IEEE CIGRE DLL Documentation

Document Version 1.1 - 2023/04/13

Prepared by César Martin from RTE

cesar.martin@rte-france.com

Contents

1. Introduction 2
   1.1 Benefits of a model in the IEEE CIGRE DLL format 2
   1.2 How it works 2
2. Build your model in IEEE CIGRE DLL format 2
   2.1 Various structures and functions to be modified 3
   2.2 Compilation 4
   2.3 Rules and good practice 5
      2.3.1 Messages 5
      2.3.2 Use state arrays to store the model's internal variables 5
3. On the simulation tool side 5
   3.1 Timing of function calls 5
   3.2 EMTP Import Tool 6
   3.3 PSCAD Import Tool 6
4. Current known bugs 6
   4.1 txt_generator_tool_EMTPStdLibs.exe not found in EMTP 7
5. Future work 7
6. Appendices 8
   6.1 How to handle model specific structures using state arrays 8
   6.2 Making an IEEE CIGRE DLL from an already compiled control model 9

1. Introduction

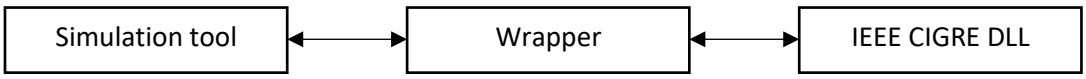
1.1 Benefits of a model in the IEEE CIGRE DLL format

An IEEE CIGRE DLL is a black-box model of a control system. This is a single .dll file (no associated .lib file, the simulation tool will explicitly link to this DLL). It shall only contain control code (no electrical components modelled) and must be independent from a simulation tool and its solver.

There are several advantages to using a DLL rather than other types of black-box model such as .lib file. In particular, the use of DLLs avoids compiler version compatibility concerns. This means that a DLL can be built with any compiler and can be loaded and called from any compiler. A DLL model will be able to run, without modification, on future versions of simulation software.

IEEE CIGRE DLL provides a standard target for simulation tools. By developing an import tool, the process to call this code in any simulation program is automated.

1.2 How it works



An import tool must be developed on the simulation tool side to be able to run a simulation containing a DLL in IEEE CIGRE DLL format.

An IEEE CIGRE DLL import tool includes all the necessary components for the DLL to function within the tool, such as scripts to automatically draw the block along with its inputs and outputs (if the tool is graphical), display of configuration parameters, as well as the Wrapper.

The intermediate code that facilitates communication between the tool and the DLL during the simulation will be referred to as the Wrapper.

The provided example includes the EMTP import tool (which takes the form of an EMTP toolbox). This toolbox will be available in future release versions of EMTP.

For PSCAD, Electranix has developed an import tool, but it is not freely available. A similar tool is being jointly developed by RTE and TU Delft. For information, Electranix has provided the wrapper in PSCAD for the SCRX example (this wrapper is specific to the example and will not work for other IEEE CIGRE DLLs).

See part 3 for more details on the import tools in EMTP and PSCAD.

2. Build your model in IEEE CIGRE DLL format

You should refer to the provided example (folder “SCRX9_dll_source”) to identify the parts to be modified and thus build your own model. The two header files “IEEE_Cigre_DLLInterface.h” and

“IEEE_Cigre_DLLInterface_types.h” must not be modified. The model can of course contain several .c files.

2.1 Various structures and functions to be modified

To build your model, you will have to adapt almost all the structures and functions in the "SCRX9.c" file. Only the "Model_GetInfo" function should not be modified because it is called by the simulation software and its purpose is simply to return the "Model_Info" structure. It is not necessary (but allowed) to modify the "Model_PrintInfo" function as it simply prints information about the model.

