THE SIMULATION PLATFORM FOR POWER ELECTRONIC SYSTEMS

Web-Based Simulation Manual Version 3.7

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

Web-Based Simulation Manual

Contents

1

Introduction

PLECS Web-Based Simulation (WBS) is a software technology for publishing simulation models on web pages. These simulation models can be made available to large audiences using the world-wide web. WBS provides the web page visitor with a graphical user interface (GUI) for running PLECS simulations embedded in a web page. There are two main application areas of WBS:

- **Marketing:** PLECS WBS allows component manufacturers to offer product information not only in the form of pictures and data sheets, but also allows for customer interaction with the products as part of simulated real systems of different application areas. Also, WBS provides a tool for non-technical employees to discuss customer-specific needs in a sales conversation.

- **Education:** Universities can use PLECS WBS to guide students through areas of electrical engineering. The lecturer can decide which part of the model or simulation can be modified by the student and therefore set a focus on a specific topic. WBS also has the potential to enhance existing online exercises and exams concerning power electronics and motor drives. The same approach can be used in companies for internal education and training.

Setup

The setup of PLECS WBS is given in Fig. 1.1. The overall system includes three different parties: the web site visitor, web site owner and the simulation provider. Each one provides a certain hardware and software platform to execute part of the code.

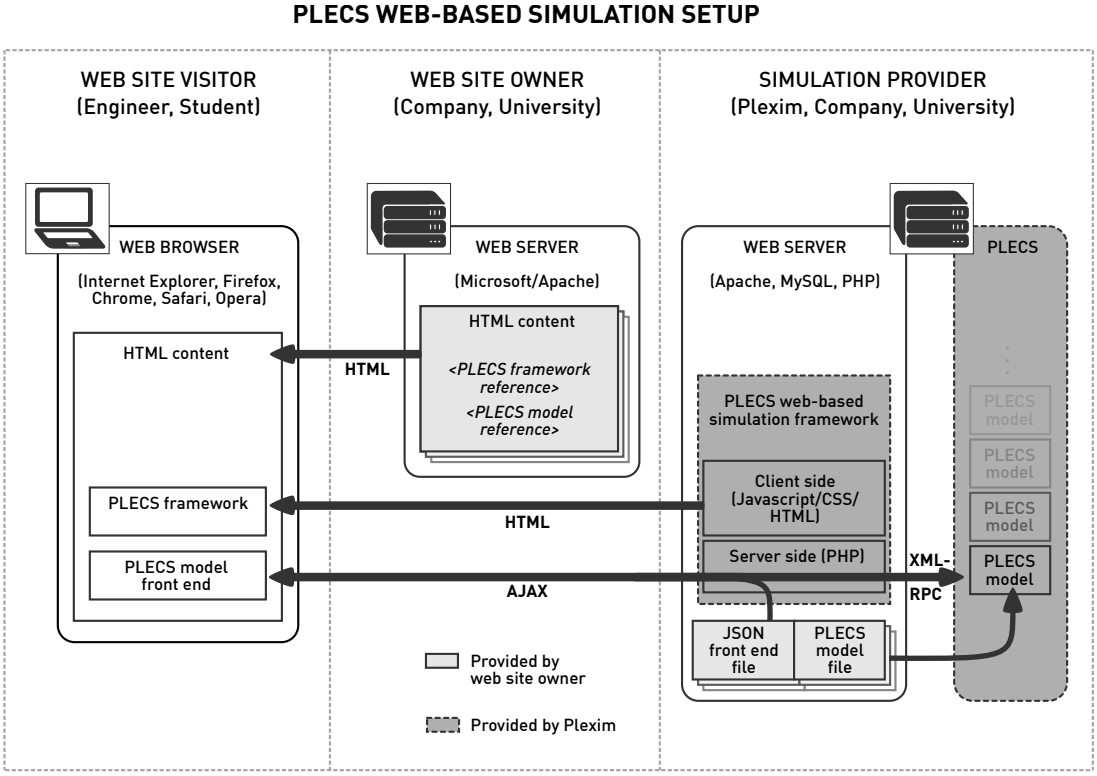

Figure 1.1: Setup of PLECS Web-Based Simulation.

The web site visitor uses a web browser on any device (e.g. desktop PC, laptop, tablet or smart phone) to display a specific web page from the company's web server. Besides the normal HTML content the page contains references to the PLECS framework and to a specific PLECS model. Inside the browser these references are resolved and the WBS GUI is displayed.

The web site owner processes the visitor's page load requests on a web server (e.g. Microsoft Windows Server, Apache Server). It contains different HTML pages, each of them with the same PLECS framework reference, and a different, or possibly the same, PLECS model reference.

The simulation provider supplies client side code that is executed on the visitor's machine. Server side scripts are executed on the simulation

provider's web server. Together, the client and server side code represent the PLECS WBS framework. In the background, a PLECS server application is running and executing all simulation requests from the web site visitor. The communication between PLECS and the server side code happens via an XML-RPC interface[1]. The PLECS WBS server version of PLECS Standalone is able to run several simulations of different models in parallel depending on the number of CPU cores available.

Workflow

As apparent from Fig. 1.1, the web site owner is involved in three different parts of PLECS WBS. The owner needs to provide:

- The HTML content. A PLECS WBS model can be seamlessly integrated in every web page that contains a PLECS framework reference in either the <*head*> or <*body*> section of the HTML code.
- The JSON front end file. It defines how the user can interact with the PLECS model and simulation.
- The PLECS model file. This file is needed to run a simulation.

As creating content for PLECS WBS usually involves different people within an organization the workflow depicted in Fig. 1.2 is proposed.

