Guidelines for Use of Standardized DLL Interfaces for Real-Code in Simulation Models

Garth Irwin, *Member, IEEE*, Deepak Ramasubramanian, *Senior Member, IEEE*, and IEEE Task Force "Use of Real-Code in EMT Models for Power System Analysis", TASS/AMPS/PES, *IEEE*

This paper presents a dynamic link library (DLL) based modeling method to standardize simulation tool interfaces to real-code for digital controllers in power system simulation tools. Such a standardization of interfaces can allow for seamless transfer of models across simulation software and also allow for improved guarantees of representation of firmware control in simulation models.

Keywords: Real-Code, Dynamic Link Library, EMT, TS

I. Introduction

The IEEE/CIGRE DLL Modeling Method [1] defines a modeling format that allows power system controls and protection models (usually the identical code developed by a manufacturer and running in field hardware – i.e., "real-code") to be used in any power system simulation tool, including Electro-Magnetic Transients (EMT) and Transient Stability (TS) RMS simulation programs.

It should be emphasized that although DLL is used in the name, this method is not limited to only MS-Windows DLLs. The same method and format applies equally to other formats such as Linux .so shared object file formats.

This paper describes the setup of the wrappers that are needed to be represented in simulation software to allow interface with the DLL. Further, the structure of variables to be used and the definitions of the individual functions within the wrapper are highlighted. This work is an extension of earlier work done in IEC-61400-27-1, although alternate starting points (such as the FMI method - https://fmi-standard.org/) were also considered by the task force. Future revisions are to be developed under IEEE Standards group P3597.

II. General Principle and Benefits

An IEEE/CIGRE DLL controller model is inherently black-boxed (i.e. hides confidential information and source code). It is designed for modeling of control and protection code (i.e. controllers associated with inverter-based resources, voltage source converter (VSC), static var compensator (SVC), static synchronous compensator (STATCOM) and control for related converters, relays and other field controllers or protection devices), often using "real-code" (firmware code running in the field hardware) and calling this directly from simulation tools. Other applications include generic model writing (so the same code can be used in many different simulation programs).

It should be noted that the focus is on control and protection systems, and not electrical modeling. The use of the word "model" in this guide relates to only control and protection functions, and additional tool specific electrical system modeling is separate and not addressed in this guide.

The main advantages of the IEEE/CIGRE DLL Modeling method are:

- It is independent from a simulation tool and its solver. This modeling provides a standard target for simulation programs and can run in any EMT and TS programs.
- The modeling uses DLLs (Dynamic Link Libraries) (as opposed to the use of .lib or .obj files that use static linking).
- A model in the IEEE/CIGRE DLL format is black-boxed and therefore protects manufacturers' intellectual property, which is a key point for model exchange.
- Model accuracy – since the models often call the identical code that is running in field controllers (i.e., real-code), the model differential/control equations are solved in a similar manner as per the field code (including the same sample rate).
- An IEEE/CIGRE DLL model connects two distinct worlds: the manufacturers of these devices (who write the code and design the controller) and the end-user/utility (who is responsible for performing engineering and operational studies).
- Simulation tool developers have great freedom to implement significant changes in their program (including model/controller interface methods, COMMON block, main algorithm structures, call statements, variable types etc.) without making models incompatible.
- The DLL created is self-documenting (i.e., contains all information needed to interface to any program) so it can be used in all current or future simulation tools (without changes to the DLL models).

This paper summarizes the efforts of a joint task force:

- IEEE (Use of Real-Code in EMT Models for Power System Analysis – organized under the TASS Working Group (Transient Analysis and Simulation) and AMPS Committee (Analytical Methods for Power Systems)
- Cigre B4.82 (Guidelines for Use of Real-Code in EMT Models for HVDC, FACTS and Inverter based generators in Power Systems Analysis)

979-8-3315-8138-1

|The steps in the IEEE/CIGRE DLL Modeling Method are:

- Model Writers (often controller/device OEMs – Original Equipment Manufacturers) add a small code wrapper around their source code (then compile their full code set to a DLL). This wrapper adds static information about the model (i.e., documenting the inputs, outputs, parameters, units, default values, sampling time step etc.), but also standardizes main interfaces to call the model code during initialization and at each sample time.
- Simulation Tool Developers develop/release a DLL import tool/feature. This tool initially reads static information in the DLL and creates any interfaces of the model to a given simulation tool.
- End Users – they obtain the DLL from the Model Writer, run the DLL import feature (for their chosen simulation tool) to create interfaces to the model, connect the controller model to other electrical devices, and can then run the overall simulation.