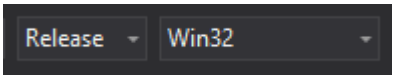
- MyModelInputs: List the inputs in order, including their type and name. Use the types defined at the top of the IEEE_Cigre_DLLInterface_types.h file. If the input is a vector (dimension > 1), use [dim] at the end. Example : int32_T in_name[3];
- InputSignals: Detail the information for each input. Some information is redundant, such as the name and type that need to be indicated again.
    - .Description and .Unit: Specify a small description and the unit (any character string).
    - .DataType: The value must be chosen among the IEEE_Cigre_DLLInterface_DataType enumeration of IEEE_Cigre_DLLInterface_types.h and correspond to the type indicated in MyModelInputs
    - .Width: Allows you to define the size of this input (a vector-type input if width > 1).
- MyModelOutputs and OutputSignals: Specify outputs information. Do the same as with MyModelInputs and InputSignals
- MyModelParameters: Specify model parameters. These parameters will be accessible and modifiable in the simulation software (in the model mask for EMTP and PSCAD). This structure must be filled in the same way as MyModelInputs or MyModelOutputs.
- Parameters: Same as InputSignals or OutputSignals with additional attributes.
    - .FixedValue: Can be 0 or 1. 0 for parameters which can be modified at any time, 1 for parameters which need to be defined at T0 but cannot be changed. For information, the current version of the EMTP import tool does not allow parameter values to be changed during the simulation, only at the beginning of the simulation. It is planned to be able to transform these parameters into model inputs, in EMTP, if the user wants to vary them during the simulation. This functionality will be available in the next version of the EMTP import tool.
    - .DefaultValue.<Type>: Allows you to define the default value of the parameter as well as its type using the .<Type>. The type must be chosen among the "union DefaultValueU" values of IEEE_Cigre_DLLInterface.h.
    - .MinValue.<Type> and .MaxValue.<Type>: The same applies to MinValue and MaxValue, which define the minimum and maximum values that the parameters can reach.
- Model_Info: This structure is important because it allows the simulation tool to get the main information of the model.
    - .DLLInterfaceVersion: Indicates the version of the IEEE CIGRE DLL interface used. At the time this documentation is written, the only two values allowed are { 1, 1, 0, 0 } for 1.0.0.0 or { 1, 0, 0, 0 } for 1.1.0.0. as these are the only two versions of IEEE CIGRE DLL that exist. The example uses the latest version so { 1, 1, 0, 0 }.

- .ModelName, .ModelVersion, etc.: All attributes with the prefix "Model" provide various information about the model.
- .FixedStepBaseSampleTime: The time step of the model in seconds. Indicates when the model must be called (when the Outputs function is called by the simulation tool).
- .EMT_RMS_Mode: Indicates the model type (EMT = 1, RMS = 2, EMT & RMS = 3, otherwise: 0).
- .NumInputPorts, .NumOutputPorts and .NumParameters: The number of inputs, outputs and parameters of the model. Must be consistent with MyModelInputs, InputSignals, etc. A vector input (or output) is counted as one input (or output).
- .InputPortsInfo, .OutputPortsInfo and .ParametersInfo: They should refer to the structures defined above.
- .NumIntStates, .NumFloatStates and .NumDoubleStates: Sizes of the IntStates, FloatStates and DoubleStates arrays that are specific to an instance of the model (see the IEEE_Cigre_DLLInterface_Instance structure of IEEE_Cigre_DLLInterface.h)

- Model_CheckParameters: Checks the parameters on the given range. Standard min/max checks should be done by the simulation tool, but other checks can be made here. Like other DLL entry points, it is possible to return a message and to interrupt the simulation if necessary. See part 2.3.1 for the different possible return values of an IEEE CIGRE DLL function.
- Model_Initialize: Initializes the system and resets the internal states (IntStates, FloatStates and DoubleStates arrays). Called at initialization after Model_FirstCall, Model_CheckParameters and Model_PrintInfo (See part 3.1).
- Model_Outputs: Calculates model outputs. Function called at each time step of the model (every FixedStepBaseSampleTime seconds).
- Model_Terminate: Destroys any objects allocated by the model code. Called at the end of the simulation.
- Model_FirstCall: Called almost first at T0 (just after setting parameters to the instance). It is not mandatory to implement this function, it can be useful if actions must be performed upstream, before Model_CheckParameters, before having initialized the output values via SetModelOutputs and before Model_Initialize.
- Model_Iterate: Do not implement at this time as it is not used in current versions of simulation software wrappers. Will be useful for RMS type models.

