***eis system***

*didactic SmartHome electrical simulator*

**How to write a new device**

*upf, 2013-07-05, ver. 1.1*

1. **Preliminary operations**

After the successful installation of the ***eis*** system on the machine, go to the ***eis*** main directory (the same shown in the *$eis\_conf[“path”]* variable inside the *eis\_conf.php* file). Then copy the folder “*eis.device\_template*” in the “*system*” directory to the “*devices*” directory renaming it in accordance with the ***eis*** system requirements:

*device\_brand\_name****.****device\_model\_name****.****device\_serial\_string*

The 3 fields can contain only alphanumeric and “\_” characters, separated by “.” character. The first 2 are mandatory while the third is optional (examples: *saturn****.****wash\_and\_dry****.****02B34598* or *eis.meteo*).

Finally move to the newly created directory (your device).

1. **Configure your device**

Edit the “*device\_conf.php*” file in the “private” folder, which contains the definition of the associative array *$eis\_dev\_conf* storing all the specific configuration parameters of this device. Change the contents of the following indexes according to your needs:

* ***version***  
  the version of the device implementation as *version.subversion.release* (string)
* ***date***  
  the date of the device implementation release (string)
* ***author***  
  the name of the device implementation author(s) (string)
* ***ifport***  
  the number (integer) of the UDP port used by the HTML realtime interface (see after). Must be different for each device installed in this host (see device classes document).
* ***class***  
  the class ID (string) which this device belongs to. Recognized class IDs are listed in another specific document.
* ***description***a short description (string) of this device
* ***type***the type ID (string) which this device belongs to. Recognized types are:   
   ***load*** a device that only consumes electrical energy   
   ***generator*** a device that only generates electrical energy   
   ***load&gen*** a device that can consume and generate electrical energy   
   ***virtual*** a virtual device that has no electrical energy (e.g. managers, meteo, etc.)

To understand the other indexes, it is important to remember the electrical model of the ***eis*** simulator:

grid interface

storage

unprotected line

protected line

gen

load

gen

load

grid

When the “*grid interface*” is installed, the simulator will simulate a grid-connected system where the electrical energy can be purchased from or sold to the public grid (at the current prices). If the “*protected line*” is active, an electrical storage system must be installed. In this case a grid-connected system with storage will be simulated.

When the “*grid interface*” is not installed, the simulator will simulate an off-grid system with electrical storage. In this case no loads are connected to the “*unprotected line*” which is used to connect an auxiliary generator (usually a diesel generator).

Following this scheme, the device must be configured for its consumption parameters and/or generation parameters.

***Consumption parameters*** (only for device type “***load***” or “***load&gen***”):

* ***cpower1, cpower2, cpower3***the max electrical power (integer) in watts that the device can consume on a specific electrical phase (1, 2 or 3). Set a value to zero if the corresponding phase is not connected (e.g. a 1-phase device will have 2 of 3 values set to zero).

***Generation parameters*** (only for device type “***generator***” or “***load&gen***”):

* ***gpower1, gpower2, gpower3***the max electrical power (integer) in watts that the device can generate on a specific electrical phase (1, 2 or 3). Set a value to zero if the corresponding phase is not connected (e.g. a 1-phase device will have 2 of 3 values set to zero).

***Configuration parameter***:

* ***configurations***an array containing couple of “configID” => configuration parameter array. For single configuration, use ***default*** ID.

Single or multiple configurations can be handled by the system.