An **application engineer** develops a PLECS model for WBS or modifies an existing one. He decides which parameters can be modified and which subsystems are visible to the end user by editing the corresponding JSON file. To check the appearance and usability of the developed model for WBS a local server environment is installed on the application engineer's machine (e.g. MAMP or XAMPP for OS X, Windows or Linux). In the last step the engineer will upload the PLECS model and JSON front end file to the simulation server. This process is repeated for every WBS model.

An **IT administrator** needs to enable scripting, allow customer-specific HTML attributes and embed the PLECS WBS framework into the content management system (CMS). This will be done only once since all WBS models share the same framework. It is recommended to set up a test page referring to a generic WBS model and check if the model is displayed properly on various web browsers and devices.

[1]XML-RPC stands for "Extensible Markup Language Remote Procedure Call" and uses HTTP as a transport mechanism.

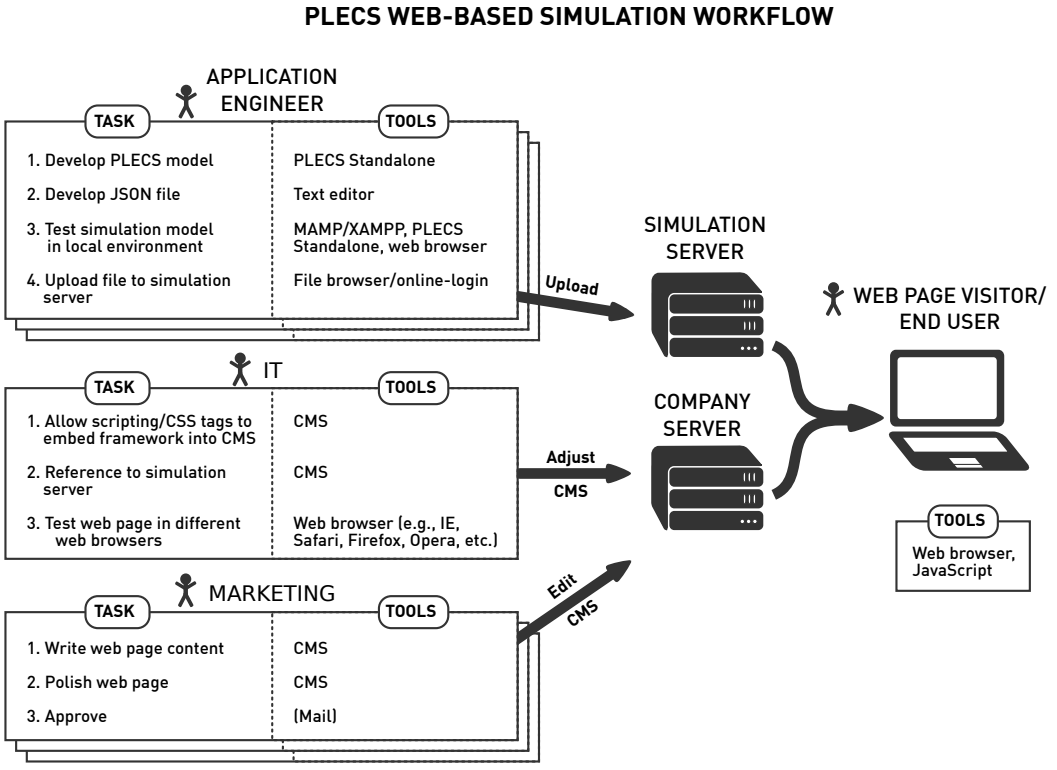

Figure 1.2: PLECS Web-Based Simulation Workflow.

Technical **marketing** is usually responsible for the overall appearance of the web site. They write the web page content, polish the final design, and approve each web page before taking them online.

The **web page visitor** in the end only needs a web browser with JavaScript enabled to perform a simulation. There is no need to download any software or browser plug-ins.

2

Quick Start

This chapter explains how you can install a development environment for PLECS Web-Based Simulation (WBS) on your local computer. You will then develop your first own WBS model and test it with your preferred web browser.

As stated in the previous chapter a development environment for PLECS WBS requires different software tools:

- PLECS Standalone is needed to create and modify a PLECS model. The PLECS Standalone application in version 3.6 and higher will also function as a simulation server on your local computer. No additional WBS license is required as long as this feature is used only for development and test purposes. In contrast to the the PLECS WBS server application, PLECS Standalone limits the number of WBS models simultaneously open to 1.

- The PLECS WBS framework is shipped with your PLECS Standalone distribution and can be placed in a directory of your choice.

- A web server environment must be running on your local computer, to let you test the developed models in a web browser before publishing them. The WBS model embedded in a HTML page can be verified and improved without interfering with the operation of the public web server.

- Last but not least a text editor (preferably with JSON syntax highlighting) is needed for writing the JSON file that defines the GUI of the WBS model.

Installation of Local Web Server Environment

In the following section the installation process of a local server environment is discussed on the basis of MAMP and XAMPP – two freeware tools that are available for Windows and MAC OS X. The XAMPP installation can also be carried out on a Linux operating system and works analogous to the installation on Windows.

