Device Mask options and programming

Jean Mahseredjian, 2017-08-14 13:48:00

1 Introduction

The user can create a mask after creating a subcircuit. An example of top circuit with a subcircuit is shown in Figure 1. After the creation of the subcircuit the user can right-click on the subcircuit and select "Subcircuit Info". This is where a subcircuit (subnetwork) mask can be specified. The selection "Use the default properties script" selects the "script_black_box.dwj" script. Other readily available scripts can be selected from the available dropdown menu:

❑ black_box.dwj:	basic mask with parameters and values, no calculations can be made
❑ no_script_box.dwj	does not allow entering data, only documentation information can be created

These scripts are described in their respective sections presented below.

Masking is part of open-architecture options in EMTP-EMTPWorks. It can be used at the basic level or with more advanced options. Advanced options may require writing codes (programming). The programming rules must be followed by the user or simulation problems or complicated error messages can occur otherwise.

The user must read this document before developing masks in EMTPWorks. All demonstration designs cited in this document are available in the program folder Examples\ShowHow\MaskOptions of EMTPWorks.

Tooltips are available for all Mask options as in all EMTPWorks devices.

2 The black_box.dwj properties script

2.1 Introduction

The “black_box.dwj” properties script is the most simple mask that can be created and used for a subcircuit. It can be selected using the dropdown menu shown in Figure 2.

The “black_box.dwj” script cannot be modified by the user since it is used by several built-in devices of EMTPWorks, but the user can program similar scripts (masks) by copying this code and related files.

All entered data is saved into the FormData attribute of the masked device. The actual data transmitted to the EMTP Netlist is saved into the ModelData attribute.

After masking the subcircuit Fault_device shown in Figure 1, double-clicking on the device will not open its subcircuit, but its mask. If the user wants to position into the subcircuit contents then the required command is “Options>Subcircuit>Push Into” or use Alt+double-click. The contents of the mask are initially blanked. This is shown in Figure 3.

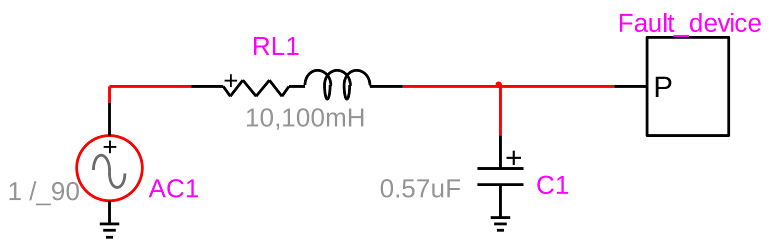
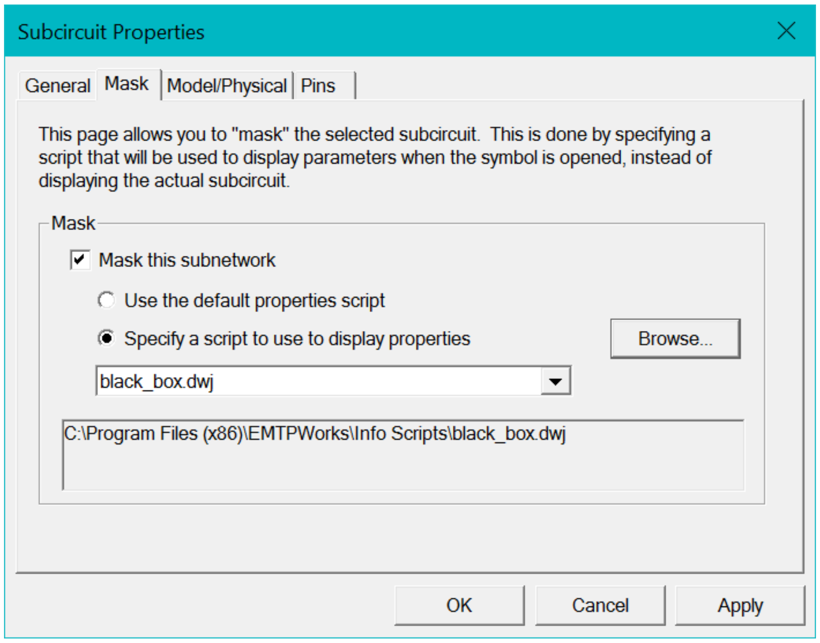


Figure 1 A top circuit with a subcircuit (see black_demo.ecf)


Subcircuit Properties

General | Mask | Model/Physical | Pins

This page allows you to "mask" the selected subcircuit. This is done by specifying a script that will be used to display parameters when the symbol is opened, instead of displaying the actual subcircuit.

Mask

☑ Mask this subnetwork

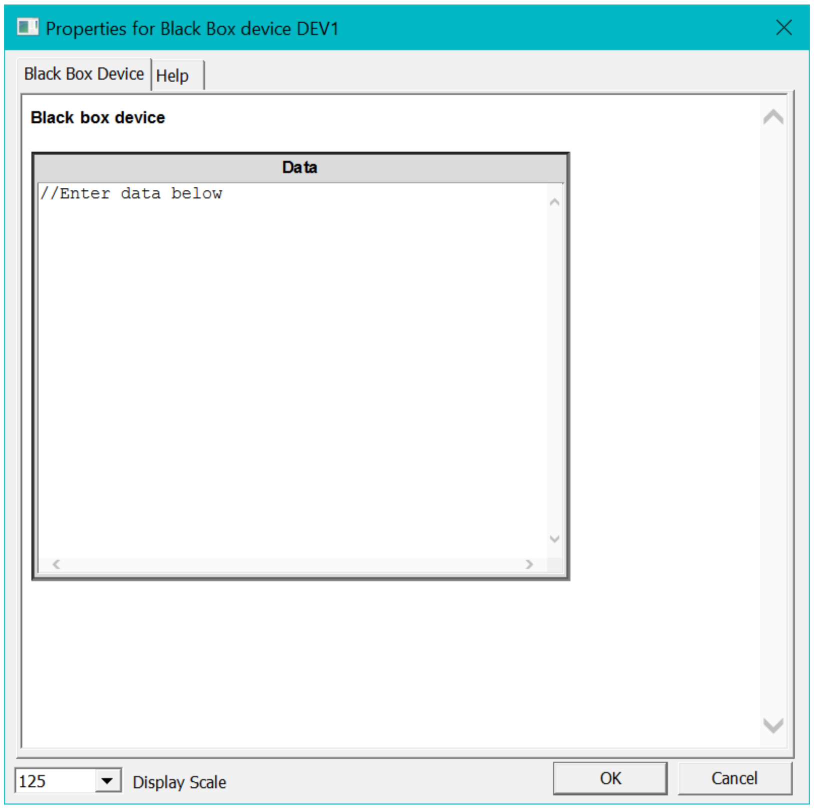
○ Use the default properties script

● Specify a script to use to display properties

black_box.dwj

C:\Program Files (x86)\EMTPWorks\Info Scripts\black_box.dwj

Figure 2 The subcircuit mask selection menu



//Enter data below

Figure 3 Initial startup of the black box mask

2.2 Available options

2.2.1 Syntax

The Data text area of the black box device can be used to enter simple expressions with equality sign. The generic format is given by:
Variable_Name = *Value*;

The *Variable_Name* field can contain a maximum of 30 alphanumeric (A-Za-z0-9_) characters including the underscore. The first character cannot be a number. It can be used as a named value (programmed value or variable) by devices located in the masked subnetwork. It is also visible in subnetworks contained in the masked subnetwork and at any level. A named value is a string enclosed between two '#' characters and entered in device data fields instead of entering numeric values. Example:

```
#Rfault#
```

This is a data programming method.

The number of characters before the equality sign is arbitrary.
The equality sign is mandatory.
The number of characters after the equality sign is arbitrary.

The *Value* field is limited to 30 characters. The Value field can be a number, a number followed by units, a string or a named value. When it is a named value it follows the same rules as the *Variable_Name* since it can refer only to an existing *Variable_Name*.

The *Variable_Name*, the equality sign and the *Value* field must appear on the same line.

The *Value* field can be also a vector. The vector is started using a left square bracket '[' and terminated with a right bracket ']'. A vector can be entered using more than one line. The contents of the vector are other *Value* fields. When a *Value* field is a vector, its target named value can use index parameters, such as, for example:

```
#Myvector(k)#
```

or

```
#Myvector(k:m)#
```

where k and m must be also defined in a black box script.

The *Value* field must be terminated with the semicolon character ';'.

Comments must start with '//'. Comments are single-line.

It is not allowed to enter more than 100 characters per data line.

2.2.2 Recognized units

The currently recognized units are those that are used by the target data fields. Default (no scaling factor) units are not entered. The Greek character μ is entered as: u.

2.2.3 Specific attributes

Subcircuit device attributes are used to set extra parameters for the black box. Attributes can be selected and modified by right-clicking on the subcircuit symbol and selecting "Attributes". This is an advanced option, there is limited error checking and careless usage can cause bugs and corrupted data.

2.2.3.1DeviceDoc attribute

The DeviceDoc attribute is used to provide a web page help location for the created device. This replaces the default Help tab of the black box mask.

The syntax is:
Root_location,*html_file*
The *Root_location* is the top directory where the *html_file* can be found. The "black_box.dwj" script appends the *Root_location* to the script path.
In this example:

```
controls,generator\help.htm
```

the full path of the help file is found under "C:\Program Files\EMTPWorks\Info Scripts\controls\generator", since the script "black_box.dwj" is located in the directory "Info Scripts".
If the *Root_location* is omitted, then the user can enter the full path of the html page:

```
,C:\Program Files\EMTPWorks\Info Scripts\controls\generator\help.htm
```

2.2.3.2HideSBBRules attribute

This attribute is used to disable the Data text area shown in Figure 3 and to provide a title instead of the default title "Black Box device".
The syntax is:
Hide_rule,*my_title*
When the *Hide_rule* is set to 0, then the Data text area is disabled. This is useful for blocking the mask to other users and thus avoiding data input errors.
The *my_title* is any title to be shown.

2.2.4 Debugging

In addition to manual testing and corrections for error messages, if the user realizes that some data is still not correctly substituted, then the only remaining option is to request EMTP to provide the final Netlist it is decoding.

The final Netlist includes all variable substitutions and can be obtained by selecting “Simulate>Advanced>Simulation Options” and then “Dump decoded data” from the Output tab.

2.2.5 Unmasking

When the user decides to unmask by going back to the “Subcircuit Properties” (Figure 2) and deselecting the mask, it is very important to remember that the ModelData attribute is not erased and is still transmitted into the Netlist. The user can erase the ModelData attribute contents through the device “Attributes” right-click menu. There is also a specific message sent to the user requesting to confirm the automatic deletion of ModelData contents. If the previously existing mask is not erased it may cause conflicts in the simulation and modify simulation results.

2.3 Examples

If it is desired, for example, to control from the mask the fault occurrence times and the fault resistance, then the following data can be entered in Figure 3 (see Figure 4):

```
//Enter data below
//Switch times:
Fault_on =1ms;
Fault_off=5ms;
//Fault resistance:
Rfault = 100;
//Scopes in the subnetwork Fault_device
Scope_of_switch='?v,?i';
//Turn on the Rextra
PartR_on_or_off='RLC';
//define fault inductance
L_fault=100mH;
```

The subcircuit is shown in Figure 5.