In case of single configuration use this example code:

$eis\_dev\_conf["configurations"]=array(

// default configuration

"default" => array (

"config"=>"default configuration",

"cpower1"=>2000,

"cpower2"=>2000,

"cpower3"=>2000

// put here other vars if needed

)

);

In case of multiple configurations, see this example code:

$eis\_dev\_conf["configurations"]=array(

// small 3kW 1 phase connection with ITA constant price plan

"ESE\_3kW\_1P\_1price" => array (

"config"=>"3kW 1P ITA Enel Servizi Elettrici tariffa monoraria",

"gpower1"=>3000,

"gpower2"=>0,

"gpower3"=>0,

"cpower1"=>3000,

"cpower2"=>0,

"cpower3"=>0,

"price"=>"ESE-1price-small"

),

// small 3kW 1 phase connection with ITA 2-prices plan

"ESE\_3kW\_1P\_2prices" => array (

"config"=>"3kW 1P ITA Enel Servizi Elettrici tariffa bioraria",

"gpower1"=>3000,

"gpower2"=>0,

"gpower3"=>0,

"cpower1"=>3000,

"cpower2"=>0,

"cpower3"=>0,

"price"=>"ESE-2prices-small"

),

// ……….other configuration here

)

and set the status variable ***configID*** to the ID of your desired default configuration, e.g.

$eis\_dev\_conf["status"]=array(

// set default configuration

"configID" => "ESE\_3kW\_1P\_1price",

1. **Configure your device status variables**

In the ***eis*** system, devices interact using HTTP calls. Therefore the device code is executed each time a call is made by another device or an HMI, this means that all the contained variables will be re-initialized at each execution if not explicitly saved at the end of a call and loaded at its beginning.

For this reason the ***eis*** system implements a built-in “status” mechanism, which allows having persistent variables, i.e. variables that maintain their values from a call to another. Such variables need also to have known (default) values to be assumed when the device is initialized.

To declare a status variable and set its initial value, the *$eis\_dev\_conf["status"]* associative array is available into the “*device\_conf.php*” file. If the variable “*mystatusvar*” with initial value “37” has to be declared, simply add the following index to the *$eis\_dev\_conf["status"]* array:

*“mystatusvar” => 37*

**System-managed status variables:**

The following status variables are always required and managed by the system itself (must not explicitly declared into the *device\_conf.php* file):

* ***timestamp***holds the current UNIX timestamp of the current simulation (default 0)
* ***masterurl***holds the URL of the current simulation master device
* ***enabled***holds the enable/disable device status
* ***power (boolean)***holds the power-on/power-off device status
* ***sim\_id***holds the current simulation ID (an unique string that identifies the simulation)
* ***sim\_step***holds the current simulation time step in minutes (time between 2 simulation steps)
* ***sim\_type***holds the current simulation type (can be only “*off-grid*” or “*grid-connected*”).

**Required status variables:**

The following status variables are always required but must be managed by your code:

* ***cline (only for “load” or “load&gen”)***the electrical line which this device (as load) is connected to (recognized types are: ***unprotected*** or ***protected***).
* ***cpower1, cpower2, cpower3 (integer) (only for “load” or “load&gen”)***hold the current consumed electrical power (watt) on phase 1,2 or 3
* ***cenergy1, cenergy2, cenergy3 (float) (only for “load” or “load&gen”)***hold the consumed electrical energy (kWh) on phase 1,2 or 3 from the beginning of current simulation.
* ***gline (only for “generator” or “load&gen”)***the electrical line which this device (as generator) is connected to (recognized types are: ***unprotected*** or ***protected***)
* ***gpower1, gpower2, gpower3 (integer) (only for “generator” or “load&gen”)***hold the current generated electrical power (watt) on phase 1,2 or 3
* ***genergy1, genergy2, genergy3 (float) (only for “generator” or “load&gen”)***hold the generated electrical energy (kWh) on phase 1,2 or 3 from the beginning of current simulation

**Other private status variables:**

Other status variables can be freely added to the *$eis\_dev\_conf["status"]* array. They will have to be completely managed by your code.

1. **Predefined commands and signals**

In the ***eis*** system, your new device is now ready to accept commands and signals and to answer to them. Some commands and signals are predefined and already implemented without any additional coding.