Windows Operating System (explained using XAMPP)

1. In PLECS Standalone, select the entry **PLECS Extensions...** from the menu item **File**. This will open a dialog box. In the tab **Web** choose a web framework path, e.g. *C:/webframeworks*. From the available web frameworks choose *version_1* and install the selected framework at the path specified.
2. Download and install XAMPP. XAMPP is a local server environment consisting of an Apache server, MySQL database system, and PHP interpreter on a desktop computer that supports Windows, MAC OS, and Linux operating systems. Within this environment the full PLECS WBS setup and work flow can be tested without any need to set up an online network.
3. Open the configuration file "httpd.conf" in the XAMPP installation directory (e.g., *C:/xampp/apache/conf/httpd.conf*) and change/set the two lines:

   ```
   DocumentRoot "C:/webframeworks/version_1/public"
   <Directory "C:/webframeworks/version_1/public">
   ```

4. Start PLECS 3.7 and enable XML-RPC on port 1080 in the PLECS preferences menu.
5. Copy the default configuration file *webframeworks/version_1/private/config.ini.default* to *webframeworks/version_1/private/config.ini*. This file can be edited to adjust the configuration of PLECS WBS on your system.
6. Open XAMPP and start the Apache and MySQL servers. Wait until the Apache and MySQL servers are started. Use (preferably) Internet Explorer, Firefox (>=24.0), Chrome, or Opera and open the web site: *localhost*. You should see the PLECS web interface with some example circuits listed.

7. Click on the link "Buck converter". You should see a schematic of a buck converter and a PLECS Scope. In the background the same simulation model is loaded in PLECS 3.7. Clicking the "Simulate" button should give an instantaneous result of current and voltage characteristics in the Scope. Note that a PLECS window will be displayed in the background or pop-up that shows the buck converter circuit and performs the actual simulation.

OS X Operating System (explained using MAMP)

1. In PLECS Standalone, select the entry **PLECS Extensions...** from the menu item **File**. This will open a dialog box. In the tab **Web** choose a web framework path, e.g. *~/webframeworks*. From the available web frameworks choose *version_1* and install the selected framework at the path specified.

2. Download and install MAMP. MAMP is a local server environment consisting of an Apache server, MySQL database system, and PHP interpreter on a desktop computer that supports Mac OS only. Within this environment the full PLECS WBS setup and work flow can be tested without any need to set up an online network.

3. Start MAMP and click on "Preferences...", then "Apache", and set the document root to the folder named "public" inside the "webframeworks" folder (e.g., *~/webframeworks/version_1/public*).

4. Start PLECS 3.7 and enable XML-RPC on port 1080 in the PLECS preferences menu.

5. Copy the default configuration file *webframeworks/version_1/private/config.ini.default* to *webframeworks/version_1/private/config.ini*. This file can be edited to adjust the configuration of PLECS WBS on your system.

6. Open MAMP and click "Start Servers". Wait until the Apache and MySQL Server are started. Use (preferably) Safari, Firefox, Chrome, or Opera and open the web site: *localhost:8888*. You should see the PLECS web interface with some example circuits listed.

7. Click on the link "Buck converter". You should see a schematic of a buck converter and a PLECS Scope. In the background the same simulation model is loaded in PLECS 3.7. Clicking the "Simulate" button should give an instantaneous result of current and voltage characteristics in the

Scope. Note that a PLECS window will be displayed in the background or pop-up that shows the buck converter circuit and performs the actual simulation.

Overview of Data Structure

After installation of the XAMPP or MAMP server the provided examples (also accessible online at demo.plexim.com) should work out of the box. These examples provide a foundation for customization or creating new implementations. The descriptions of the most important folders and files are given in Fig. 2.1.

/

- **models**... Directory containing PLECS models with corresponding JSON files and thermal descriptions folders. The file names and thermal description folder names follow the pattern: *model_name.plecs*, *model_name.json* and *model_name_plecs*/.
  - **plexim**... Example PLECS and JSON files provided by Plexim.
  - **custom**... PLECS and JSON files provided by the user.
- **private**... Directory with only local access on the web server.
  - **config.ini.default**... Default configuration file of the PLECS web service including host information, model path, model expiry time and scope font style and size.
  - **embed-example.html**... Example that shows how to embed a PLECS model into a web page.
- **public**... Directory with public access.
  - **plexim**... Example HTML pages with framework and WBS model references provided by Plexim.
    - **index.html**... Web page with list of WBS examples.
  - **css**... Style sheets for the WBS GUI and general web page appearance.
  - **custom**... Custom HTML and style sheets for the general web page appearance defined by the user.

Figure 2.1: Directory tree of the PLECS WBS framework.

Create Your First PLECS WBS Model

Setting up a PLECS WBS model consists of three different tasks: Creating the model in PLECS, writing a JSON file to define the interaction of the user with the model on a web page and setting up the actual web page that loads the model. These three tasks are discussed step-by-step using the example of a Buck converter. The web page set up is done first in order to check progression of the PLECS Web-Based Simulation model at every work step. To follow this tutorial a valid PLECS Standalone license is needed. Notice that with a PLECS Standalone license only one web-based model can be simulated at a time. To be able to serve multiple users and parallel PLECS simulations online a PLECS web server license is needed.

Web Page

In the next step the HTML page is created that actually loads the WBS model. Write the following code in a file named *tutorial_circuit.html* and place this file inside the *public/custom/* folder. More information about the individual lines of this code can be found on page 23 of this manual.

```
<!DOCTYPE html>
<html xmlns:ng="http://angularjs.org" id="ng-app" ng-app="plecsModel">
  <head>
    <title> Tutorial circuit</title>
    <link href="../css/plecs.css" rel="stylesheet" type="text/css" />
    <link href="../plexim/css/plexim_theme.css" rel="stylesheet" type="text/css" />
    <script src="../js/angular.min.js"></script>
    <script src="../js/plecs.min.js"></script>
  </head>
  <body>
    <a href="../index.html">Home</a>
    <h1>Tutorial Circuit</h1>
    <noscript class="plecs">
    <div class="msg msg-red"> Error: JavaScript is disabled in your browser.
       Please enable JavaScript to use this page.</div>
    </noscript>
    <div pl-model="tutorial_circuit" pl-path="custom">
    </div>
  </body>
</html>
```

When navigating to *localhost/custom* (or *localhost:8888/custom* using MAMP) with a web browser an error message is shown because there doesn't exist a PLECS and JSON file for this specific model "tutorial_circuit" so far. The creation of these two files will be discussed in the next sections of this tutorial.