- The switch SWFault and the resistance Fault_resistance are given named values (programmed values or variables) to receive the variables defined in the mask.
- The device data web of SWFault is shown in Figure 8. The variable Rfault is transmitted to both Fault_resistance and Rfault2.
- The switch is using a programmable scope modified at the mask level using the Scope_of_switch variable. More information on programmable scopes can be found in the help section of the Attributes tab.
- The resistance Rextra located in the subcircuit DEVF shown in Figure 6, is optional and can be disconnected through its part name. This is achieved by entering the name x_PartR_on_or_off into the device Part attribute. When the variable PartR_on_or_off is given the original part name of the device, the device becomes active. To disable (delete) the device, it is needed to make PartR_on_or_off equal to the string ‘Exclude’.

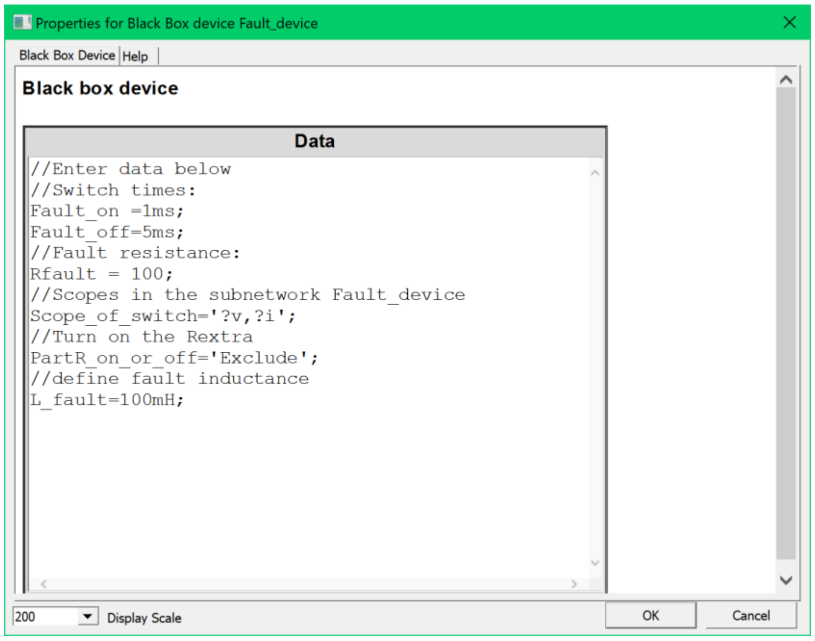
The subcircuit DEVF has its own mask (see Figure 7) and its data is:

```
//Enter data below
Rextra=Rfault;
```

This mask demonstrates that a mask in a subcircuit can use and transmit variables defined at a higher level. If the data above is modified to include L_fault:

```
//Enter data below
Rextra=Rfault;
L_fault=11.45e-3;
```

Then, although L_fault has been already defined at a higher level, it is the definition from the mask closest to the device that takes precedence.



```
//Enter data below
//Switch times:
Fault_on =1ms;
Fault_off=5ms;
//Fault resistance:
Rfault = 100;
//Scopes in the subnetwork Fault_device
Scope_of_switch='?v,?i';
//Turn on the Rextra
PartR_on_or_off='Exclude';
//define fault inductance
L_fault=100mH;
```

Figure 4 Mask of subcircuit Fault_device in Figure 1

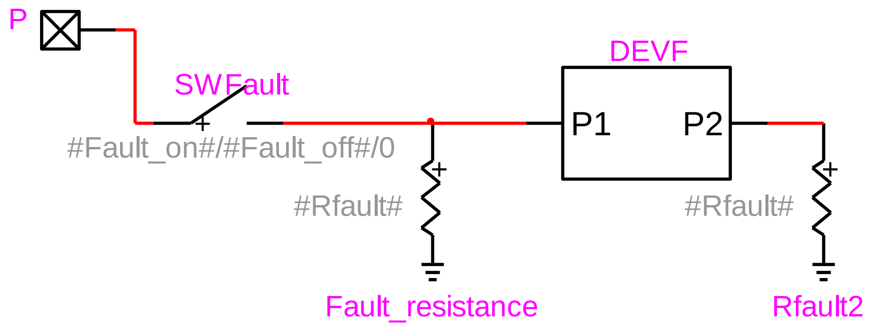


Figure 5 Subcircuit Fault_device

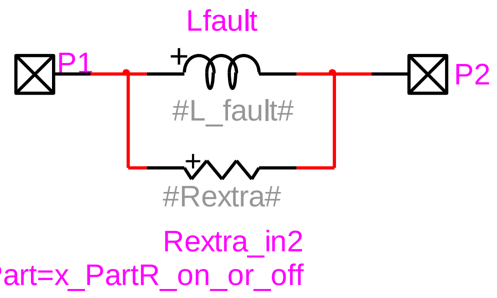
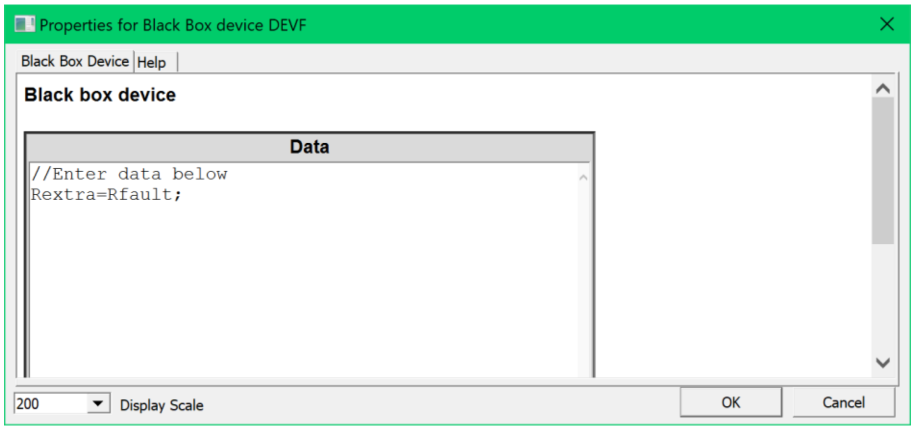
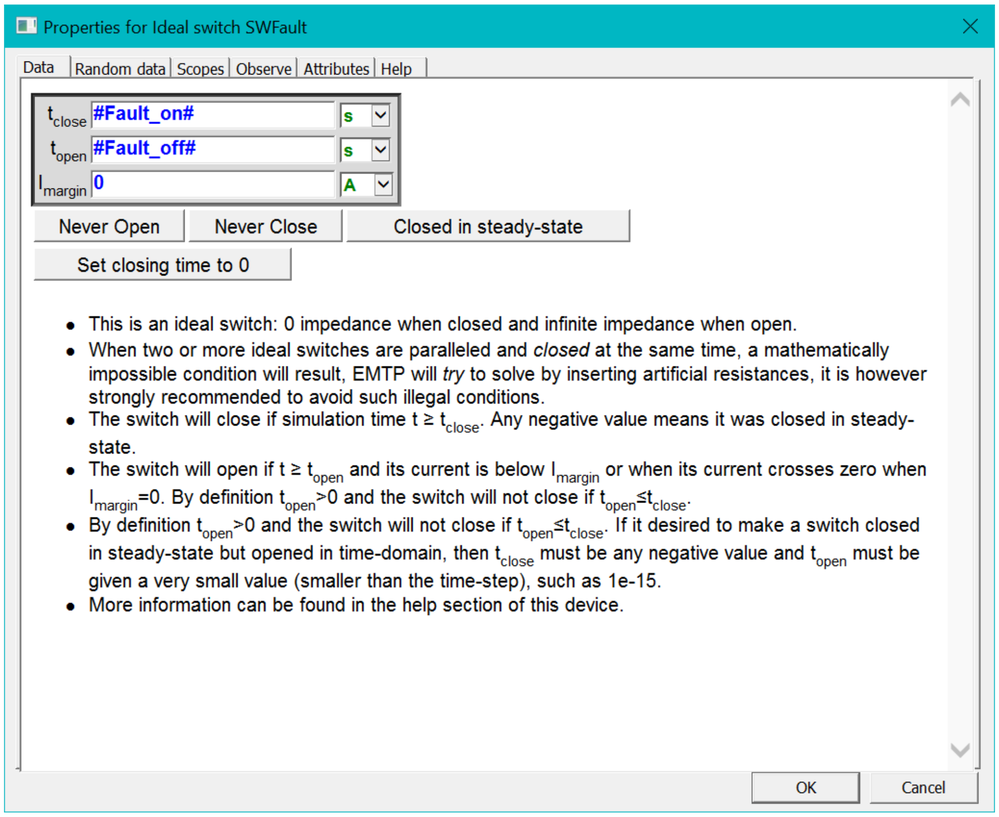


Figure 6 Subcircuit DEVF



```
//Enter data below
Rextra=Rfault;
```

Figure 7 Mask of DEVF shown in Figure 6



t_{close} #Fault_on# s

t_{open} #Fault_off# s

I_{margin} 0 A

- This is an ideal switch: 0 impedance when closed and infinite impedance when open.
- When two or more ideal switches are paralleled and *closed* at the same time, a mathematically impossible condition will result, EMTP will *try* to solve by inserting artificial resistances, it is however strongly recommended to avoid such illegal conditions.
- The switch will close if simulation time $t \geq t_{close}$. Any negative value means it was closed in steady-state.
- The switch will open if $t \geq t_{open}$ and its current is below I_{margin} or when its current crosses zero when I_{margin}=0. By definition t_{open}>0 and the switch will not close if $t_{open} \leq t_{close}$.
- By definition t_{open}>0 and the switch will not close if $t_{open} \leq t_{close}$. If it desired to make a switch closed in steady-state but opened in time-domain, then t_{close} must be any negative value and t_{open} must be given a very small value (smaller than the time-step), such as 1e-15.
- More information can be found in the help section of this device.

Figure 8 SWFault data for Figure 5

An example of vector usage is given by the black box data of the transformer device shown in Figure 9:

```
R1=1.674421875;
L1=0.17766167881829983;
Rm=248062.5;
PhiLnonl=[682.2347010794667 733.4023036604267 784.5699062413867 818.68164129536 839.148682327744
1173.4436858566828];
ILnonl=[1.0368210551463188 5.184105275731594 12.960263189328985 25.92052637865797 51.84105275731594
1036.821055146319];
Lnonl='Lnonl';
Rmag='RLC';
R2=0.026999999999999996;
```

```
L2=0.0028647889756541152;
Phiss01=0;
Phiss02=0;
Phiss03=0;
Ratio=0.38095238095238093;
Lmag_scope='?i,?f';
W1_scope  ='';
W2_scope  ='';
```

The subcircuit contents are shown in Figure 10.
The vectors PhiLnonl and ILnonl are captured by a nonlinear inductance connected in the following level subcircuit xfmr_YY_unit.

