PLECS

Tutorial

Running PLECS Standalone Simulations from Matlab

Using the JSON-RPC interface

Tutorial Version 1.0

1 Introduction

This tutorial explains how to run a PLECS Standalone simulation using Matlab® and JSON-RPC. For this tutorial you will need:

- A working Matlab® installation
- A working PLECS Standalone installation
- An internet connection to download the latest version of the JSON-RPC client from Plexim's GitHub page.

The tutorial is split into three sections: Installation of the JSON-RPC client, setting up PLECS Standalone and writing the simulation script in Matlab®.

Before you begin Ensure that you have the files `standalone_from_matlab.plecs`, `simulation_script_1.m`, `simulation_script_2.m` and `simulation_script_3.m` in your working directory. The files are available together with this PDF in the tutorials section at `www.plexim.com`.

2 Install the JSON-RPC client

Your Task:

1. Please visit `https://github.com/plexim/matlab-jsonrpc` and download the latest version of the JSON-RPC client by clicking on "Code" and then "Download ZIP".
2. Extract the `matlab-jsonrpc-main.zip` file.
3. Open Matlab and add the folder `matlab-jsonrpc-main` that contains the JSON-RPC client to the Matlab® Search Path (see for example this `link` for an explanation on how to do this).

3 Setting up PLECS Standalone

Your Task:

1. Open PLECS Standalone
2. Enable the RPC interface in the PLECS Preferences: Click on **File + PLECS Preferences** and enable the RCP Interface on port *1080*. On MacOS the preferences can be found under **PLECS + PLECS Preferences**.
3. Open the model `standalone_from_matlab.plecs`

The model consists of a buck converter with analog controls. More information about the model can be found in the description of the "Buck converter with parameter sweep" demo model.

4 Simulation Script

4.1 Single Simulation

Your Task:

1. Open Matlab® and create a new file called `simulation_script.m` in the same folder as the `standalone_from_matlab.plecs` file.
2. In the m-file editor, write the following code:

```
proxy = jsonrpc('http://localhost:1080', 'Timeout', 10);
```

   This creates a new JSON-RPC proxy object with a connection timeout of 10 seconds
3. Load the PLECS model with the following code:

```
path = pwd;
model_name = 'standalone_from_matlab';
proxy.plecs.load([path '/' model_name '.plecs']);
```

   The command “pwd” prints the path of the current working directory that contains the PLECS model we want to simulate.
4. Create a simulation struct and run a simulation

```
simStruct = struct('ModelVars', struct('varL', 50e-6));
results = proxy.plecs.simulate(model_name, simStruct);
```

5. Execute the script in Matlab®.

At this stage your simulation script should look like the example solution `simulation_script_1.m`.

4.2 Parallel Simulations

In this task we set up a parameter sweep using parallel simulations. The concept of performing the simulations is the same as for a single simulation but instead of only one set of simulation parameters we hand over a cell array that contains the parameter sets for each individual simulation.

Your Task:

1. Create a new file called `simulation_script_2.m` with the following content:

```
proxy = jsonrpc('http://localhost:1080', 'Timeout', 10);
path = pwd;
model_name = 'standalone_from_matlab';
proxy.plecs.load([path '/' model_name '.plecs']);
simStruct = struct('ModelVars', struct('varL', 50e-6));
```

   This is the same code as in the previous task but without the `plecs.simulate` command.
2. Let’s clear all traces in the Scope so that we can add later the traces of all parallel simulation runs.

```
proxy.plecs.scope([model_name '/Scope'], 'ClearTraces');
```

3. Now we create a cell array that contains the parameters for each individual simulation.

```
% Set value for L1 to be swept
inductorValues = 40:20:220; % in uH

% Allocate memory for cell array
simStructs = cell(size(inductorValues));

% Initialize simStruct as cell array with all values for L1
for ix = 1:length(inductorValues)
   simStructs{ix}.ModelVars.varL = inductorValues(ix) * 1e-6;
   simStructs{ix}.Name = ['L=' mat2str(inductorValues(ix)) 'uH'];
end
```

in the for loop every member of the cell array is assigned a "ModelVars" struct and a name. The name is useful to later identify the simulation result. For example, the name can be used to label the trace in the scope.

4 Now we use the cell array for the `plecs.simulate` command:

```
results = proxy.plecs.simulate(model_name, simStructs);
```

5 In the last step we use the `plot` function to plot one of the simulation results (simulation index 5):

```
plot(results(5).Time,results(5).Values);title(simStructs{5}.Name)
```

6 Execute the script in Matlab®.

The script runs all simulations in parallel, however, only one of the simulation results is shown in the Scope and the Matlab® plot window. Please note at this point, that those results do not necessarily be the same because all simulations run in parallel and it is not clear which simulation finishes first.

In the next step we create a callback function that holds the traces and post-processes the results of all simulations.

At this stage your simulation script should look like the example solution `simulation_script_2.m`.

4.3 Adding a Callback Function

To use the full potential of parallel simulations in the context of automated parameter sweeps, we add a callback function to the argument of the `plecs.simulate` command.

Your Task:

1 Continue writing the simulation from the previous step `simulation_script_2.m`. At Before the `plecs.simulate` command add the following code:

```
callback = sprintf([ ...
'if ischar(result)\n' ...
'   disp(["Simulation errors can be asserted in this message."]);\n' ...
'else\n' ...
'   plecs(''scope'', ''./Scope'', ''HoldTrace'', name);\n' ...
'   result = max(result.Values(1,:));\n' ...
'end\n' ...
```

```
]);
```

This code writes a callback function in the Octave language which can be processed by PLECS Standalone after every simulation run. This is a useful tool to analyze certain key figures from each simulation run and add a parametric plot at the end of the sweep. Here only the maximum current value is returned after the simulation. The callback function also holds a trace in the Scope so that all results can be inspected in the PLECS Scope after the script has finished.

2 Change the call of the `plecs.simulate` command to:

```
results = proxy.plecs.simulate(model_name, simStructs, callback);
```

3 Now we can add a parametric plot of maximum current vs. inductance value

```
plot(inductorValues,results,'-*');
title('Maximum current in A vs. Inductor Values in uH')
```

4 Execute the script in Matlab®.

At this stage your simulation script should look like the example solution `simulation_script_3.m`.

5 Conclusion

In this tutorial you learned how to run PLECS Standalone Simulations from Matlab®. First a JSON-RPC client needs to be installed and configured in the simulation script and then the Standalone simulation is called over the JSON-RPC interface. The powerful feature of parallel simulations is explored at the end of this tutorial. This feature is only available in PLECS Standalone.

Revision History:

Tutorial Version 1.0	First release

How to Contact Plexim:

☎	+41 44 533 51 00	Phone
	+41 44 533 51 01	Fax
✉	Plexim GmbH Technoparkstrasse 1 8005 Zurich Switzerland	Mail
@	`info@plexim.com`	Email
	`http://www.plexim.com`	Web

PLECS Tutorial