PLECS Model

In this section you will build a PLECS model of a buck converter that is suitable for WBS. The model is based on the PLECS workshop tutorial "Thermal Simulation of a Buck Converter using PLECS".

1. Open PLECS 3.7 and create a new model and save it under the name *models/custom/Tutorial_Circuit.plecs*.

2. Set up the circuit as shown in Fig. 2.2 with the parameter values in Tab. 2.1. The resistor and duty cycle take a variable parameter value "VarR" and "D", respectively. In the initialization script the variables are set to 5 Ω and 0.5, respectively. Later in the online simulation the user should be able to change these values in a corresponding parameter field. Note that you should leave enough space between the capacitor and resistor so that a parameter field fits in between them on the web site.

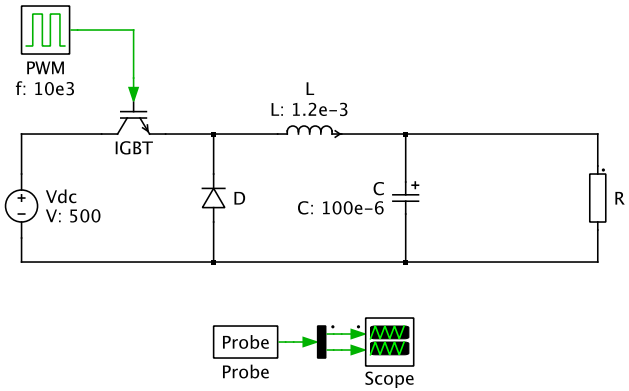

Figure 2.2: Schematic of a buck converter for PLECS WBS.

3. First drag the DC voltage source and then the resistor and inductor into the probe block. Check the source voltage, resistor voltage and inductor current. The number of outputs of the multiplexer is set to [2 1]. Open the scope and label the individual traces and plots for easier understanding by the user. Remember that the user will later not have the full access to the model so it is important to give as much information inside the scope as possible (including e.g. units or specific component labels, etc.).

Component	Value	Range
R	VarR	0.1 Ω – 100 Ω
C	100 μF	constant
L	1.2 mH	constant
Vdc	500 V	constant
Duty cycle	D	0.01 – 0.99
Frequency f	10 kHz	constant

Table 2.1: Parameter values.

4. The "RADAU" solver is chosen in the simulation parameters window **(Ctrl + E)**. This is due to the fact that a web user can later change the resistor value and create a "stiff" system that the DOPRI solver is not able to solve. Also the "Refine factor" is set to 10 in order to increase the smoothness of the output curves in the scope. The "Stop time" is set to 20 ms.

JSON File

In this section a basic JSON file is written in order to get a first overview of the PLECS WBS model. More information about the different keywords can be found in section 3 of this manual.

1. Create a file named *models/custom/tutorial_circuit.json* with a text editor of your choice. Use preferably UTF-8 encoding as this is the standard for all JSON files.

2. The minimum required content of the JSON file is the entry about the root schematic. Note that the optional height and width parameters eventually have to be adjusted depending on the graphical design of your buck converter. If no width and hight are specified the natural size of the PLECS schematic is taken as default.

```
{
  "schematics":
  [
    {
```

```
    "path" : "",
    "height" : 280,
    "width" : 430,
    "resize" : "",
    "display" : "always"
   }
  ]
}
```

The empty path refers to the root schematic. The values for the resize and display keywords are optional while the default values for these are "" and "always", respectively.[1]

3. In the next steps you will create the JSON file entries for the two user parameters (duty cycle and resistance values). To display the parameter field for the load resistance in the WBS schematic use the following JSON code snippet:

```
"paramtables":
[
 {
  "display": "schematic",
  "component": "R",
  "labelpos": "left",
  "parameters":
  [
   {
   "variable": "VarR",
   "label": "R",
   "unit": "Ω",
   "min": 0.1,
   "max": 100,
   "value": 5
   }
  ]
 }
]
```

The value "left" for the "labelpos" keyword places the parameter field on the left hand side of the component named "R", i.e. the resistor, where the end user is able to provide a value for the resistance. The keyword "variable" defines which variable in PLECS is assigned by the user input, and "min" and "max" define upper and lower limits for the input. To write the symbol Omega (Ω) either use the Unicode character U+03A9 or the Java source code "\u03A9".

[1] In a JSON file an empty value ("") can be replaced by the expression *null*.

4. In the same manner as in the last step use the keyword "dialog" instead of "schematic" to display the parameter field for the duty cycle in a dialog box. The minimum and maximum value of the duty cycle are 0.01 and 0.99, respectively. The keyword "labelpos" is ignored and can be omitted.

5. Now make the scope visible to the WBS user by using the following JSON code:

```
"scopes":
[
 {
  "path": "Scope",
  "height": 400,
  "width": 400,
  "resize": "xy",
  "display": "open"
 }
]
```

   The value "open" means the scope is initially open but can later be closed by the user. Also, the width and height are both initially fixed to 400 pixels but can be resized in the x and y directions.

6. A transient simulation is added by default to a web-based simulation if nothing is specified in the JSON file. For the sake of completeness you will add the "simulations" keyword to the JSON file with the following configuration:

```
"simulations":
[
 {
  "label": "Simulate",
  "analysis": ""
 }
]
```

   Here, "" stands for the default transient simulation.