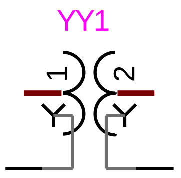


Figure 9 Masked transformer with modified subcircuit symbol (see black_demo.ecf)

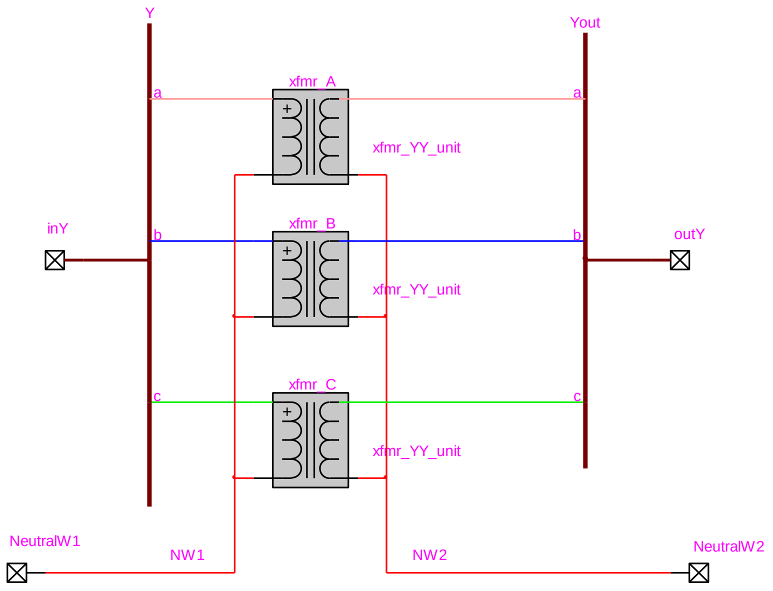
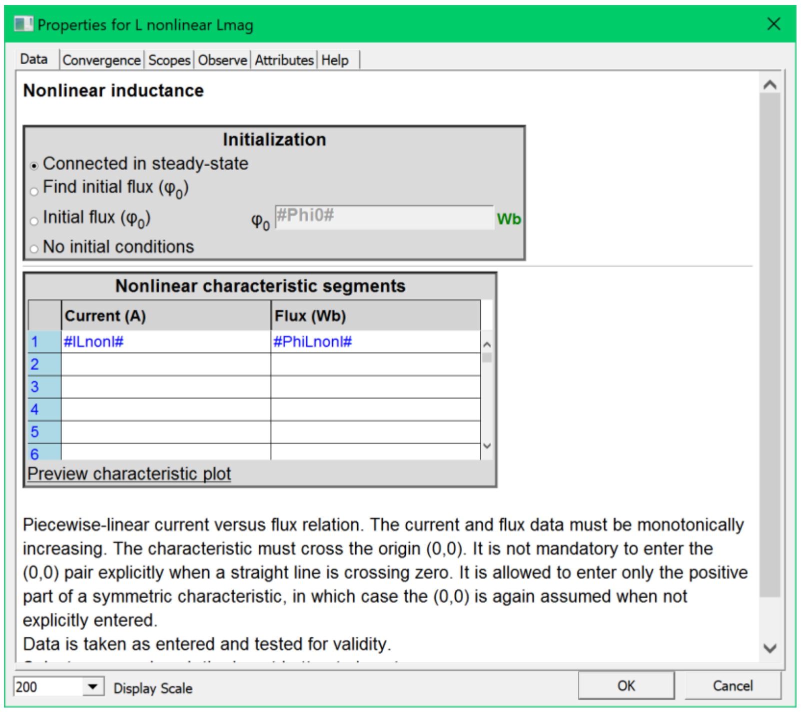


Figure 10 Subcircuit for the transformer of Figure 9


Properties for L nonlinear Lmag

Data | Convergence | Scopes | Observe | Attributes | Help

Nonlinear inductance

Initialization

[✓ Connected in steady-state]
Find initial flux (φ_0)
Initial flux (φ_0) φ_0 #Phi0# Wb
No initial conditions

Nonlinear characteristic segments

	Current (A)	Flux (Wb)
1	#ILnonl#	#PhiLnonl#
2		
3		
4		
5		
6		

Preview characteristic plot

Piecewise-linear current versus flux relation. The current and flux data must be monotonically increasing. The characteristic must cross the origin (0,0). It is not mandatory to enter the (0,0) pair explicitly when a straight line is crossing zero. It is allowed to enter only the positive part of a symmetric characteristic, in which case the (0,0) is again assumed when not explicitly entered.
Data is taken as entered and tested for validity.

200 Display Scale OK Cancel

Figure 11 Nonlinear inductance located in the subnetwork xfmr_YY_unit of Figure 10

3 The no_script_box.dwj properties script

This script is provided only for creating a help panel. It can use only the "DeviceDoc" attribute explained above.

4 The script_black_box.dwj properties script

4.1 Introduction

This script is based on JavaScript programming. It can be selected from the EMTPWorks "Info Scripts" directory using the Browse button in Figure 2 or by simply selecting the initial default script.

The objective is similar to "black_box.dwj". It is to transmit data through the ModelData attribute.
The data transmitted to the EMTP Netlist is saved into the ModelData attribute.

The advantage in this mask is that in addition to entering data in the "Initial Parameters/Values" section (see Figure 12), it is allowed to provide rules for transforming primitive (initial) data into actual circuit parameter values using the "Rules/Calculations" text-area. The "Rules/Calculations" section is programmed using JavaScript. Extensive coding can be applied using JavaScript and calling various functions and services. It is possible to call external codes saved in files. The rules for named values that are the receptacles of calculated data, are the same as in section 2.2.1 for the black box device.

The last section of this mask is for determining the parameters that must be transmitted to the contents of the masked subcircuit.

The mask (all sections) is executed when the OK button is clicked.

The devices based on this type of mask are called scripted-mask-subcircuit devices.

4.2 Debugging

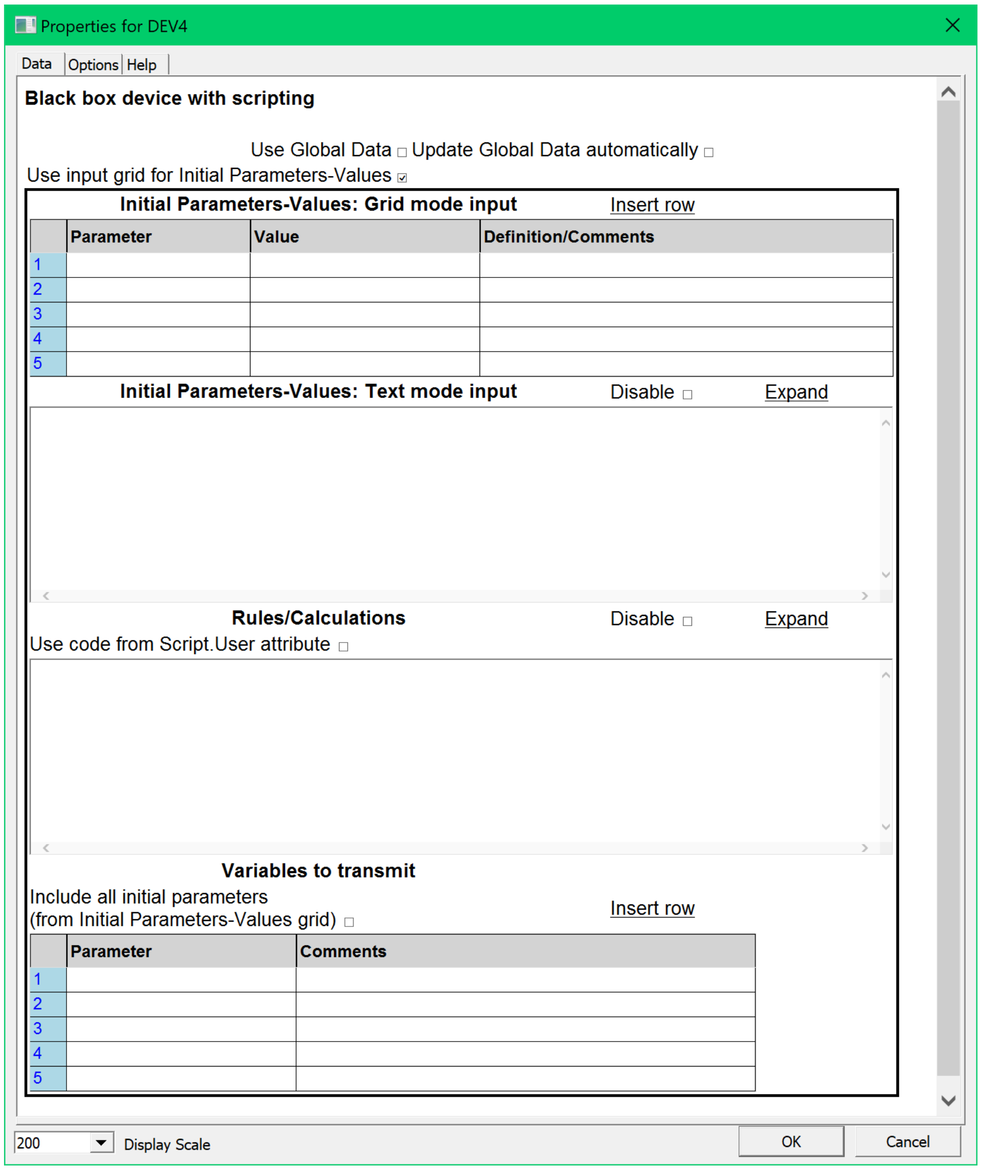
As explained above, the ModelData attribute of the masked device, must contain all final parameters and values transmitted to subcircuit contents.
In addition to manual testing and corrections for error messages, if the user realizes that some data is still not correctly substituted for the devices found in the masked subcircuit, then one option is to request EMTP to provide the final Netlist it is decoding. The final Netlist includes all variable substitutions and can be obtained by selecting “Simulate>Advanced>Simulation Options” and then “Dump decoded data” from the Output tab.

4.3 Mask sections

When this mask type is selected and the device is double-clicked then the empty mask shown in Figure 12 is opened. There are three sections in this mask:

- Initial Parameters-Values: for entering parameters and values using “Grid mode input” and/or “Text mode input”.
- Rules/Calculations: for providing codes for the transformation of parameters into actual named values used in the masked subcircuit contents.
- Variables to transmit: for selecting the variables resulting from calculations or entered directly (in Initial Parameters-Values) that must be transmitted to subcircuit contents. These are the named values used in the masked subcircuit contents.

Some or all sections can be hidden using options available in Options tab. Some sections can be disabled. The disabled sections are not used in the execution of the mask.
The checkboxes “Use Global Data” and “Update Global Data automatically” will be explained in a following section.



Black box device with scripting

Use Global Data ☐ Update Global Data automatically ☐

Use input grid for Initial Parameters-Values ☑

Initial Parameters-Values: Grid mode input

	Parameter	Value	Definition/Comments
1			
2			
3			
4			
5			

Initial Parameters-Values: Text mode input Disable ☐

Rules/Calculations Disable ☐

Use code from Script.User attribute ☐

Variables to transmit

Include all initial parameters (from Initial Parameters-Values grid) ☐

	Parameter	Comments
1		
2		
3		
4		
5		

Figure 12 Empty script_black_box.dwj mask version

4.3.1 Initial Parameters-Values

In this section (see Figure 12) the user can enter the mask parameters and values. There are two options: “Grid mode input” and/or “Text mode input”. Although the “Grid mode input” should be the preferred choice for users, and the “Text mode input” can be disabled (Disable checkbox), it could be useful for some cases, where long and complicated expressions must be entered.