2.2 Compilation

The compilation must be done in Release mode and in 32 bits:



You need to ensure that compilation in DLL is selected (Project properties ➔ General ➔ Configuration type ➔ Dynamic Library (.dll))

All these options are set by default if you open the "SCRX9.sln" solution.

Note: The C language standard must be C99 or later (For Visual Studio IDE users, it is therefore preferable to use Visual Studio 2015 or later, although the build may work with some versions of Visual Studio 2013).

2.3 Rules and good practice

2.3.1 Messages

Each entry point of the DLL (function called by the simulation tool) can return 3 values:

- IEEE_Cigre_DLLInterface_Return_OK: no message generated
- IEEE_Cigre_DLLInterface_Return_Message: message generated but the simulation continues
- IEEE_Cigre_DLLInterface_Return_Error: message generated and the simulation is interrupted.

Messages must be assigned to the LastGeneralMessage attribute of a model instance.

2.3.2 Use state arrays to store the model's internal variables

State arrays (IntStates, FloatStates and DoubleStates arrays) allow the storage of information that can be transmitted from one time step to another. They are specific to an instance of the model. These arrays can be used to store more information than just integers or real numbers. See part 6.1 to see how to store objects or structures.

3. On the simulation tool side

An important point to make is that the simulation software will take care of allocating the memory for the different instances of the model, managing these instances (storing their address in an array for example), and releasing them at the end of the simulation. It will therefore instantiate an "IEEE_Cigre_DLLInterface_Instance" structure for each IEEE CIGRE DLL block present on the circuit.

It will also allocate the necessary space for the state arrays, according to the sizes indicated in Model_Info (NumIntStates, NumFloatStates, and NumDoubleStates).

3.1 Timing of function calls

The default behavior of each simulation tool must respect this order for the DLL functions calls during the simulation (IEEE CIGRE DLL functions are written in bold):

- **Model_GetInfo**. Called at the beginning of the simulation to get nb inputs, outputs, etc.
- step (called at each time step of the simulation tool):
    - if simulation time is 0:
        - assign parameters from the simulation tool side to model parameters
        - **Model_FirstCall**
        - **Model_CheckParameters**
        - assign initial values to model outputs, provided either through pins or the component mask
        - **Model_Initialize**
    - if the model sampling time has been reached (also true for t0):

        - assign inputs from the simulation tool side to model inputs
        - **Model_Outputs**
- End of the simulation:
    - **Model_Terminate**

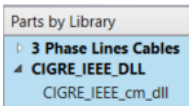
Model_PrintInfo is usually called at t0 but there is no constraint on when it should be called.

Each simulation software, via the IEEE CIGRE DLL import tool, is free to add configuration options that may omit certain function calls or change the order of the calls, if these options are well documented.

The order of function calls is important. For example, if assignments are made to the outputs of the DLL block in the "Model_Initialize" function, initial outputs values provided through pins or the component mask are not taken into account.

3.2 EMTP Import Tool

The tool for importing IEEE CIGRE DLLs is a toolbox, available in the files provided with the example. Copy the folder EMTP\Toolbox\CIGRE_IEEE_DLL into the Toolboxes folder of EMTP (Usually here: “C:\Program Files (x86)\EMTPWorks 4.X.X\Toolboxes”) before opening EMTP. Thus, a new library "CIGRE_IEEE_DLL" will be available in EMTP:



By placing the component on the circuit and double clicking on it to open the form, you will have access to a documentation from the Help tab. The help file gives details of the different configuration options offered by this toolbox.

Once an IEEE CIGRE DLL is selected from the General tab, this device is used to create all the necessary connections (pins) and provide access to some modifiable parameters of the DLL. The interfacing pins (ports) of this device in EMTPWorks are created automatically based on the information contained in the DLL.