7. Now save the JSON file and open the website *localhost/custom* (or *localhost:8888/custom* using MAMP) in a web browser of your choice (preferably a current version of Safari, Firefox, Chrome or Opera). Click on the link "Tutorial circuit" and then on the simulation button. After opening the scope the result should look similar to the one given in Fig. 2.3. In case of an error, the WBS framework has an inbuilt syntax validator that shows in which part of the JSON file a syntax error occured. You can also check the syntax of your JSON file online at http://jsonlint.com.

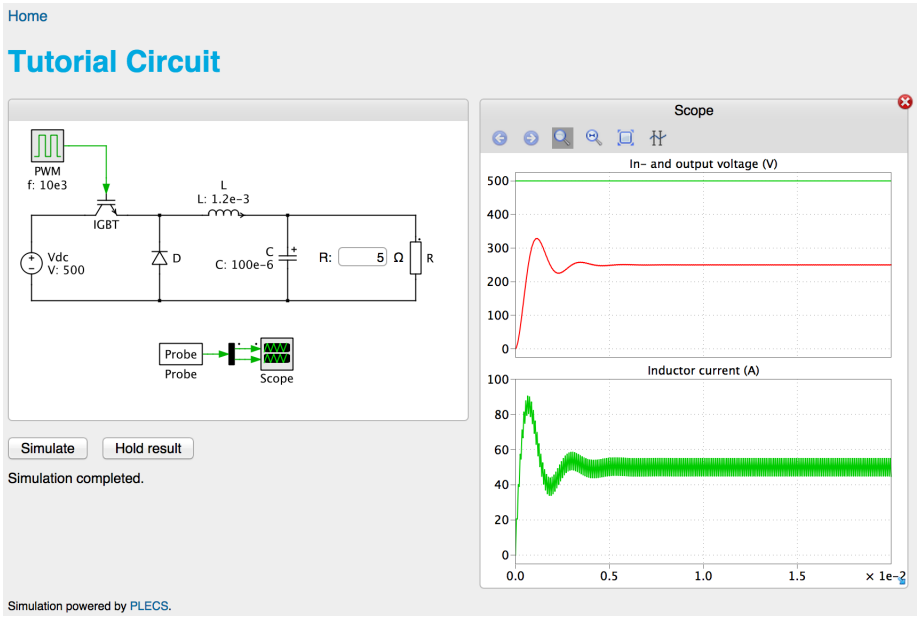

Figure 2.3: Web site with PLECS WBS schematic of a buck converter.

3

Model Development

This chapter explains in detail the syntax of the JSON file and the object structure required for defining the GUI of a WBS model.

JSON Files

JSON stands for "JavaScript Object Notation"[1]. The standard encoding for JSON files is UTF-8. JSON's basic types are:

- **Number** – a signed decimal number that may contain a fractional part and may use exponential E notation. JSON does not allow non-numbers like NaN and inf, nor does it make any distinction between integer and floating-point numbers. (Even though JavaScript uses a double-precision floating-point format for all its numeric values, other languages implementing JSON may encode numbers differently.)
- **String** – a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
- **Boolean** – either of the values *true* or *false*.
- **Array** – an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation where elements are comma-separated.
- **Object** – an unordered associative array (name/value pairs). Objects are delimited with curly braces ({ and }) and use commas to separate each

[1] This section was taken from Wikipedia and extended with some information concerning PLECS WBS.

pair, while within each pair the colon (:) character separates the key or name from its value. All keys must be strings and should be distinct from each other within that object.

- **null** – an empty value, using the word *null*.

JSON generally ignores any whitespace around or between syntactic elements (values and punctuation, but not within a string value). However, JSON only recognizes four specific whitespace characters: the space, horizontal tab, line feed (e.g., $\backslash n$ is needed if a PLECS component name has a line break), and carriage return. JSON does not provide or allow any sort of comment syntax.

Example The following example shows a possible JSON representation describing a person.

Listing 3.1: Example content of JSON file.

```
{
   "firstName": "John",
   "lastName": "Smith",
   "isAlive": true,
   "age": 25,
   "height_cm": 167.64,
   "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021−3100"
   },
   "phoneNumbers": [
      { "type": "home", "number": "212 555−1234" },
      { "type": "fax",  "number": "646 555−4567" }
   ]
}
```

JSON Files for PLECS Web-Based Simulation

A JSON file for PLECS WBS contains a basic object with the following keywords: **"schematic"**, **"scopes"**, **"paramtables"**, **"results"** and **"simulations"**. Each of these keywords are followed by an array, indicated by square brackets ([ ]), that contains the different properties.

The following code listings show the basic structure and syntax for these keywords. All keywords are highlighted in **bold font**, string values in quotation

marks, and containers for strings and numbers in *italic font*. Different possible values for a certain keyword are separated by a vertical line (|) and default values are marked with an underline. All keywords which feature a default value can be omitted. Note that the vertical order of the objects (e.g., scopes and schematics) in a WBS model web page corresponds to their order in the JSON file.

Keyword: "schematics"

With the "schematics" keyword the visibility of a .plecs schematic in the PLECS WBS GUI can be specified. This can be the root schematic or any number of subsystem schematics.

Listing 3.2: Schematics entry in JSON file.

```
"schematics":
[
  {
    "path" : string,
    "height" : null | number,
    "width" : null | number,
    "resize" : null | "x" | "y" | "xy",
    "display" : "always" | "closed" | "never" | "open"
  }
]
```

To reference the root schematic write "" for the "path" keyword. If the height and width are not specified the schematic is displayed with the original dimensions of the PLECS schematic.

Keyword: "scopes"

With the "scopes" keyword the visibility of a PLECS Scope within a schematic of a PLECS WBS model can be specified.