4.3.1.1 Grid mode input

In the “Grid mode input” version:

1. Parameter: any parameter name. Illegal names will be rejected by the software.
2. Value: the value for entered parameter.
    a. It is allowed to enter vectors using the JavaScript format, example: [1, 2].
    b. It is allowed to enter strings using the JavaScript format, example: ‘?v,?i,’
    c. It is allowed to enter global variables, example: oGlobalData.Deltat

   d. It is not allowed to enter units. The users may however decide to use a scale (for example, 10 may mean 10mH) and make the appropriate conversions in the “Rules/Calculations” section presented below.
   e. The Parameter name can be left blank.
   f. The Value cell is evaluated only when the corresponding Parameter name is not blank.
3. Definition/Comments: the user can enter helpful information about each Parameter.
   a. This cell is free-text and multiline.
   b. The complete contents also appear when the mouse is placed over the cell.
   c. The cell contents can be left blank.
4. The input grid can be modified and adjusted using the keyboard options listed in the Tooltip of the title “Initial Parameters-Values”.
5. The Parameter-Value pairs are assembled and evaluated by the mask using JavaScript.

4.3.1.2 Text mode input

In the “Text mode input” parameters and values are entered using JavaScript. The full set of the JavaScript language with EMTPWorks extensions is supported. The underlying code simply applies the JavaScript “eval” function on the contents of this text area. This approach provides more flexibility and can be used in combination with the “Grid mode input” to enter long vectors or other complex expressions.
The “Text mode input” also allows to enter vectors, strings and global variables. For the strings transmitted to the EMTP code (Netlist) it is necessary to use enclosing double quotes. This is because EMTP decodes strings entered in between single quotes.
As for the “Grid mode input” mode, It is not allowed to enter units for parameter values. The users may however decide to use a scale (for example, 10 may mean 10mH) and make the appropriate conversions in the “Rules/Calculations” section presented below.
An example of long vector usage in the “Text mode input” area is given by (see usage for Figure 11):

```
PhiLnonl=[682.23, 733.402, 784.569, 818.68164129536, 839.148682327744, 1173.4436858566828];
```

Such vectors are automatically converted to the required EMTP format when the user clicks on the OK button.

The “Text mode input” area can be left empty or disabled (Disable checkbox), in which case its contents are ignored.

4.3.2 Rules/Calculations

This section is for entering actual rules using the full JavaScript language with EMTPWorks extensions, such as data testing or mathematical operations. For example:

```
//limit L_fault
if( L_fault > 90e-03) {
  L_fault= 90e-03;
}
L_fault = L_fault/2/PI/60
```

This section is also evaluated using the “eval” function. There is no actual boundary between the two sections, since both sections can contain any JavaScript code, it is just for more convenient presentation to other users of the black box device. The Rules (abbreviated named for Rules/Calculations) section can be left empty.
It is possible to call various script files and other external codes from this section.
The Rules section can be also disabled (Disable checkbox), in which case, the contents of the Rules text area are ignored.

4.3.2.1Use code from Script.User attribute

The “Use code from Script.User attribute” allows evaluating codes saved into the device Script.User attribute. In some cases, when the codes are not very large, it is possible to save codes into the Script.User attribute. This allows better design portability. The codes can be edited using a separate file editor and saved back into the Script.User attribute.

4.3.2.2 Script files

It allowed to call script files from the Rules section. The contents of the Rules section can be saved in a file called myrules.dwj (the “.js” extension is also acceptable):

```
myrules();
function myrules(){

   with (Math){
      //limit L_fault
      if( L_fault > 90e-03) {
        L_fault= 90e-03;
      }
      L_fault = L_fault/2/PI/60;
   }
}
```

To call this function file from the Rules section, the user must provide the full path of the script file and parse it using an EMTPWorks function. The contents of the Rules section become:

```
parseScriptFile("C:\\d\\myEMTPWScripts\\myrules.dwj");
```

The parseSriptFile is an EMTPWorks command for parsing the script file myrules.dwj located in the directory "C:\d\myEMTPWScripts\”. If the function calling method is not provided in the script file, then it is needed to call the function separately:

```
parseScriptFile("C:\\d\\myEMTPWScripts\\myrules.dwj");
myrules();
```

If the script file path is not entered, then EMTPWorks will search existing default paths. These are named in the EMTP.INI file (located in the EMTPWorks directory) under the header [JScript]. The last searched location is the folder of the actual design file.
Another option is to add search paths dynamically through a JavaScript call. This can be done, for example, by opening a new JavaScript window (File>New>JavaScript) and entering the command:

```
addScriptSearchPath('C:/d/myEMTPWScripts');
```

Push CTRL-R to run the command. This is an EMTPWorks command that puts the named directory into the EMTPWorks search path for scripts. An extra argument specifies whether this setting should be "permanent", i.e. the settings should be saved by the application and restored automatically each time the application is started. If this is false, the settings remain in effect only until the application quits. Defaults to true.

The user can also call script files from the “Initial Parameters-Values Text mode input” section or call scripts from scripts.

4.3.3 Variables to transmit

This section is for choosing the variables to transmit to the devices contained in the masked subcircuit. The transmitted variables are the same as the named values (enclosed by # characters) entered into device data fields located below the mask.
The following rules must be followed:

1. The “Variables to transmit” must result from computations made in the previous “Initial Parameters-Values” and “Rules/Calculations” sections.
2. It is optionally possible to automatically transmit all Parameters identified in the “Initial Parameters-Values: Grid mode input” section using the checkbox “Include all initial parameters (from Initial Parameters-Values grid)”. This is in addition to those identified in the grid.
3. All selected parameters are transmitted downwards from this mask to all subnetworks and devices found below the mask. This is valid for any number of subnetwork levels. If a parameter is modified in any other mask located below the top mask level where it was initially defined, then the lowest level definition takes precedence. The lowest level mask is the one closest to the device where the name value corresponding to the parameter is used.

4. The input grid can be modified and adjusted using the keyboard options listed in the Tooltip of the title "Variables to transmit".
5. **Important**: If a variable is terminated with the underscore character "_" then its contents are transmitted directly as a string without adding/using its name. This is useful for programming masks that can access other masks.

4.3.4 Example

The script_black_demo.ecf is presented in Figure 13. The Fault_device_ is using the "Grid mode input" and "Text mode input" options.

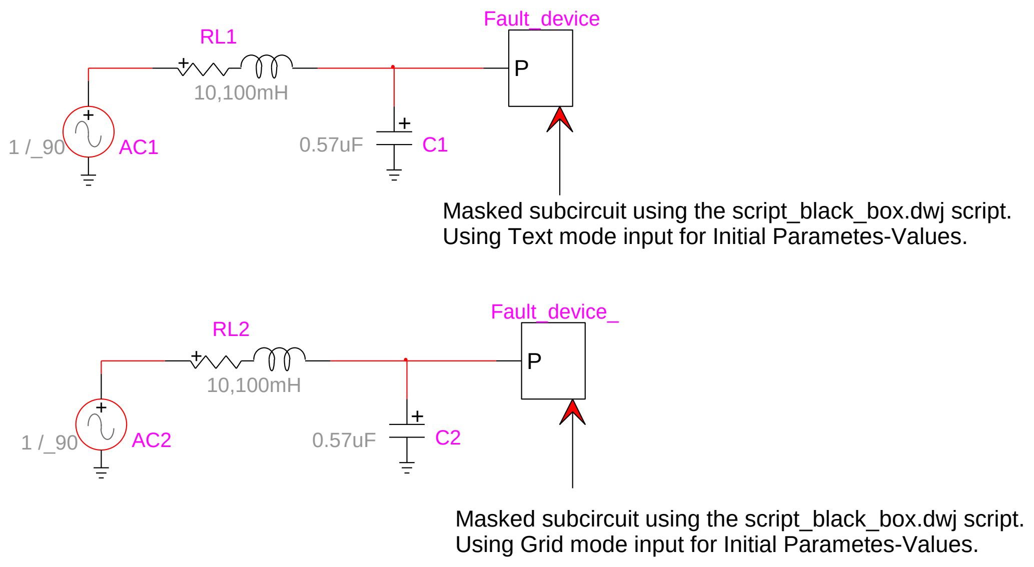
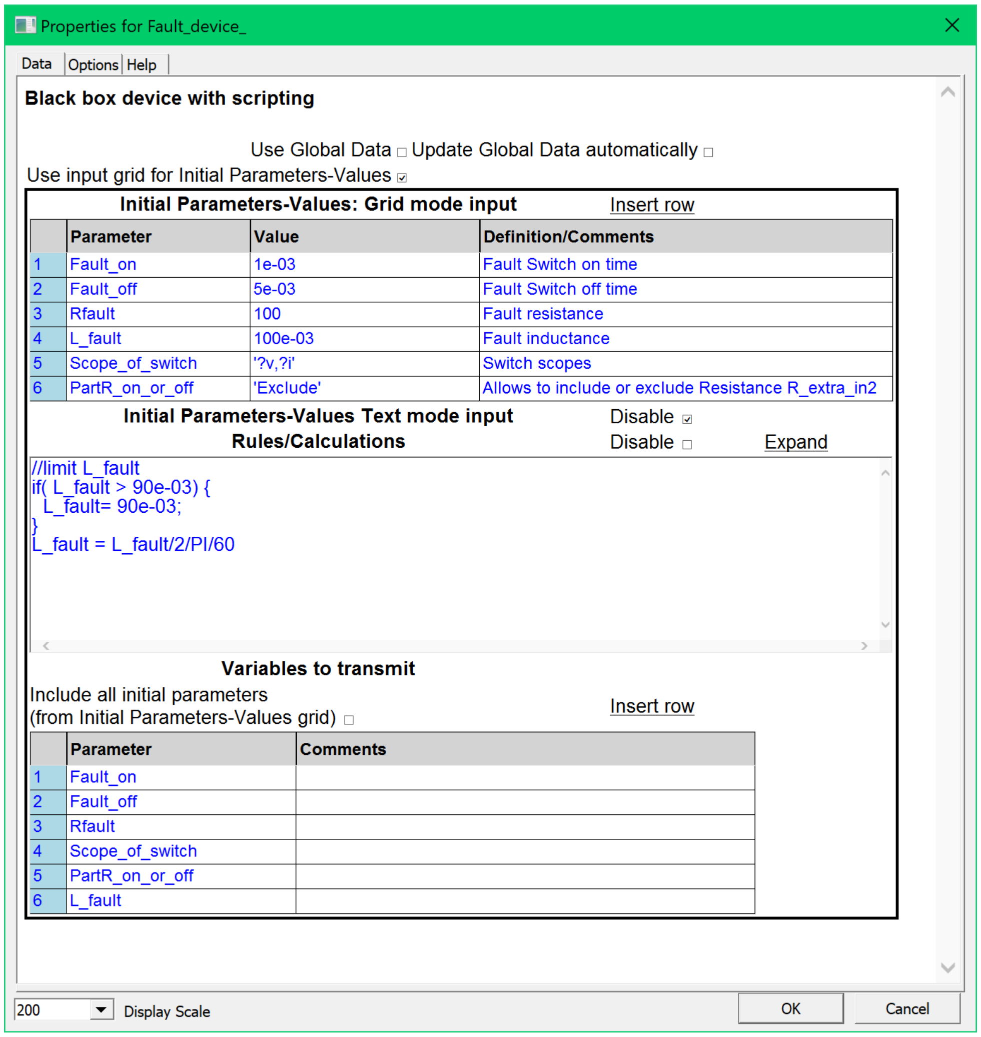

Fault_device
RL1
10,100mH
P
1 /_90
AC1
0.57uF
C1
Masked subcircuit using the script_black_box.dwj script.
Using Text mode input for Initial Parametes-Values.
Fault_device_
RL2
10,100mH
P
1 /_90
AC2
0.57uF
C2
Masked subcircuit using the script_black_box.dwj script.
Using Grid mode input for Initial Parametes-Values.

