in module pymls.layers.utils), 9

## H

hdf5\_export() (in module pymls.utils.hdf5\_export), 13

## I

IncompleteDefinitionError, 5

## K

KEYS\_ANALYSIS (pymls.utils.yaml\_loader.YamlLoader attribute), 14

## L

Lambda (pymls.media.eqf.EqFluidJCA attribute), 11

Lambda\_prime (pymls.media.eqf.EqFluidJCA attribute), 11

Layer (class in pymls.layers.layer), 8

layers (pymls.solver.Solver attribute), 5

## M

MAP\_BACKING (pymls.utils.yaml\_loader.YamlLoader attribute), 14

media (pymls.solver.Solver attribute), 5

Medium (class in pymls.media.medium), 9

MEDIUM\_TYPE (pymls.media.medium.Medium attribute), 9

MODEL (pymls.media.medium.Medium attribute), 9

## N

name (pymls.media.medium.Medium attribute), 9

## O

omega (pymls.media.medium.Medium attribute), 9

## P

parse\_yaml() (pymls.utils.yaml\_loader.YamlLoader method), 14

PEM (class in pymls.media.pem), 13

pem\_elastic\_interface() (in module pymls.interface.interfaces), 7

pem\_fluid\_interface() (in module pymls.interface.interfaces), 7

phi (pymls.media.eqf.EqFluidJCA attribute), 10

pymls (module), 14

pymls.analysis (module), 7

pymls.backing (module), 7

pymls.interface (module), 8

pymls.interface.interfaces (module), 7

pymls.interface.utils (module), 7

pymls.layers (module), 9

pymls.layers.elastic (module), 8

pymls.layers.fluid (module), 8

pymls.layers.layer (module), 8

pymls.layers.pem (module), 8

pymls.layers.screen (module), 9

pymls.layers.utils (module), 9

pymls.media (module), 9

pymls.media.air (module), 11

pymls.media.elastic (module), 12

pymls.media.eqf (module), 10

pymls.media.fluid (module), 10

pymls.media.medium (module), 9

pymls.media.pem (module), 13

pymls.media.screen (module), 13

pymls.solver (module), 5

pymls.utils (module), 14

pymls.utils.hdf5\_export (module), 13

pymls.utils.indicators (module), 14

pymls.utils.yaml\_loader (module), 14

## R

register() (pymls.layers.layer.Layer method), 8

reinit() (pymls.layers.layer.StochasticLayer method), 8

resultset (pymls.solver.Solver attribute), 6

rho (pymls.media.fluid.Fluid attribute), 10

rigid() (in module pymls.backing), 7

rigid\_interface() (in module pymls.interface.utils), 7

## S

Screen (class in pymls.media.screen), 13

sigma (pymls.media.eqf.EqFluidJCA attribute), 11

solve() (pymls.solver.Solver method), 6

Solver (class in pymls.solver), 5

StochasticLayer (class in pymls.layers.layer), 8

## T

TL\_from\_T() (in module pymls.utils.indicators), 14

transfert\_elastic() (in module pymls.layers.elastic), 8

transfert\_fluid() (in module pymls.layers.fluid), 8

transfert\_pem() (in module pymls.layers.pem), 8

transfert\_screen() (in module pymls.layers.screen), 9

transmission() (in module pymls.backing), 7

## U

update\_frequency() (pymls.layers.layer.Layer method), 8

update\_frequency() (pymls.media.air.Air method), 12

update\_frequency() (pymls.media.elastic.Elastic method), 12

update\_frequency() (pymls.media.eqf.EqFluidJCA method), 11

update\_frequency() (pymls.media.fluid.Fluid method), 10

update\_frequency() (pymls.media.medium.Medium method), 10

update\_frequency() (pymls.media.pem.PEM method), 13

update\_frequency() (pymls.media.screen.Screen method), 13

## Y

yaml\_is\_valid() (pymls.utils.yaml\_loader.YamlLoader method), 14

YamlLoader (class in pymls.utils.yaml\_loader), 14