Listing 3.3: Scopes entry in JSON file.

```
"scopes":
[
  {
    "path" : string,
    "height" : 400 | number,
    "width" : 400 | number,
    "resize" : null | "x" | "y" | "xy",
    "display" : "always" | "closed" | "never" | "open",
    "analysis":
```

```
    {
      "cursor1": { "time": null | number},
      "cursor2": { "time": null | number},
      "delta":
      {
        "locked":  false | true,
        "time": null | number
      }
      "thd": { "label": null | string}
    }
  }
]
```

Note that if the time values for cursor 1 and cursor 2 are specified together with a delta value then the time value of cursor 1 is omitted. Besides the calculation of the total harmonic distortion (THD) other functions are available using the keywords implying their respective meanings: "min", "max", "mean" and "rms".

Keyword: "results"

With the "results" keyword a two-dimensional result table can be specified. The tables are displayed below the last open scope in the WBS GUI.

Listing 3.4: Results entry in JSON file.

```
"results":
[
  {
    "title" : null | string,
    "width" : number,
    "columns":
    [
      { "header": null | string }
    ],
    "rows":
    [
      { "header": null | string, "data": [ ]}
    ]
  }
]
```

The data array is specified as follows:

Listing 3.5: Data array entry for the results section.

```
"data":
[
  {
```

```
    "scope": number,
    "plot": number,
    "signal": number,
    "analysis": "cursor1" | "cursor2",
    "unit": null | string
  }
]
```

The scope, plot and signal number count starting at zero.

Keyword: "paramtables"

The "paramtables" keyword specifies where and how a user can input a parameter value. The parameter tables can be displayed in three different ways:

- "schematic": The parameter field is displayed at the position of a component in the schematic.
- "dialog": The parameter field or combobox is displayed in a dialog box, that opens when clicking on the specified component.
- "table": The parameter table in the form of a radio button or combobox is displayed just above the simulation button.

The JSON file entries for the three possibilities are given below:

Listing 3.6: Paramtables entry with the display option "schematic".

```
"paramtables":
[
  {
    "display": "schematic",
    "component": string,
    "labelpos": "left" | "right" | "top" | "bottom",
    "description": string,
    "parameters":
    [
      {
        "variable": string,
        "label": null | string,
        "unit": null | string,
        "min": null | number,
        "max": null | number,
        "value": number,
        "disabled": false | true,
        "tracename": false | true
      }
    ]
  }
]
```

Listing 3.7: Paramtables entry with the display option "dialog".

```
"paramtables":
[
 {
  "display": "dialog",
  "component": string,
  "description": string,
  "parameters":
  [
   {
    "variable": string,
    "label": null | string,
    "unit": null | string,
    "min": null | number,
    "max": null | number,
    "value": number,
    "disabled": false | true,
    "tracename": false | true
   },
   {
    "variable": number,
    "addvariables": null | [string],
    "label": null | string,
    "type": "combobox",
    "options":
    [
     { "value": number, "addvalues": null | [number], "label": null | string }
    ],
    "unit": null | string,
    "value": null | number,
    "tracename": false | true
   }
  ]
 }
]
```

Listing 3.8: Paramtables entry with the display option "table".

```
"paramtables":
[
 {
  "display": "table",
  "parameters":
  [
   {
    "variable": string,
    "addvariables": null | [string],
    "label": null | string,
    "multiple": true | false,
    "type": "radio" | "combobox",
    "size": null | number,
    "tracename": false | true
    "options":
```

```
      [
       {
        "value": number,  "addvalues": null | [number], "label": null | string,
        "suffix": null | string, "link": null | string,
        "filter": null | [ { "variable": string, "value": number } ],
       }
      ],
      "value": null | number
     }
    ]
   }
]
```

If the keyword "tracename" takes the boolean value *true* the automatic labelling of simulation traces is enabled. This means for every simulation run the trace is labelled with a list of corresponding parameter values.

The keywords "suffix" and "link" will only take effect if the type of the parameter table is specified as "radio" (for "combobox" the entries are ignored). Also note that for the combobox and radio button it is possible to set additional variables by specifying the keyword "addvariables", which take the values "addvalues" inside the "options" array. Examples of variables that could be updated based on a user selection are passing thermal impedance information for a heat sink that is appropriate for a specific device or maximum operating conditions that can be used to flag an error message should they be exceeded during simulation.

The keyword "multiple" allows for a multiple selection of entries in a combo box or radio button. The vertical size of the combo box as a number of entries can be set with the "size" keyword (for the radio button the entry is ignored).

Keyword: "simulations"

The "simulations" keyword defines the possible simulation types (transient and steady-state analysis) for a PLECS model. The entry in the JSON file looks as follows:

Listing 3.9: Simulations entry.

```
"simulations":
[
 {
  "label": null | string,
  "analysis": null | "Steady−State Analysis"
 }
]
```

If no analysis method is specified a transient simulation with the label "Simulate" is automatically enabled.

PLECS Model

To set up a PLECS WBS example a working PLECS model with transient or steady-state analysis is needed. In this step there are several points to note:

- The model needs to be saved in the models folder together with all corresponding data files and thermal descriptions. The file names and thermal description folder name follow the pattern: *model_name.plecs*, *model_name.json* and *model_name_plecs/*
- Define variables for parameters a user is allowed to tune in a PLECS WBS session and initialize all variables in the initialization script of the PLECS model.
- Consider choosing a "stiff" solver in PLECS as a given combination of individual user parameters might create a numerically stiff system.
- Provide meaningful component names to make it easier to set up the JSON file.
- Provide meaningful trace and signal descriptions in the PLECS Scopes as the user won't be able to trace back to the origin of the individual datasets.
- Component names that should be visible in a PLECS WBS model also have to be visible in the schematic of the PLECS model.
- Leave enough space around components that have a parameter table of type "schematic" to avoid overlap of the text field and component symbols. Also, the component name and a parameter table of type "schematic" should not be placed in the same position adjacent to a component.