Figure 13 script_black_demo.ecf design (see subcircuit contents in Figure 5 and Figure 6)

The mask of Fault_device_ ("Grid mode input") is presented in Figure 14. In this case, the "Text mode input" is not used and disabled. A simple computation is made in the Rules area.
When the user clicks OK, the contents of the ModelData attribute become:

```
Fault_on=0.001;
Fault_off=0.005;
Rfault=100;
Scope_of_switch='?v,?i';
PartR_on_or_off='Exclude';
L_fault=0.000238732414637843298;
```

This attribute is transmitted to EMTP for computations.



Black box device with scripting

Use Global Data ☐ Update Global Data automatically ☐

Use input grid for Initial Parameters-Values ☑

Initial Parameters-Values: Grid mode input Insert row

	Parameter	Value	Definition/Comments
1	Fault_on	1e-03	Fault Switch on time
2	Fault_off	5e-03	Fault Switch off time
3	Rfault	100	Fault resistance
4	L_fault	100e-03	Fault inductance
5	Scope_of_switch	'?v,?i'	Switch scopes
6	PartR_on_or_off	'Exclude'	Allows to include or exclude Resistance R_extra_in2

Initial Parameters-Values Text mode input Disable ☑

Rules/Calculations Disable ☐ Expand

```
//limit L_fault
if( L_fault > 90e-03) {
  L_fault= 90e-03;
}
L_fault = L_fault/2/PI/60
```

Variables to transmit

Include all initial parameters
(from Initial Parameters-Values grid) ☐ Insert row

	Parameter	Comments
1	Fault_on	
2	Fault_off	
3	Rfault	
4	Scope_of_switch	
5	PartR_on_or_off	
6	L_fault	

200 Display Scale OK Cancel

Figure 14 Mask of Fault_device_ using “Grid mode input”

The above example is presented in Figure 15 using the “Text mode input”.
There are 3 sections in Figure 15.

- Initial Parameters-Values: Text mode input

The contents of the text area are:

```
//Switch times:
Fault_on =1e-03;
Fault_off=5e-03;
//Fault resistance:
Rfault = 100;
//Scopes in the subnetwork Fault_device
Scope_of_switch="'?v,?i'";
//Turn on the Rextra
PartR_on_or_off="'Exclude'";
//define fault inductance
L_fault=100e-03;
```

It is noticed that since the EMTP code expects to receive strings enclosed between quotes, the Scope_of_switch string parameter must include the quotes using double quotes. This is not needed in the “Grid mode input” option, since it able to add the double quotes automatically.

- Rules/Calculations

This section is for entering actual rules and calculations using the JavaScript language, such as data testing or mathematical operations. In this example:

```
//limit L_fault
if( L_fault > 90e-03) {
  L_fault= 90e-03;
}
L_fault = L_fault/2/PI/60
```

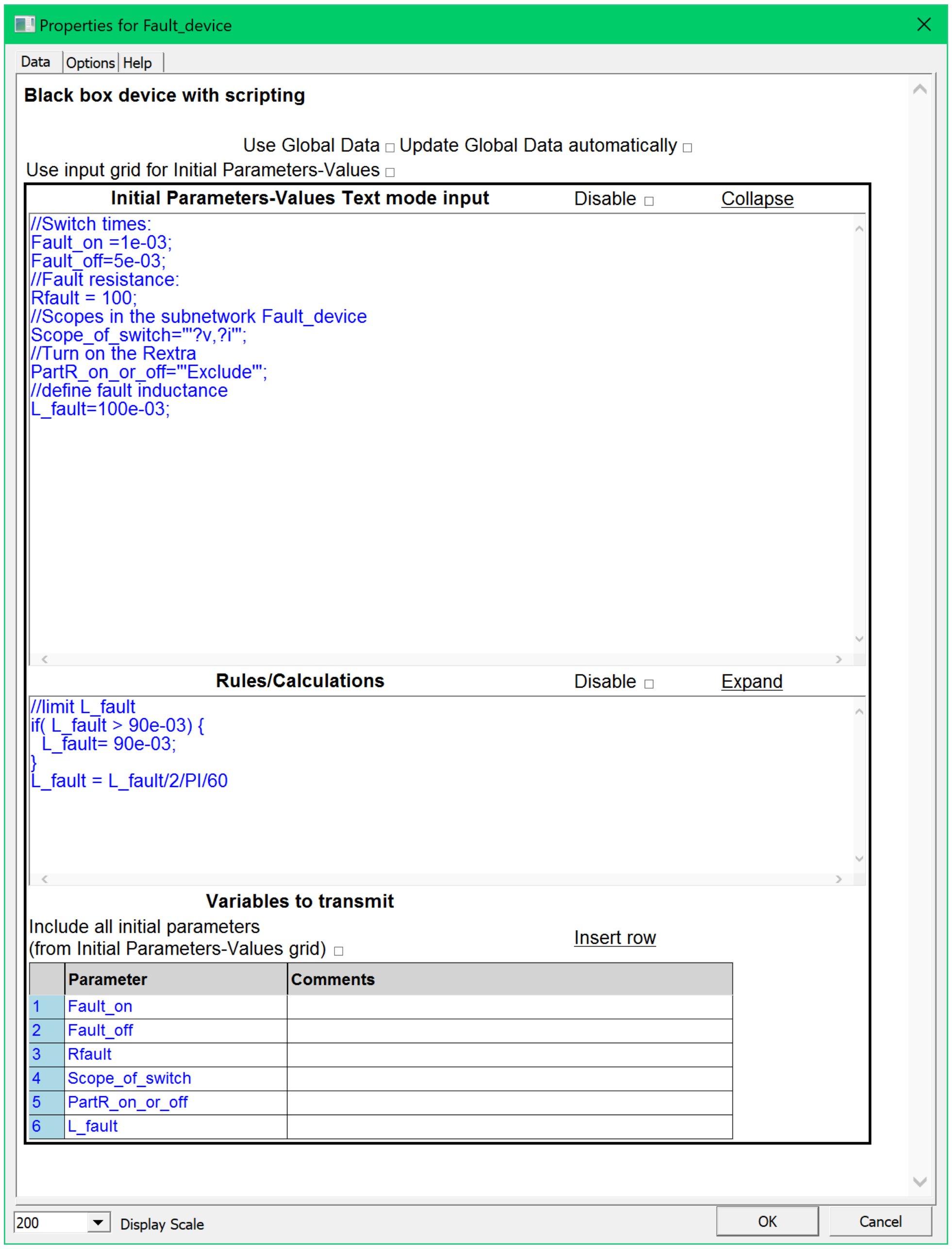
This section is also evaluated using the “eval” function. There is no actual boundary between the two sections above, since both sections can contain any JavaScript code, it is just for more convenient presentation purposes. The Rules section can be empty.

The mask of the subnetwork DEVF shown in Figure 7 is not changed in this example. It demonstrates that it is allowed to mix different property scripts within various subnetwork levels. It is however noticed that the default setup does not include any global data communications between script_black_box.dwj masks. Such exchanges can be however established through more advanced programming available in EMTPWorks.

In this example, it would have also been possible to use a script_black_box.dwj for DEVF, as shown in Figure 16. It is noticed here that such a change of variables is not possible through “Grid mode input” since it allows transmitting only strings and not variable names. The single quotes are used by the JavaScript “eval” function to create:

```
Rextra=Rfault;
```

in the ModelData attribute. It is noticed that the above statement is not needed if there is no change in variable name or specific need, in other words, Rfault is transmitted directly downwards to all devices and can replace targeted named values.



Black box device with scripting

Use Global Data ☐ Update Global Data automatically ☐

Use input grid for Initial Parameters-Values ☐

Initial Parameters-Values Text mode input Disable ☐

```
//Switch times:
Fault_on =1e-03;
Fault_off=5e-03;
//Fault resistance:
Rfault = 100;
//Scopes in the subnetwork Fault_device
Scope_of_switch="'?v,?i'";
//Turn on the Rextra
PartR_on_or_off="'Exclude'";
//define fault inductance
L_fault=100e-03;
```

Rules/Calculations Disable ☐

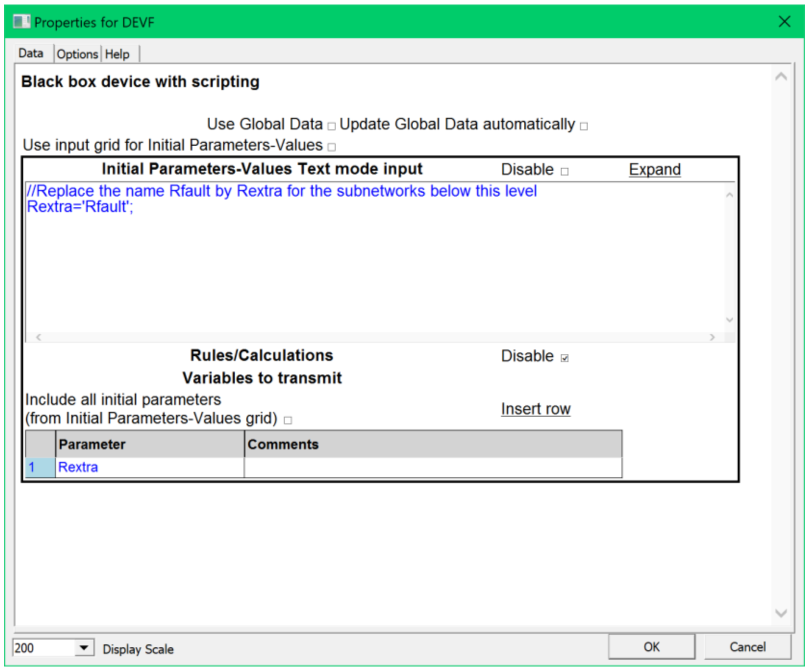
```
//limit L_fault
if( L_fault > 90e-03) {
  L_fault= 90e-03;
}
L_fault = L_fault/2/PI/60
```

Variables to transmit

Include all initial parameters
(from Initial Parameters-Values grid) ☐

	Parameter	Comments
1	Fault_on	
2	Fault_off	
3	Rfault	
4	Scope_of_switch	
5	PartR_on_or_off	
6	L_fault	

Figure 15 Mask of Fault_device using “Grid mode input”


Properties for DEVF

Black box device with scripting

Use Global Data ☐ Update Global Data automatically ☐

Use input grid for Initial Parameters-Values ☐

Initial Parameters-Values Text mode input Disable ☐ Expand

```
//Replace the name Rfault by Rextra for the subnetworks below this level
Rextra='Rfault';
```