III. Specification of the IEEE/CIGRE DLL Method V1.1

A. Overview of IEEE/CIGRE interface, .h headers files and .c wrapper code

An IEEE/CIGRE DLL Format has been designed in the C language but can be written, interfaced, or translated into another language, using equivalents for types, numerations, and structures. Models can be written directly in Fortran (for example) or other languages. We will refer to the C language in the remainder of this document (as most manufacturers use C to develop real-code models in the field firmware and many tools are C-based).

The current version of the interface is 1.1 which uses structures for all inputs, outputs, and parameters, and allows other data types (unlike in previous versions of this interface that only allowed the "double" data type).

A model in the IEEE/CIGRE format includes a .c wrapper, which includes two header files:

- IEEE_Cigre_DLLInterface.h
- IEEE_Cigre_DLLInterface_types.h

These files are common to all IEEE/CIGRE models and must not be modified. These files can be downloaded from Cigre or from [2], although copyrights for both Cigre and IEEE apply.

B. IEEE_Cigre_DLLInterface_Model_Info Structure:

This is a passive/static structure defined in the .c wrapper code – it defines basic information/documentation about the model, including a list of all inputs, outputs and parameters used in the model, main simulation tool properties (such as the sample time at which the code should be called), version and other model information.

The Model_GetInfo() routine can be called and will return the full set of static information in the IEEE_Cigre_DLLInterface_Model_Info structure. This method will be called by DLL import tools (to create interfaces to the DLL model for a given simulation program) but also can be called within a model during dynamic runs (for example to get the sample step size).

C. IEEE_Cigre_DLLInterface_Instance Structure:

This is a structure that is used dynamically to pass all information between the simulation tool interface and the DLL model dynamically during a simulation.

The member variables in the IEEE_Cigre_DLLInterface_Instance structure include:

`ExternalInputs`	Input signals array
`ExternalOutputs`	Output signals array
`Parameters`	Parameters array
`Time`	Current simulation time
`SimTool_EMT_RMS_Mode`	Mode: EMT = 1, RMS = 2
`LastErrorMessage`	Error string pointer
`LastGeneralMessage`	General message
`IntStates`	Int State array
`FloatStates`	Float State array
`DoubleStates`	Double State array

These variables are used dynamically in simulation. Typically, in a normal simulation tool execution sequence, the main simulation tool first determines if this is the correct sampling instant, and then defines:

- ExternalInputs
- Parameters
- Time
- SimTool_EMT_RMS_Mode (i.e., if the simulation tool is of type EMT or RMS)

The simulation tool interface then calls the Model_Outputs functions to pass these inputs to the .c wrapper code. The .c wrapper code then extracts the inputs and parameters, passes them to the model code, which solves the outputs (and state variables).

After calling the internal DLL code for this specific sampling instant, the .c wrapper code then:

- Saves state variables
- Defines ExternalOutputs
- Optionally return any error or warning messages.
- Returns to the main simulation program

Methods using the IEEE_Cigre_DLLInterface_Instance Structure:

The subroutines which pass and/or use the IEEE_Cigre_DLLInterface_Instance structure are normally called during each dynamic simulation – these include:

Model_FirstCall:

This subroutine is called once at the beginning of a simulation, e.g., for memory allocation inside the DLL. Note the caller (simulation tool) allocates state variable memory.

Model_PrintInfo:
This subroutine is called once at the beginning of a simulation – it allows models to write key information about inputs or parameters to the main simulation tool.
Model_CheckParameters:
This subroutine is called once at the beginning of a simulation (and in some tools may be called every time a variable parameter changes its value). It allows the model writer the opportunity to check input parameters with a more complex algorithm (as opposed to a simple min/max which is already part of the parameter dialog).

Model_Initialize:
This subroutine gets called in the first time-step (after Model_FirstCall). The desired initial output signal values (often from powerflow or other steady-state electrical solvers) must be applied to the ExternalOutputs structure before calling this subroutine, as they are used to initialize the model.