Note that it is possible to use the assertion framework of PLECS inside WBS models. An evaluation of an assertion may cause an abortion of the simulation or display a warning or error message at the bottom of the root schematic in the WBS GUI. Using assertions to pause and later continue the running of a simulation is not possible.

4

Deployment

In this chapter the steps are discussed to make the WBS model available for use either on a local-host or on an openly accessible web server.

Model Embedding

Direct Model Embedding

For the case where the web server application and the PLECS Standalone application run on the same physical server, the WBS model can be directly embedded into a web page. When working with a local server environment to test and develop WBS models (see section 2) the following code for a generic HTML page can be used:

Listing 4.1: Generic HTML test page for WBS.

```
<!DOCTYPE html>
<html xmlns:ng="http://angularjs.org" id="ng-app" ng-app="plecsModel">
  <head>
    <link href="../css/plecs.css" rel="stylesheet" type="text/css" />
    <!--[if lte IE 8]>
      <script src="../js/jquery.min.js"></script>
    <![endif]-->
    <script src="../js/angular.min.js"></script>
    <script src="../js/plecs.min.js"></script>
  </head>
  <body>
    <div pl-path="model_path" pl-model="model_name" pl-url-eval="boolean"></div>
  </body>
</html>
```

The code listing is divided into the following parts:

2 The AngularJS module called "plecsModel" is bootstrapped.

4 The CSS file "plecs.css" is loaded. It is needed to render all objects related to the PLECS WBS correctly and should not be changed. If you would like to change the appearance of the WBS model include your own CSS file after "plecs.css" and override selected styles.

5-7 Including jQuery is only required if your page needs to support the old version 8 of Internet Explorer.

8 The compressed version of the AngularJS script is loaded.

9 The JavaScript file for PLECS WBS is loaded.

12 The PLECS WBS model *model_name* is loaded into the web page. The model path *model_path* is specified relative to the directory *models*. If the optional attribute *pl-url-eval* is set to *true* a WBS model can be loaded with custom parameters encoded in the URL string.

Note that depending on the root path specified in the web server installation, the paths of the scripts and style sheets might need to be adjusted.

Custom Default Parameters

It is possible to load a WBS model with custom parameters that override the default values specified in the JSON file. This can be done by providing a query string after the URL address of the web page in which the WBS model is embedded. The basic syntax of the query string is given in the following:

http://*webserver/webpage.html*?*variable1*=*value1*&*variable2*=*value2*&...

This example loads an arbitrary WBS model in the web page *webpage.html* that is hosted by *webserver*. Additionally, the model is initialized with the parameters listed after a question mark (?). Each parameter is assigned a value and the different parameters are separated by an ampersand (&). In this way the default values for the parameter variables given in the JSON file can be modified by the web site visitor. Note that special characters in the query string need to be properly encoded[1]. To allow the web site visitor overriding parameter values with the URL string the attribute *pl-url-eval=true* needs to be set when embedding the WBS model in the web page (see the previous section). It is also possible to repeat the expression *variable1* = *valuex* several times with different values for *valuex* to select multiple items in a checklist or combobox.

[1]For a list of character encodings please refer to Wikipedia or http://urlencode.org to directly encode a specific string.

Indirect Model Embedding using CORS

In Fig. 1.1 the simulation provider and web site operator are not the same instance and the PLECS WBS application runs on a different server than the web server delivering the HTML content. In this case the WBS framework needs to be loaded externally from the simulation server and all simulation requests through the GUI of the WBS need to be forwarded to the simulation server. This indirect model embedding is done using *cross-origin resource sharing* (CORS).

The code listing 4.2 can be included either in the <*head*> or the <*body*> section of the HTML page. There the PLECS WBS framework is loaded and the JavaScript file *xdomain.js* is loaded to handle cross domain requests. Note that all resources are loaded with an inherited transfer protocol indicated by a double slash (//) so that the WBS model can be loaded over secured and non-secured connections depending on what the web page visitor uses to load the HTML page.

Listing 4.2: First part of the code needed to include PLECS Web-Base Simulation using CORS.

```
<link href="//demo.plexim.com/css/plecs.css" rel="stylesheet" type="text/css" />
<script src="//demo.plexim.com/js/xdomain.min.js" slave="//demo.plexim.com/proxy.html"></script>
<script src="//demo.plexim.com/js/angular.min.js"></script>
<script src="//demo.plexim.com/js/plecs.min.js"></script>
```

If the web server already includes an older version of jQuery (such as Drupal 7) or the simulation provider wants to support Internet Explorer 8 the following alternative code needs to be included:

Listing 4.3: Alternative code to listing 4.2.

```
<link href="//demo.plexim.com/css/plecs.css" rel="stylesheet" type="text/css" />
<script>
  if(window.jQuery) {
    var arr = jQuery.fn.jquery.split('.');
    if (arr[0] < 1 || (arr[0] == 1 && arr[1] < 7))
      jQueryOld = jQuery.noConflict(true);
  }
</script>
<script src="//demo.plexim.com/js/xdomain.min.js" slave="//demo.plexim.com/proxy.html"></script>
<!--[if lte IE 8]>
  <script src="//demo.plexim.com/js/jquery.min.js"></script>
<![endif]-->
<script src="//demo.plexim.com/js/angular.min.js"></script>
<script>if (window.jQueryOld) jQuery = jQueryOld;</script>
<script src="//demo.plexim.com/js/plecs.min.js"></script>
```

An example to indirectly load a WBS model is given in listing 4.4. The code needs to be placed in the <*body*> section where the model should appear within the web page. This example loads a single-phase diode rectifier with custom default user parameters enabled (*pl-url-eval* = *true*).