Rules/Calculations Disable ☑

Variables to transmit

Include all initial parameters (from Initial Parameters-Values grid) ☐

Insert row

	Parameter	Comments
1	Rextra	

200 Display Scale OK Cancel

Figure 16 The script_black_box.dwj version of the mask in Figure 7

4.4 Using Global Data

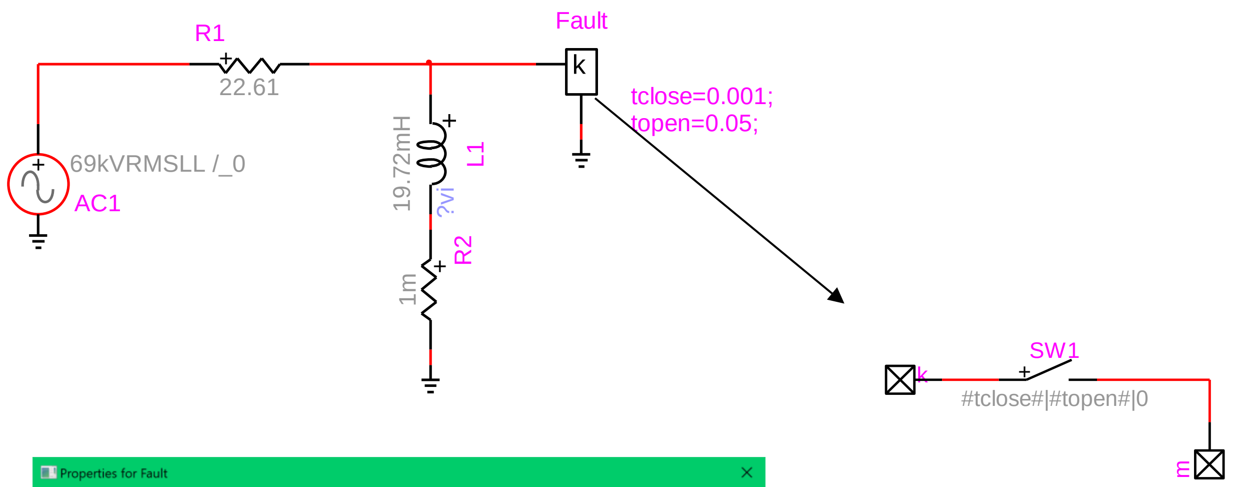
On the first tab of the script_black_box.dwj device there are two options for global data usage. Global data usage is available through a global object named oGlobalData. This object is created internally for object data fields and methods. Some of the Simulation options, such as time-step and maximum simulation time are also available through the global object. The user can create other variables accessed through the global object by using the EMTPWorks menu "Design>Utilities>Define Global Variables" (Global variable definition). More documentation on the global object oGlobalData creation and usage is available in the help menu of the panel opened when "Design>Utilities>Define Global Variables" is clicked. Hereinafter the term Global variable definition is used for global variables defined through the panel "Design>Utilities>Define Global Variables" or other top level menus, such as "Simulate>Advanced>Simulation Options".

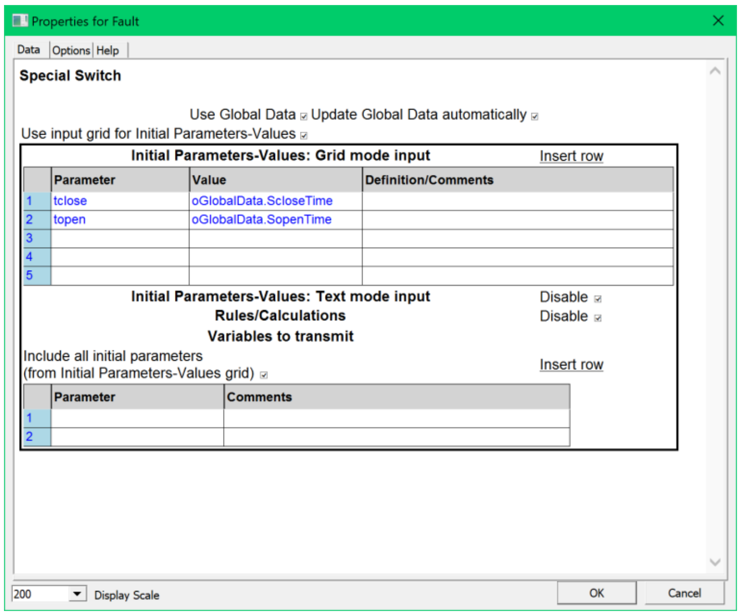
The basic rules to remember when using oGlobalData in a script_black_box.dwj mask are:

1. To use oGlobalData the checkbox "Use Global Data" must be on. This allows to access oGlobalData fields in all scripts of the mask.
2. If the script needs to continuously update all changes in oGlobalData then the checkbox "Update Global Data automatically" must be on. This will trigger the mask scripts automatically at every change in oGlobalData through the Global variable definition process. The triggering will also occur before starting the simulation and this will guarantee that all variables in all masks are up to date.
3. If the script needs to grab a variable without updating it, then "Use Global Data" is sufficient. This could cause problems however when the corresponding data field is redefined due to a change in Global variable definition.
4. Password protected device contents are not eligible for automatic updating. This means that masked devices contained in password protected subcircuits are not automatically updated with changes in global data. The mask of the password protected device is updated and it must perform its own internal definition functions.
5. In the case of subcircuit devices marked as Read-Only (Lock opening subcircuit selected from right-click menu Properties) the updating permission for device contents is optional (see Options tab below).
6. A given mask may also define (add) new data fields in oGlobalData, but this is not a standard procedure since there will be no automatic updating in other masks and the order of updating through the Global variable definition process is not predictable.
7. If the used global object oGlobalData field is not defined, then it will be given the value "undefined".

8. Documentation on built-in variables is available in the Help section of “Design>Utilities>Define Global Variables”. When the design is opened after saving, the Global variable definition process is automatically started, but the device masks are not triggered since their data has been already established before saving the design.

A simple example of global data usage is shown in Figure 17. The contents of the mask for the device named Fault are used to define switch closing and opening times for the switch contained in the subcircuit of the device. The mask is using two global variables defined in the menu “Design>Utilties>Define Global Variables” shown in Figure 18. The global variables are defined before using them in the mask.





Special Switch

Use Global Data ☑ Update Global Data automatically ☑

Use input grid for Initial Parameters-Values ☑

Initial Parameters-Values: Grid mode input Insert row

	Parameter	Value	Definition/Comments
1	tclose	oGlobalData.ScloseTime	
2	topen	oGlobalData.SopenTime	
3			
4			
5			

Initial Parameters-Values: Text mode input Disable ☑

Rules/Calculations Disable ☑

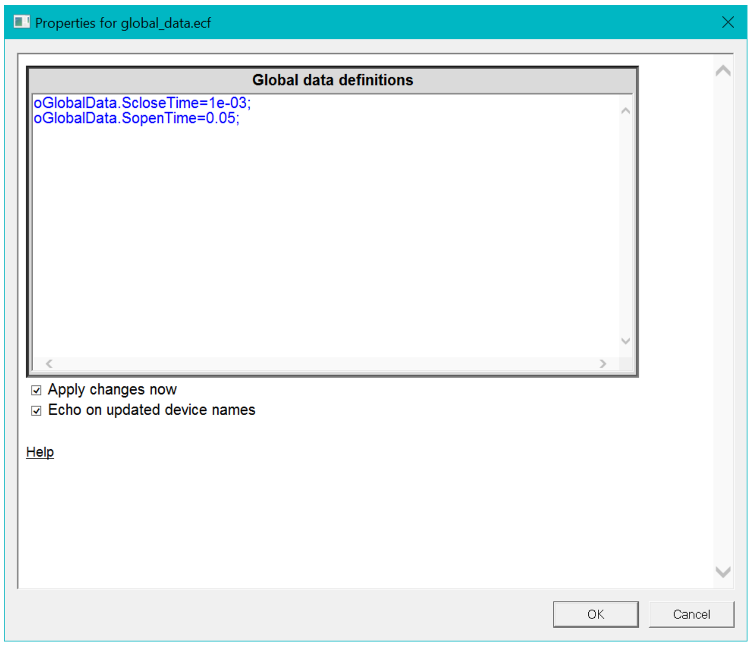
Variables to transmit

Include all initial parameters
(from Initial Parameters-Values grid) ☑ Insert row

	Parameter	Comments
1		
2		

200 Display Scale

Figure 17 Global data usage example (global_data.ecf)



```
oGlobalData.ScloseTime=1e-03;
oGlobalData.SopenTime=0.05;
```

☑ Apply changes now
☑ Echo on updated device names

Figure 18 Definition of Global data using “Design>Utilities>Define Global Variables”

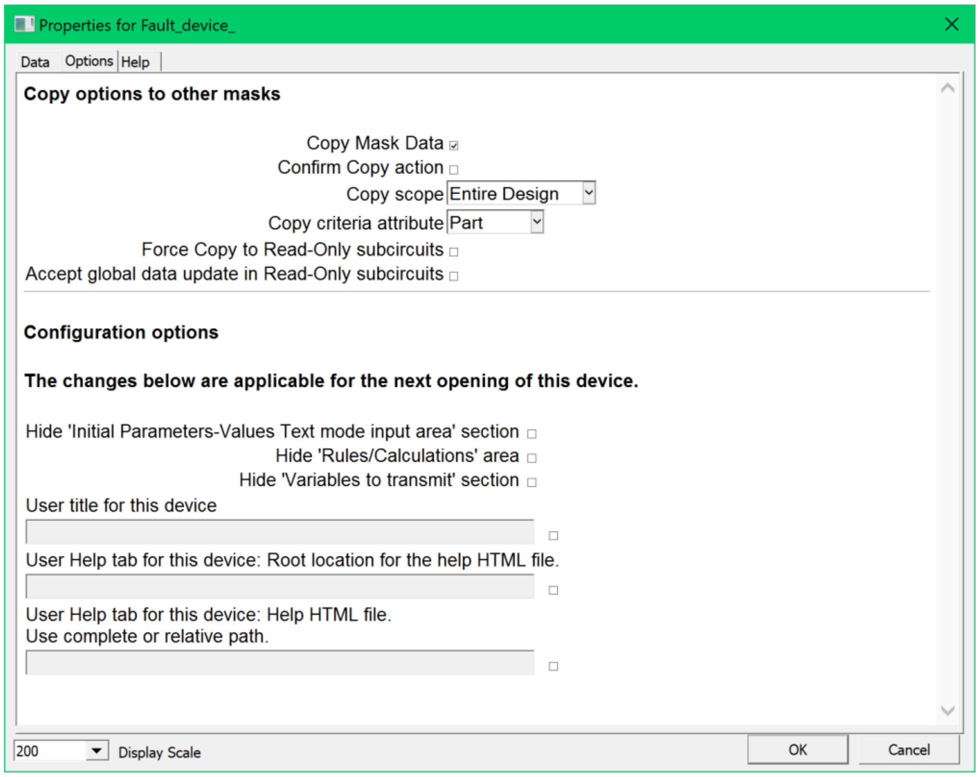
4.5 Options

The second tab of the mask data web is used to define various options. The default version of this tab is shown in Figure 19. The user can use tooltips to get quick indications on the presented options. The available options are:

1. **Copy Mask Data**: If this option is checked, the data entered on the previous tab will be copied to other devices using the scope and criteria below. The copy is performed only for devices with the same mask script. The copy is performed only when the user clicks OK. This option is useful for maintaining data in similar masked devices. When this option is checked, the user is offered the choice of “Confirm Copy action” for each device. The “Copy scope” offers the selection of design scope for applying the copy action. The “Copy criteria attribute” is a method for selecting the targeted masked devices to which the copy action is performed.
2. **Force Copy to Read-Only subcircuits**: When this option is selected, the Copy Mask Data action is allowed to be performed on devices contained in Read-Only subcircuits.
3. **Accept global data update in Read-Only subcircuits**: This checkbox is related to granting permission for updating global data (explained above) in masks for devices contained in Read-Only subcircuits.
4. **Hide 'Initial Parameters-Values Text mode input area' section**: Selecting this option will hide the indicated section in the previous data tab. This is useful if the entered data does not require changes from the user and offers some protection. This option takes effect only after clicking OK.
5. **Hide ‘Rules/Calculations' area**: Selecting this option will hide the “Rules/Calculations” section of the previous data tab. This is useful if the entered data does not require changes from the user and offers some protection. This option takes effect only after clicking OK.
6. **Hide ‘Variable to transmit’ section**: Selecting this option will hide the “Variables to transmit” section of the previous data tab. This is useful if the entered data does not require changes from the user and offers some protection. This option takes effect only after clicking OK.
7. **User title for this device**: Specify a title that will appear instead of the default title shown on the previous tab. Enable this field by clicking on its checkbox. The field is disabled by default.
8. **User Help tab for this device**: There are two options. The “Root location for the help HTML file” and the “Help HTML file”. The objective is to change the contents of the default Help tab by replacing it with the Help documents of the masked device.
    a. The Root location allows specifying a folder located in a folder below the current script location. This is normally "C:\Program Files\EMTPWorks\Info Scripts\". It can be left empty.

b. The “Help HTML file” is the actual HTML file with its own internal links.

An example of Help tab customization is given by the built-in device “i(t) 3-phase probe” available in the “meters.clf” library. This Options tab of this device is using “meters” for the Root location and “i_3ph_probe/help.htm” for actual Help file. This means that the complete file location is resolved into: "C:\Program Files (x86)\EMTPWorks\Info Scripts\meters\ i_3ph_probe/help.htm".


Properties for Fault_device_

Data | Options | Help

Copy options to other masks

Copy Mask Data ☑
Confirm Copy action ☐
Copy scope: Entire Design
Copy criteria attribute: Part
Force Copy to Read-Only subcircuits ☐
Accept global data update in Read-Only subcircuits ☐

Configuration options

The changes below are applicable for the next opening of this device.

Hide 'Initial Parameters-Values Text mode input area' section ☐
Hide 'Rules/Calculations' area ☐
Hide 'Variables to transmit' section ☐
User title for this device ☐
User Help tab for this device: Root location for the help HTML file. ☐
User Help tab for this device: Help HTML file.
Use complete or relative path. ☐

200 Display Scale

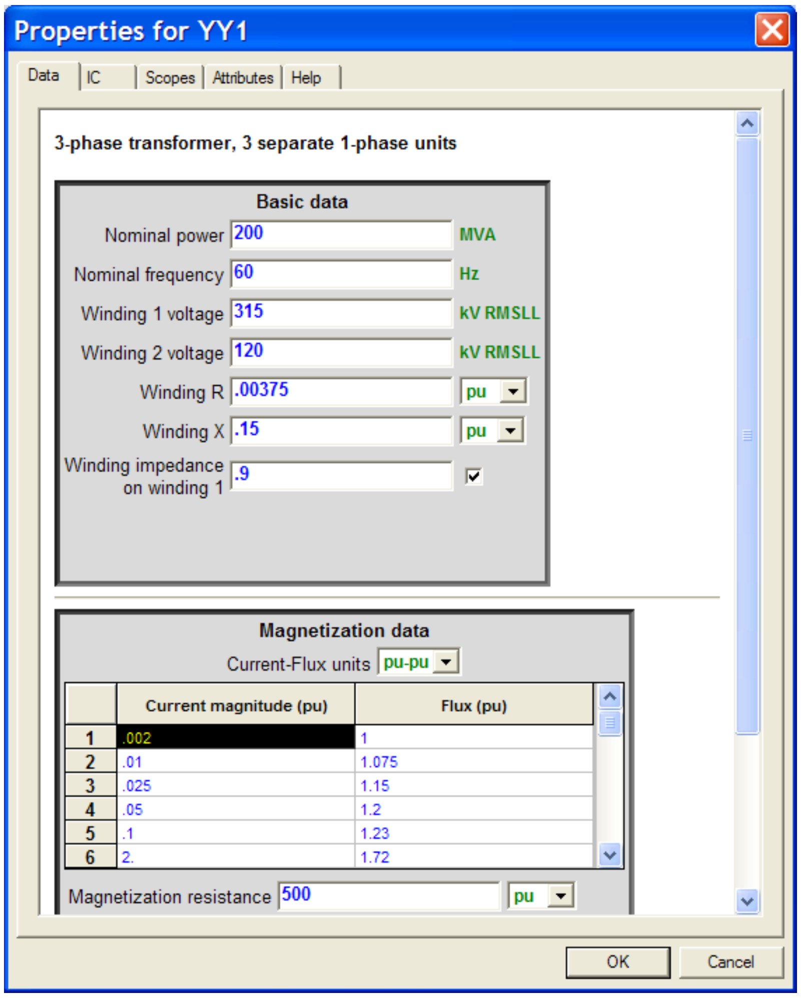
Figure 19 Tab Options for script_black_box.dwj with “Copy Mask Data” selection

4.6 Unmasking

The rules are identical to those for the “black_box.dwj” script given above.

5 Advanced scripting

In addition to the standard JavaScript, EMTPWorks provides a large set of methods available through JavaScript codes for exchanging and setting data for various tasks. It is feasible to create advanced masks, such as the one shown in Figure 20. It is a combination of JavaScript code with special EMTPWorks methods and DHTML data forms. The number of options and possibilities is very high. Users can study the script “yy_info.dwj” located in the directory “Info Scripts\branches”.



Properties for YY1

Data | IC | Scopes | Attributes | Help

3-phase transformer, 3 separate 1-phase units

Basic data

Nominal power	200	MVA
Nominal frequency	60	Hz
Winding 1 voltage	315	kV RMSLL
Winding 2 voltage	120	kV RMSLL
Winding R	.00375	pu
Winding X	.15	pu
Winding impedance on winding 1	.9	✓

Magnetization data

Current-Flux units pu-pu

	Current magnitude (pu)	Flux (pu)
1	.002	1
2	.01	1.075
3	.025	1.15
4	.05	1.2
5	.1	1.23
6	2.	1.72

Magnetization resistance 500 pu

Figure 20 Advanced mask for the transformer of Figure 9

5.1 Scripting methods

To create advanced masks you must learn the JavaScript language and DHTML (Dynamic HTML). The required level of knowledge can be minimal in most cases. You can also reuse and modify existing scripts.
The EMTPWorks architecture is shown in Figure 21. The bottom layer is programmed using C/C++. The intermediate layer is an exchange layer programmed in EMTPWorks extended JavaScript. The JavaScript language is extended with a large set of EMTPWorks methods (see electronic documentation “JavaScript based scripting”). In addition to providing a library of useful programming functions, this layer allows modifying device attributes in EMTPworks. Although any extension can be used, the standard approach is to use files with the extension “dwj” for the programming of this layer. This is why this layer is called the dwj-layer.
The top level can be actually based on any method callable from the dwj-layer. The standard approach is to use again JavaScript with DHTML. Most of the EMTPWorks extensions to JavaScript are also available at this level.

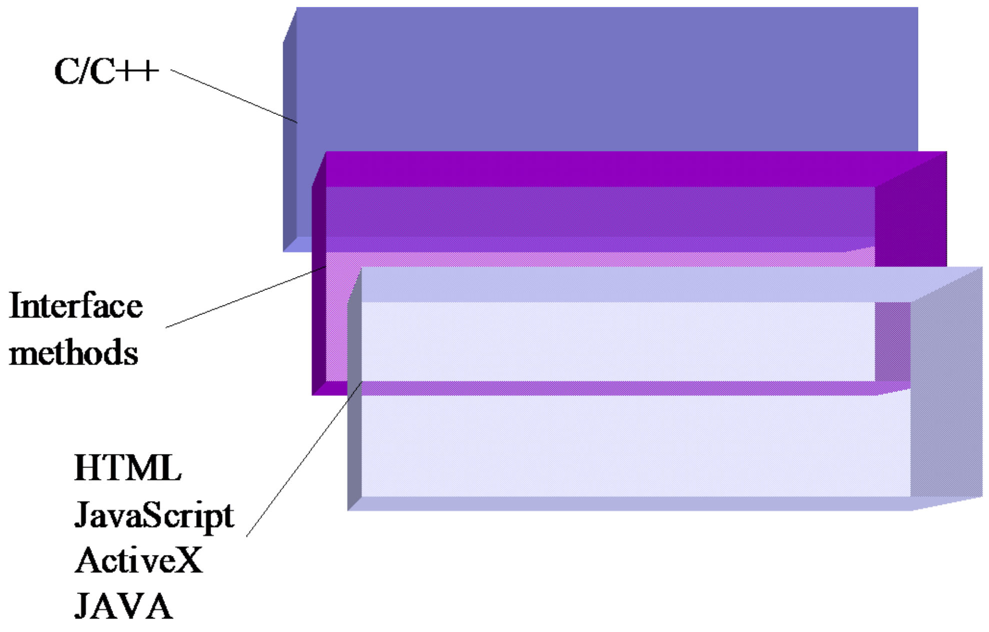


Figure 21 EMTPWorks architecture layers

Although the scripting methods are generic and can be used for various tasks, they are explained here in the context of advanced mask programming methods.

5.2 Example: programming the mask of “RLC_Load (PQ)”

This section presents the programming of the “RLC_Load (PQ)” mask. The first step is to create a subcircuit from a given circuit. The subcircuit level of “RLC_Load (PQ)” is shown in Figure 22.

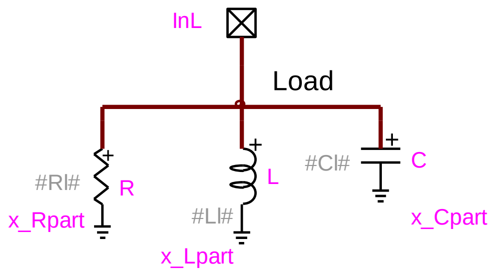


Figure 22 The subcircuit contents of “RLC_Load (PQ)”

All data fields are given named values. All branches (R, L and C) are given a programmable scope variable in the Attributes data tab. In the case of R, all three phases are set to the “Rl_scope” variable. The same is fixed for L and C by changing the appropriate letter.
The next (optional) step is to modify the device symbol using the Symbol Editor (“Edit Symbol” right-click command). If the new device is to be reused frequently in new designs then it is best to give it a significant Part attribute and save it into a user library under a significant name. In this case this device is part of the built-in libraries and it is saved under the name “PQ load (RLC)” in “Load Models.clf”. The chosen Part attribute is “RLC_Load”. Several changes to device attributes can be made before and after saving it into its library. Attributes can be set in the design or in the Symbol Editor.
The next change is to mask the device. This can be done using the standard masking approach available in the right-click menu “Subcircuit Info”. Another approach is to modify directly the attribute Script.Open.Dev. This attribute names the script called when the device is double-clicked. In this is case it is named rlcload_info.dwj.