3.3 PSCAD Import Tool

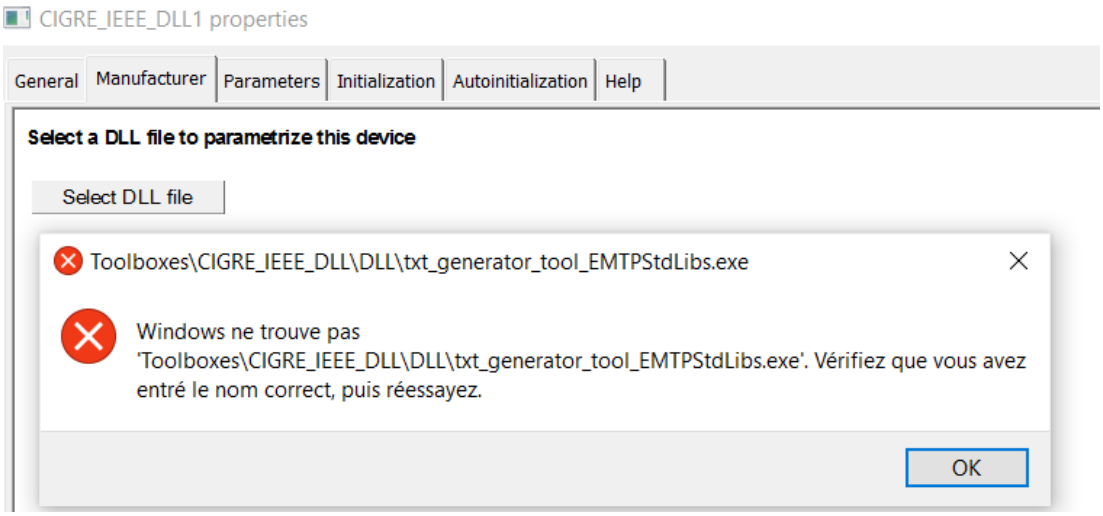
Under development. The basic principle will be a PSCAD component linked to a fortran file that will load and call the DLL.

The aim is to create a tool that, from the IEEE format DLL, can automatically :

1. Draw the block in PSCAD (with correct inputs / outputs)
2. Create the wrapper (fortran file)
3. Create the connection between these different components

4. Current known bugs

4.1 txt_generator_tool_EMTPStdLibs.exe not found in EMTP



This error can occur (the error is not systematic) with the following sequence of actions:

- EMTP is opened for the first time
- A new ".ecf" file is created or an existing one is opened from its interface
- An attempt is made to import the DLL file from the "Manufacturer" tab of a CIGRE_IEEE_cm_dll block

To avoid this, simply save the ".ecf" file into which the DLL will be imported, close EMTP and reopen it by double clicking on the ".ecf" file that has just been saved.

5. Future work

Most of the ideas below were submitted to the members of the Joint IEEE TASS Task Force and CIGRE B4.82 Working Group “USE OF REAL-CODE IN EMT MODELS FOR POWER SYSTEM ANALYSIS”.

- Allow a non-fixed time step for an IEEE CIGRE DLL

For now, an IEEE CIGRE DLL is created for a given time-step but it may be too specific. We could:

  - Handle no time-step dependent model (a simple Gain for example)
  - Handle models that can use the simulation tool time-step
  - Allow several time-steps for a model

- Provide examples in other languages

The manufacturers' codes are not necessarily in C language, it would be good to provide examples of IEEE CIGRE DLL in other languages like Fortran, C#, etc.

For information, the current code can be compiled in C++, provided that a recent version of the C++ standard is used (C++20 or later).

- Test different 32/64-bit combinations

Although EMTP can currently only load 32-bit DLLs, it will be compatible with 64-bit DLLs in the future. Moreover, in terms of performance and memory management, it is often preferable to use 64-bit executables. Therefore, it would be interesting to test :

  - IEEE CIGRE DLL compilation in 32 bits (for EMTP) and 64 bits (for PSCAD configured with a 64 bits Fortran compiler) and compare results
  - Use a 32 bits IEEE CIGRE DLL in PSCAD configured with a 64 bits Fortran compiler (we need to take care of some data conversion, variable memory size to avoid compatibility problems or errors at runtime)

- Think about a snapshot feature (like in PSCAD)

It is theoretically possible to make a snapshot of the model by saving the IntStates, FloatStates, DoubleStates arrays and the parameters that may change during the simulation. The question is whether this snapshot should be made on the DLL side or on the wrapper side of the simulation tool. Modifications in the IEEE CIGRE DLL code may be necessary.