**Main predefined commands:**

* ***ping***  
  *returns the call input parameters as return parameters   
  input parameters : any   
  output parameters : the input parameters   
  errors : none*
* ***delay***  
  *delay the answer of “duration” seconds (default 10)   
  input parameters : duration (integer, optional)   
  output parameters : none   
  errors : none*
* ***getlog***  
  *returns the last “numrow” lines (default 10) from the device log   
  input parameters : numrow (integer, optional)   
  output parameters : getlog (array of log lines)   
  errors : none*
* ***getstatus***  
  *returns the value(s) of status variable(s). With no input parameters returns all the device status variables. If the “fields” parameter is present, returns only the status variables specified in it as a list of comma separated names   
  input parameters : fields (string, optional) example: varname1,varname2   
  output parameters : all the requested variables as name=>value   
  errors : none*
* ***setstatus***  
  *sets the value(s) of status variable(s). The variables to set are given as input parameters (name=>value)   
  input parameters : all the requested variables as name=>value   
  output parameters : none   
  errors : none*
* ***getconfig***  
  *returns the $eis\_dev\_conf array as output parameters. If the “status” input parameter is present and equal to “current” the $eis\_dev\_conf[“status”] array will contain the actual status values instead that the predefined ones.   
  input parameters : status (string, optional) sets the returning status behaviour  
  output parameters : $eis\_dev\_conf array as name=>value   
  errors : none*
* ***help***  
  *returns an help string (if any) as output parameter   
  input parameters : none   
  output parameters : “help” field containing the help string   
  errors : none*
* ***reset***  
  *reset the device status to its initial values (stored into device\_conf.php file) and returns the whole $eis\_dev\_conf array as output parameter   
  input parameters : none   
  output parameters : $eis\_dev\_conf array as name=>value   
  errors : none*
* ***install***  
  *reset the device status to its initial values (stored into device\_conf.php file), executes any installation code and returns the device status array as output parameter   
  input parameters : none   
  output parameters : status array as name=>value   
  errors : none*
* ***init***  
  *initializes a new simulation described by some parameters passed as input (timestamp, sim\_id, sim\_step, sim\_type, cline, gline). The device status is reset to its initial value (stored in $eis\_dev\_conf[“status”] array) and enabled. The input parameter values are stored into the corresponding status variables, e.g. the “timestamp” status variable will hold the starting UNIX time of the simulation. The “masterurl” status variable is set to the URL of the device issuing the call (that will be assumed as master). This command returns the current status array as output parameters.   
  input parameters : timestamp (integer, mandatory)   
  output parameters : status array as name=>value, with set “status” fields  
  errors : system:parameterMissing (a required input par is missing)*
* ***simulate***  
  *executes the simulation step at the time indicated by the “timestamp” input parameter. For the loads and generators the “blackout” Boolean input parameter power off the devices when set to true and power on them when reset to false. Other input parameters depends on the class or type of device, as follows:*

|  |  |  |
| --- | --- | --- |
| **class or type** | **Input parameters** | **required status vars** |
| meteo\_station | timestamp | temperature (C), humidity (%), windspeed (m/s), winddir (deg), pressure (mbar), radiation (w/m2) |
| any load | timestamp, meteo array (6 fields as in meteo\_station required status vars), blackout (true/false) | cpower1 (W), cpower2, cpower3, cenergy1 (kWh), cenergy2, cenergy3 |
| any generator | blackout (true/false), timestamp, meteo array (as before), cpower array (1=>cpower on phase 1, 2=> on phase 2, 3=> on phase 3) | gpower1 (W), gpower2, gpower3, genergy1 (kWh), genergy2, genergy3 |
| electrical\_storage | timestamp, meteo array (as before), cpower array (as before), gpower array (1=>gpower on phase 1, 2=> on phase 2, 3=> on phase 3) | cpower array (1=>cpower on phase 1, 2=> on phase 2, 3=> on phase 3) + other to be defined |
| grid | timestamp, meteo array (as before), cpower array (as before), gpower array (as before) | cpower1 (W), cpower2, cpower3, cenergy1 (kWh), cenergy2, cenergy3, gpower1 (W), gpower2, gpower3, genergy1 (kWh), genergy2, genergy3, price\_sell, price\_buy, tpower (total power) tenergy (total energy), total\_sell, total\_buy, total\_money, gridstatus (ok, disconnected, overload, overgen) |
| auxiliary\_generator | timestamp, meteo array (as before), cpower array (as before), gpower array (as before) | to be defined |
| energy\_manager | to be defined | to be defined |
| user\_model | to be defined | to be defined |
|  |  |  |

