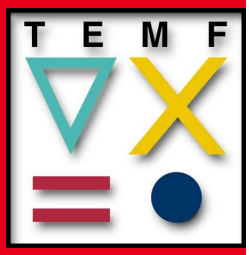
Octave Interface for Onelab

Alexander Krimm

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Stand: April 29, 2014

Abstract

Open Numerical Engineering LABoratory (ONELAB)[1] is a Project that aims to simplify the use of various Finite Element Software by providing a client/server based Protocoll for different solvers to exchange data. Non native clients can be used by wrapping their input data and the calls to them into *.ol files, but it is more comfortable for the solvers to interface directly with the ONELAB Server. The reference server is included in the meshing software gmsh[2] and includes C++ Headers and a Python module. The goal of the Octave Interface for Onelab is to provide similar functionalities for solvers written in octave[3], as this is a language often used in the field of electromagnetical field calculations.

1 Installation

1.1 Installation from Octave Package

If you've got the `onelab-?.?.?.tar.gz` you can install it from octave by running:

```
pkg install onelab-?.?.?.tar.gz
```

This installs the octave Package into your home-directory or if you've got the necessary privileges into the respectiv global directory.

1.2 Compilation from Source

Onelab comes with a small Makefile, so on unixoid systems a simple `make all` in the `src` directory should do the job. To use the octfile compiled this way you have to add your build directory to the octave-path with `addpath`.

1.3 Generating the Octave package from source

If you want to change things in the code and install your modified version as an octave package you can build one with `make pkg`. Be sure to run `make clean` before. The files in the parent directory of src are also important for building the package.

2 Usage

To use onelab in octave make sure you have a recent gmsh verion[4] and the package is installed somewhere octave can find it.

2.1 Setting up the connection

To start your own Scripts with onelab you have to add them as solvers from the gmsh interface via "add solver". This results in gmsh starting your Skript once to initalize and for everytime you click the "run"-button. To be run your skripts should start with the proper shebang:

```
#!/usr/bin/octave -qf
```

To establish the connection your script has to read out where to connect from the options and open the connection with the `ol_connect` command.

1 `http://onelab.info/`
2 `http://www.geuz.org/gmsh/`
3 `http://www.gnu.org/software/octave/`
4 The Version used to test this was 2.8.4

```
% load the onelab api
pkg load onelab;
% parameter parsing
for i = [1:nargin]
  if (i+2 > nargin)
    error('invoke with -onelab name server_addr');
  elseif (strcmp(argv({i},'-onelab'))
    name = argv({i+1};
    addr = argv({i+2};
    break;
  endif
endfor
% Start the client
ol_client(name, addr);
```

To find out what is expected from our solver we have to read out the Action Parameter from onelab, for example with:

```
% Quit if the requestet action is not to compute
action = ol_getParameters([name '/Action']);
if (!strcmp(action{1}.value, 'compute'))
    printf('Nothing to do');
    ol_disconnect()<++>;
    exit;
endif
```

Before exiting the script should properly close the connection with:

```
ol_disconnect(;
```

2.2 Reading and Writing Parameters

To read Parameters from ONELAB the package provides the `ol_getParameters` command. If called without argument it returns all parameters from the server, if called with a parameter name, it returns just that Parameter or an empty list. Note that the function always returns cell arrays, so you have to index them with `{i}` even if just one element is returned. The Parameters itself are represented as maps, the most important keys are name, value, type, label, help and choices.

To write Parameters you can call the function `ol_setParameter` either with a single parameter containing a structure like the one returned by `ol_getParameters` where the fields name, value and type are mandatory or with more parameters name, type and value.

2.3 Launching Subclients

If you want to use other software from inside your solver onelab allows you to launch “subclients”. They can be either blocking or non-blocking:

```
% run subclient and wait until it is finished before continuing
ol_runBlockingSubClient('subclient', ...
'getdp ReadPoint.pro -res LaplacianNeumann.res -pos readPoint');
% run other subclient
ol_runNonBlockingSubClient('subclient2', ...
'super-cool-simulation-software -solve -real -fast');
% do some stuff
ol_waitOnSubClients();
```

```
% do someting with the results and exit
```

The first parameter is the name the subclient gets passed and is also prepended to it's debug output in the gmsh console.

2.4 Other useful commands

Other useful commands are mostly for logging:

```
% show a red error in the onelab console
ol_sendError('something went wrong')
% show informative output in the onelab console
ol_sendInfo('hey there')
% show warnings in the onelab console
ol_sendWarning('this looks strange')
% display progress in the gmsh status bar
ol_sendProgress('reading file: input.txt')
```

You can also send a Merge File request to force gmsh to read and display msh files:

```
ol_sendMergeFileRequest('foo.msh');
```

2.5 Interactive use

Often when you have computation intensive calculations running you want to check if the results can be right while the computations are still running. You can use ONELAB from the octave terminal to do exactly that. Just use `interactive.m` as a solver and start your regular solver with `ol_runNonBlockingSubClient`. While this works on some systems you might have to modify the `interactive.m` script to lauch the right terminal and respect the escaping rules of your shell.

3 Examples

The octave onelab package is at the Moment shipped with 2 examples, one being a simple standalone simulation of a pendulum, the other using getdp as a subclient and reading back field values from it.

3.1 pend.m

Just run the solver by adding it from within gmsh and after clicking run it will visualize the movement of the pendulum and export values for the angles for the timesteps. You can then visualize them over the time.

3.2 LaplacianNeumann

This solver shows the use of subclients to exchange data between octave and other onelab solvers. The problem is a simple laplacian-neumann problem in the unit square. The excitation can be varied with a onelab Parameter. This whole Problem is described in the LaplacianNeumann.pro file. The octave solver `example.m` calls getdp[5] on this problem with different parameter values. Another call to getdp is made to export the values of specific points to onelab, as specified in the ReadPoint.pro file. The octave script then reads the value of these 2 points and multiplies their values returning the result to another ONELAB Parameter.

4 License

[5] `http://geuz.org/getdp/`

[6] `http://www.gnu.org/licenses/lgpl.html`