- Extend IEEE CIGRE DLL features to include power systems

For now, IEEE CIGRE DLL is built only for control systems.

6. Appendices

6.1 How to handle model specific structures using state arrays

The provided example is quite simple, most control models contain more complex data structures (big C struct for example). One way to store this data and make it available from one time-step to another is to allocate space for the data and store its address in the IntStates array. Here are some guidelines:

1. Define the structure somewhere, for example in a file “MyModel.h”

```
typedef struct _MyStruct
{
    double ex1;
    int ex2;
} MyStruct;
```

2. Increment Model_Info.NumIntStates to be able to store the structure address

3. Allocate your structure once (for example in Model_Initialize), initialize it, and store the address:

```
// Allocate memory of my own structure and initialize values
 MyStruct* myStruct = (MyStruct*)malloc(sizeof(MyStruct));
 if (myStruct) {  // Check if allocation is OK
     myStruct->ex1 = 4.2;
     myStruct->ex2 = 9;
```

```
    }

    instance->IntStates[0] = myStruct; // Store address of the structure in IntStates
array (here, IntStates size is 1)
```

4. Retrieve this structure when you need it, thanks to the stored address

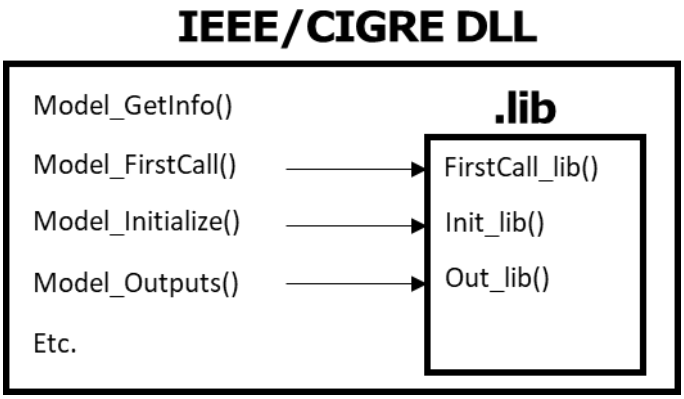
```
// Retrieve my own structure from address at IntStates[0]
 MyStruct* myStruct = instance->IntStates[0];
 if (myStruct) {  // Make sure memory had been allocated
     // Edit values
     myStruct->ex1 += 1;
     myStruct->ex2 += 2;
 }
```

5. At the end of the simulation, release the memory in Model_Terminate

```
MyStruct* myStruct = instance->IntStates[0];
 // Clear out memory
 if (myStruct) {
     free(myStruct);
     myStruct = NULL;
 }
```

6.2 Making an IEEE CIGRE DLL from an already compiled control model

If you already have a compiled control model in .lib or .obj format, it is possible to use it to create a DLL in IEEE CIGRE format. Just add an overlay with the entry points of the IEEE DLL, as shown in the figure below:



Here are the actions to perform in Visual Studio:

- Open an IEEE CIGRE DLL project in Visual Studio (you can use the provided example and clean the parts corresponding to the SCRX model)
- Right click on your project in Visual Studio ➔ Properties ➔ Linker ➔ Inputs ➔ Additional Dependencies : Indicate the path to your compiled model
- Call a function of your .lib or .obj file from one of your .c files in your project.
- Build the solution. Compiling and linking should work.

- After building your model, the created DLL contains the code of the lib file. You will get a standalone DLL.

Depending on the type of model, other actions may be required but the general principle remains the same.

Note: If your already compiled model is in DLL format but not compliant with the IEEE CIGRE DLL format, it is also possible to create the IEEE CIGRE interface. For example, from the IEEE CIGRE DLL solution, you can link explicitly to your DLL model by following these guidelines: https://learn.microsoft.com/en-us/cpp/build/linking-an-executable-to-a-dll?view=msvc-170. It will then be necessary to provide the 2 DLLs because they remain separate (one is not included in the other at the building time).