*This command returns the current status array as output parameters.   
input parameters : timestamp (integer, mandatory) + others   
output parameters : status array as name=>value, with set “status” fields  
errors : system:parameterMissing (a required input par is missing)*

**Main predefined signals:**

* ***enable***  
  *set the internal status $eis\_dev\_status["enables"] to true.* *This means that the device will respond to any implemented command or signal (normal operation)   
  input parameters : none*
* ***disable***  
  *set the internal status $eis\_dev\_status["enables"] to false.* *This means that the device will not respond to any implemented command or signal (it is not available for operations) except for the “enable” signal and the “init”,“reset” and “install” commands. These commands will set also the internal status to “enabled” again.  
  input parameters : none*
* ***poweron***  
  *set the $eis\_dev\_status["power"] to true* *and execute user-defined code for powering on the device.   
  input parameters : none*
* ***poweroff***  
  *set the $eis\_dev\_status["power"] to false* *and execute user-defined code for powering off the device.   
  input parameters : none.*

1. **How to write your code**

The new device MUST implement at least 3 commands and 2 signals. Edit the file “*device.php*” in the “private” folder which will contain your actual implementation, remembering that 4 global variables are always available:

* ***$eis\_conf***  
  an associative array as defined in “*eis\_conf.php*” (global configuration);
* ***$eis\_dev\_conf***  
  an associative array as defined in your “*device\_conf.php*” (device configuration);
* ***$eis\_dev\_status***  
  an associative array as defined in *$eis\_dev\_conf[“status”]* (plus the predefined status variables) containing the device current status;
* ***$calldata***  
  an associative array describing the current call. Useful indexes are:   
   ***from*** contains the URL of the caller   
   ***type*** call type (exec, dexec or signal)   
   ***cmd*** the command/signal name   
   ***param*** the call input parameters as an associative array

Inside this file, 7 functions are already defined:

* ***eis\_device\_install($callparam)***  
  put in it your device installation code (if any). This code will be executed when the command “***install***” is sent to the device. These additional data are available for coding:
  + *$callparam* an associative array holding the call input parameters,
  + *$eis\_mysqli* the mysqli handle for eis database. If you need to create and use private tables, you must use *$eis\_dev\_conf["tablepfx"]* as table name prefix.
* ***eis\_device\_init($callparam)***  
  put in it your device initialization code (if any). This code will be executed when the command “***init***” is sent to the device.
* ***eis\_device\_simulate($callparam)***  
  put in it your device simulation code. This code will be executed when the command “***simulate***” is sent to the device. The “timestamp” update is already done by the system.
* ***eis\_device\_poweron()***  
  put in it your device power on code, e.g. set all the powers to some non zero values. This code will be executed when the signal “***poweron***” is sent to the device. The power state update is already done by the system.
* ***eis\_device\_poweroff()***  
  put in it your device power off code, e.g. set all the powers to zero. This code will be executed when the signal “***poweroff***” is sent to the device. The power state update is already done by the system.
* ***eis\_device\_exec($calldata****)*  
  put in it the code implementing your device commands. A skeleton is already given and should be used, each case of the switch corresponds to a command implementation.
* ***eis\_device\_signal($calldata)***  
  put in it the code implementing your device signals. A skeleton is already given and should be used, each case of the switch corresponds to a signal implementation. The signals “***poweron***” and “***poweroff***” MUST always be implemented.