Listing 4.4: Example code that loads a PLECS diode rectifier model in the *<body>* section of the HTML file.

```
<div id="ng-app" ng-app="plecsModel">
  <div pl-model="diode_rect_ind_load" pl-path="plexim" pl-url-eval = true> </div>
</div>
```

Setting up the PLECS Simulation Server

For a fully-functional setup with multiple simultaneous client connections a PLECS Web Server license is required. The PLECS Web Server license allows you to run PLECS as a service (on Windows) or daemon (on OS X and Linux), i.e. it runs in the background without a graphical user interface. In this mode PLECS can be started automatically during system startup before any user is logged into the system.

To set up a PLECS simulation server first follow the steps for installing PLECS in the PLECS manual. To run PLECS as a service or daemon an additional executable, **`PLECS_server`**, is provided in the PLECS installation directory. This executable accepts the following command line options:

`-setport` ***port_number***
: Sets the TCP port that PLECS will use for XML-RPC. This port number must match the port specified in the *config.ini* file. PLECS needs to be restarted to make the change effective.

`-i`
: Installs the PLECS service (Supported on Windows only).

`-u`
: Deinstalls the PLECS service (if it has been installed with **`-i`** before, supported on Windows only).

`-t`
: Terminates PLECS and the PLECS service.

When called without arguments, PLECS is started in server mode in the background. It can be stopped by calling `PLECS_server -t`.

RC script for Linux

To start PLECS as a daemon on Linux the script below may serve as an example. It assumes that PLECS is installed in `/opt/plecs`. The script was developed for Ubuntu Linux 14.04 LTS.

Listing 4.5: Example rc script for Linux.

```
#! /bin/sh
### BEGIN INIT INFO
# Provides:          plecs
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts the PLECS daemon.
# Description:       Starts and stops the PLECS daemon.
### END INIT INFO

# Author: Oliver Schwartz <schwartz _at_ plexim.com>
#
# Please remove the "Author" lines above and replace them
# with your own name if you copy and modify this script.

# Do NOT "set -e"

# PATH should only include /usr/* if it runs after the mountnfs.sh script
PATH=/sbin:/bin:/usr/bin
DESC="PLECS server application"
NAME=PLECS_server
export LD_LIBRARY_PATH=/opt/plecs
DAEMON=/opt/plecs/$NAME
DAEMON_ARGS=""
PIDFILE=/var/run/$NAME.pid
SCRIPTNAME=/etc/init.d/$NAME

# Exit if the package is not installed
[ -x "$DAEMON" ] || exit 0

# Read configuration variable file if it is present
[ -r /etc/default/$NAME ] && . /etc/default/$NAME

# Load the VERBOSE setting and other rcS variables
. /lib/init/vars.sh

# Define LSB log_* functions.
# Depend on lsb-base (>= 3.2-14) to ensure that this file is present
# and status_of_proc is working.
. /lib/lsb/init-functions

#
# Function that starts the daemon/service
#
do_start()
```

```
{
48    # Return
      #   0 if daemon has been started
50    #   1 if daemon was already running
      #   2 if daemon could not be started
52    start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON --test > /dev/null \
          || return 1
54    start-stop-daemon --start --quiet --pidfile $PIDFILE --exec $DAEMON -- \
          $DAEMON_ARGS \
56        || return 2
}
58
#
60 # Function that stops the daemon/service
#
62 do_stop()
{
64    # Return
      #   0 if daemon has been stopped
66    #   1 if daemon was already stopped
      #   2 if daemon could not be stopped
68    #   other if a failure occurred
      pidofproc $DAEMON >/dev/null
70        [ "$?" != 0 ] && return 1
      $DAEMON -t
72        pidofproc $DAEMON >/dev/null
          [  "$?" = 0 ] && return 2
74    # Wait for children to finish too if this is a daemon that forks
      # and if the daemon is only ever run from this initscript.
76    start-stop-daemon --stop --quiet --oknodo --retry=0/30/KILL/5 --exec $DAEMON
      [ "$?" = 2 ] && return 2
78    # Many daemons don't delete their pidfiles when they exit.
      rm -f $PIDFILE
80    return 0
}
82
case "$1" in
84  start)
      [ "$VERBOSE" != no ] && log_daemon_msg "Starting $DESC" "$NAME"
86    do_start
      case "$?" in
88        0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
          2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
90    esac
      ;;
92  stop)
      [ "$VERBOSE" != no ] && log_daemon_msg "Stopping $DESC" "$NAME"
94    do_stop
      case "$?" in
96        0|1) [ "$VERBOSE" != no ] && log_end_msg 0 ;;
          2) [ "$VERBOSE" != no ] && log_end_msg 1 ;;
98    esac
      ;;
100 status)
```

```
        status_of_proc "$DAEMON" "$NAME" && exit 0 || exit $?
102     ;;
      restart|force-reload)
104     #
        log_daemon_msg "Restarting $DESC" "$NAME"
106     do_stop
        case "$?" in
108       0|1)
          do_start
110       case "$?" in
              0) log_end_msg 0 ;;
112           1) log_end_msg 1 ;; # Old process is still running
              *) log_end_msg 1 ;; # Failed to start
114       esac
          ;;
116     *)
          # Failed to stop
118       log_end_msg 1
          ;;
120     esac
        ;;
122   *)
        echo "Usage: $SCRIPTNAME {start|stop|status|restart|force-reload}" >&2
124     exit 3
        ;;
126   esac

128   :
```

plexim

electrical engineering software

Plexim GmbH info@plexim.com www.plexim.com