The named script must be in the search path of EMTPWorks. In this case it is located in the folder "Info Scripts\branches" which is explicitly named in the EMTP.INI file as being in the searched folder list.

The rlcload_info.dwj JavaScript code is given by:

```
//*This script displays a properties box for the RLC Load device type
//JavaScript based design with methods from EMTPWorks
//
//*Get this object
dev = defaultObject();     // Get the currently selected device

//*Parse all methods for this device
parseScriptFile('rlcload_m.dwj');

//*Crate the object
var oDevice = new oDevice_RLC_Load (dev);

//*Call to open the data forms of this object
oDevice.open();  //will call the save method when OK is clicked
```

This is a standard programming approach for all scripted-mask-subcircuit devices available in EMTPWorks. In a future release this approach will be propagated to all devices including non subcircuit based (primitive) devices. The first statement calls the EMTPWorks function defaultObject to retrieve the device object. The second statement parses all methods of this device. The methods file of the device contains a standardized set of functions. The device object holds its data and provides its methods. The currently standardized set of object fields is:

- Various data fields as documented for each device. These fields provide access to all user modifiable data for the device. They area initialized from the data originally entered into the device data forms. All devices are initialized.
- SaveData: the method for saving data into device attributes. Saved data is transmitted into the Netlist.
- open: the method for opening the device data forms. This is similar to double-clicking on the device.
- dev: holds the device object handle in the design.
- ExportDev: initialized to dev. This field can be used to change the device into which the SaveData method is saving data. This is useful for exporting the device mask to an upper level.
- SaveFormData: initialized to true, used to cancel saving data into data forms.

In the code lines shown above, the device is initialized by calling its object creation method oDevice_ and then it is opened by calling the open method. If the user makes changes into data forms and clicks OK, then the open method will automatically call the SaveData method.

The contents of rlcload_m.dwj are self-explanatory. The programming of such a script can use a minimal programming effort to become functional. A more complete code is designed to account for the complete set of programming and standardization rules available in EMTPWorks. It is for making the device into the list of standardized built-in and redistributable devices.
It is noticed that the name of the methods file is always derived from the actual Script.Open.Dev attribute by starting from the root name ("rlcload" in this case) and appending the string "_m".

5.3 Masking subcircuits with scripted-mask-subcircuits

In some cases it may be necessary to mask a subcircuit that contains one or more scripted-mask-subcircuits. Since all entered data in a scripted-mask-subcircuit is completely determined when the OK button is clicked, it is not possible to use named values assigned at a higher level. In the example of Figure 23, DEV1 and DEV2 are identical (function "top") subcircuits containing an "RLC_Load (PQ)" load device named "TargetLoad". It is needed to mask the Part "top" to allow changing only the load power or any desired data set. This is achieved through the mask shown in Figure 24.

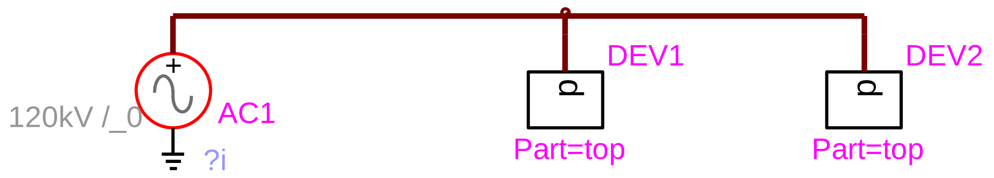


Figure 23 Example with 2 subcircuits containing scripted-mask-subcircuits (maskload.ecf)

The actual contents of the mask section for Initial values are:

```
ModelData=runme();

function runme(){
mydev = defaultObject();
sub=      mydev.subCircuit;
devs     =sub.devices('Name','TargetLoad');

//*Parse all methods of the load
parseScriptFile('rlcload_m.dwj');

//*Create the load object
LoadData= new oDevice_RLC_Load (devs[0])

//just change power and scope
LoadData.Ppower='200';
LoadData.Rl_i_scope='?i'
//Export the mask data to this level
LoadData.ExportDev=mydev;
LoadData.SaveFormData=false; //do not save back
LoadData.SaveData()
return ModelData=mydev.getAttribute('ModelData');
}
```

Bold characters are used to highlight JavaScript functions provided by EMTPWorks. Notes:

- The “mydev” object is the handle to the double-clicked device, in this case DEV1.
- The array devs is used to find the target object (device), named “TargetLoad”
- The methods of target ““RLC_Load (PQ)” are parsed using its methods script.
- The object LoadData is created using the target device handle devs[0] and holds all the data fields and methods of the target device. The data fields can be found in the target device documentation. All data fields are initialized from the existing data already entered into the device.
- The next statement changes the load power to 200. The units are not changed and remain MW.
- The next statement turns on the Resistance current scope.
- Since the mask of the target device must be exported to the level of the top device, it is needed to set the ExportDev field to mydev.
- SaveFormData is turned off, since we do not want the changes to appear in the target device which is being used also by DEV2. This ensures that the mask of DEV2 also starts from initial values of the target device.
- The SaveData method is called to calculate and save the target device data into mydev attribute ModelData, which is then retrieved and sent out from the function.

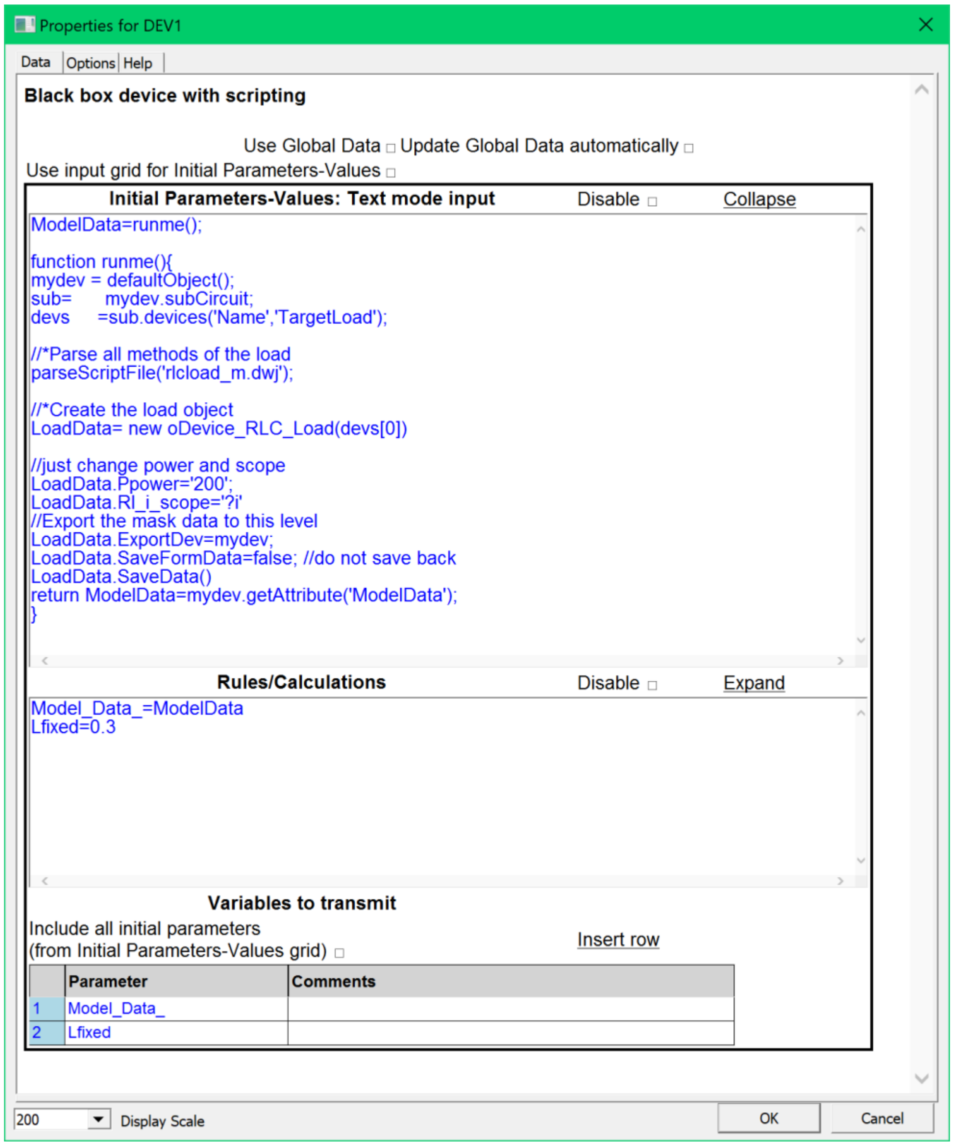
In the rules section the variable Model_Data_ is simply set to ModelData. The last underscore appearing in the name of the transmitted variable is important, it is a code indicating that the variable contents (string) must be sent directly without including its name.

The same mask is used for DEV2, only now it can set different data fields and data values without affecting DEV1 and its contents.
In a more complex case, the search and retrieval of the target device may require more sophistication since it may appear anywhere in the subcircuit hierarchy. In this code it is simply searched from the current level:

```
sub=      mydev.subCircuit;
devs     =sub.devices(512,-1,5,'Name','TargetLoad'); //512 is a filter code, -1 means no restrictions, 5 is the scope: here down
```

It is important to give the target device a unique name, to avoid for possible finding and mixing more than one device in a complicated hierarchical case.

The mask of Figure 24 is not restrictive, it can be used to mask other devices appearing in its subcircuit. In other words it can be used to set other named values or scripted-mask-subcircuits. Several different functions can be called from the “Initial values” section and other variables can be transmitted. In the example of Figure 24, an extra variable Lfixed is sending data for an inductance appearing in the subcircuit.


Properties for DEV1
Data
Options
Help
Black box device with scripting
Use Global Data
Update Global Data automatically
Use input grid for Initial Parameters-Values
Initial Parameters-Values: Text mode input
Disable
Collapse

```
ModelData=runme();

function runme(){
mydev = defaultObject();
sub=      mydev.subCircuit;
devs     =sub.devices('Name','TargetLoad');

//*Parse all methods of the load
parseScriptFile('rlcload_m.dwj');

//*Create the load object
LoadData= new oDevice_RLC_Load(devs[0])

//just change power and scope
LoadData.Ppower='200';
LoadData.Rl_i_scope='?i'
//Export the mask data to this level
LoadData.ExportDev=mydev;
LoadData.SaveFormData=false; //do not save back
LoadData.SaveData()
return ModelData=mydev.getAttribute('ModelData');
}
```

```
Model_Data_=ModelData
Lfixed=0.3
```

Variables to transmit
Include all initial parameters (from Initial Parameters-Values grid)
Insert row

	Parameter	Comments
1	Model_Data_	
2	Lfixed	

Figure 24 Scripted mask for subcircuit DEV1 in Figure 23

In addition to the above approach for working with separate data for the contained masked devices, it is feasible to detach the top device DEV1 data from DEV2 by issuing a “Make Unique Type” (see Options ribbon and “Make Unique Type”) command from the mask of DEV1.