In case of error inside *eis\_device\_install()*, *eis\_device\_init()*, *eis\_device\_simulate()*, *eis\_device\_poweron()* or *eis\_device\_poweroff()* return an appropriate message using the following code:

***return eis\_error(****"your\_brand.your\_model:your\_code ","your\_error\_message"****);***

where “*your\_brand.your\_model” should be taken from your deviceID, “your\_code*” is your specific error code and “*your\_error\_message*” is a short message describing the error.

In case of error inside *eis\_device\_exec()* or *eis\_device\_signal()*, return an appropriate message using the following code:

***return eis\_error\_msg(****"your\_brand.your\_model:your\_code ","your\_error\_message"****);***

with “*your\_brand.your\_model”* and“*your\_error\_message*” as before.

**How to write the “simulate” command:**

This required command implements a single step of the electrical simulation. The main goal of the step is:

* update the current energy values in kWh (cenergy1, cenergy2, cenergy3 and/or genergy1, genergy2, genergy3 fields of the ***$eis\_dev\_status***) using the corresponding electrical powers in watt (cpower1, cpower2, cpower3 and/or gpower1, gpower2, gpower3) and the simulation timestep in seconds *($eis\_dev\_status["sim\_step"]\*60*).
* update the electrical power consumed/generated by the device at the new timestamp (cpower1, cpower2, cpower3 and/or gpower1, gpower2, gpower3 fields of the ***$eis\_dev\_status***), considering also its power status (on or off as true or false in ***$eis\_dev\_status[“power”]*** and any other required input parameters. In order to make an input parameter (e.g. “your\_parameter”) mandatory, add the following code:   
   *if (!array\_key\_exists("your\_parameter",$callparam))   
   return eis\_error(“system:parameterMissing","your\_parameter");*

1. **Testing the device**

The new device can be tested by sending to it commands and/or signals using the console available inside the eis system page (http://localhost/eis).

1. **Writing the device help**

It is important to write a device help, which will be available through the “***help***” command or through the web page “*help.php*” in the device folder. To do this, edit the file “*help.txt*” in the “private” folder. This is a standard ASCII file written using the following simple sintax:

* lines beginning with a "**#**" char are *comments* and will be escaped
* to start a section description use {\*\* your\_section\_name \*\*}
* to start a command description use [\*\* your\_command\_name \*\*]
* to start a signal description use (\*\* you\_signal\_name \*\*)

At least the following section should be present: {\*\*Description\*\*}, {\*\*Commands\*\*}, {\*\*Signals\*\*}. Follow the skeleton for details.

1. **Writing a device interface**

The device interface is contained in the *index.php* file inside the device folder, so that it can be accessed as *http://<yourhost>/eis/deviceID*. To start writing a device interface, edit this file and follow the instruction inside it. It already contains a working interface for both generators and loads with respect to the powers, energies, enable and power status. Delete the parts that are not suitable for your device and add other features if needed.

Two main functionalities are already implemented:

* The capability to call back the page using Javascript code using the predefined JS function *eis\_callback(name, value)*. This function calls back the page using the parameter *‘name’* with value *‘value’*. From the PHP side use the predefined function eis\_callback($name) to check if a callback with parameter *$name* is active. The corresponding value can be found into *$REQUEST[$name]*. See the enable/disable buttons for a practical example.
* The capability to modify the page in real time when any change to the device status array is made. This can be implemented by adding the call to the predefined PHP function *eis\_realtime\_handler()* and by writing the Javascript function *eis\_updatepage(status)*. This JS function receives the input parameter *‘status’* as an associative array (name=>value) containing the changed $eis\_dev\_status array fields and their values. These can be used to modify the page in real time, see the “enabled” case for a practical example.

Many different ready-to-use graphic widgets are available in the ***eis\_interface\_lib.php*** library inside the ***system*** directory. See comments for explanation.