It is the responsibility of the Model Writer to specify this initial condition information, and to back-calculate and save all state variables (IntStates, FloatStates and DoubleStates arrays) in the model code/algorithm. This is particularly important for TS programs which are expected to "flat-start", whereas EMT programs often can ramp from zero initial conditions.

Model_Outputs:
This is the main subroutine called from the simulation tool, and the entry point into the DLL model code. The simulation tool is responsible to call this routine at the defined sample rate/time FixedStepBaseSampleTime seconds (so the code gets called at regular time intervals that may differ from the simulation time step used in the simulation tool). The simulation tool defines the Input and Parameter structures, passes these in the call arguments to Model_Outputs – the code algorithm solves the model equations for this sample time, saves State variables, and updates the Outputs structure with the results of the computations.

Model_Iterate:
Future support of model convergence iteration at each time step. This is not yet defined and may not be supported in some simulation tools.

Model_Terminate:
This subroutine is called once at the end of a simulation, e.g., close files, unload the DLL, release DLL memory, etc.

Return Values from all Functions:
Each function of the DLL (entry points called by the simulation tool) must return one of the following three possible values:

- IEEE_Cigre_DLLInterface_Return_OK: everything went well, and no message is generated
- IEEE_Cigre_DLLInterface_Return_Message: message is generated but the simulation continues. It can be an informational or a warning message.
- IEEE_Cigre_DLLInterface_Return_Error: error message is generated, and the simulation is interrupted.

IV. State Variables

A state variable is any model information that needs to be saved from previous time steps (or defined in initial conditions). This could include an integrator, proportional-integral-derivative (PID) controller, or most other differential equation dynamic control functions. It also includes "extra" state variables for all output signals and a counter due to sampling (see below).

It is important that the DLL model user-written code defines and groups every state variable and interfaces them via this IEEE/CIGRE method. This allows multiple instances of the DLL model to be used and allows the use of "snapshots" so the model at any point in time can be saved and re-started from this same time.

V. Guidelines for Model Writers

The basic steps to write a model are:

1. Collect all source code for the model (noting that this source code is kept confidential and should not need to be released to end users).
2. Write/add a small .c wrapper (so your code compiles to a DLL adhering to the IEEE/CIGRE DLL format).
3. Add initialization code (if the model is to be used with TS programs that require flat start, or if there are slow time constants and the EMT model requires a long time to reach steady state).
4. Compile all code to a DLL.

It may be tempting to completely avoid the use of state variable storage – in this case the model will "run", but multiple instances and snapshots may not work.

VI. Guidelines for End Users

To use a model in a simulation, the end user should:

- Obtain the model (in DLL form)
- Import the DLL into their simulation tool of choice (using tools provided in each tool)
- Connect the model to other electrical portions of the model (many tools are graphical, but some are data file driven)
- Run the simulation

The model should run in any version or tool that supports the DLL interface - this alleviates the need for OEMs to recompile models (to support newer versions of simulation tools or compilers) and provides future compatibility without the need to request that OEMs provide source code for models.

VII. Simulation Tool Developer Manual

A simulation tool developer will develop a DLL Import function which will:

- Call the Model_Info method to get information that defines the Inputs, Outputs, Parameters, Sample time, state variables etc.

- Create any interface methods of code to dynamically call the model at the specified sampling rate (which may differ from the main simulation time step).

Many tools are graphic based, so the importer should create a graphical object which has input and output signals to make connections between the DLL controller and other control functions (or electrical measurements or inputs to electrical models). The graphical component may also have parameters (along with min/max error checks, etc.).
The version number may be read (in the initial Model_GetInfo call) to conditionally support future features or changes.
The interface code (created by the DLL import tool for a given simulation program) is also responsible for:

- Creating a sample/hold loop (so the Model_Outputs code gets called at the sampling rate defined inside the DLL).
- Allocate state variable storage (DoubleState, IntState etc.).
- Add state variables for all Output signals and storage of a sampling counter.

A. *Timing/Order of Function Calls*

A simplified flow-chart of the simulation tool time sequence and calling order of the DLL routines is shown in Fig. 1.

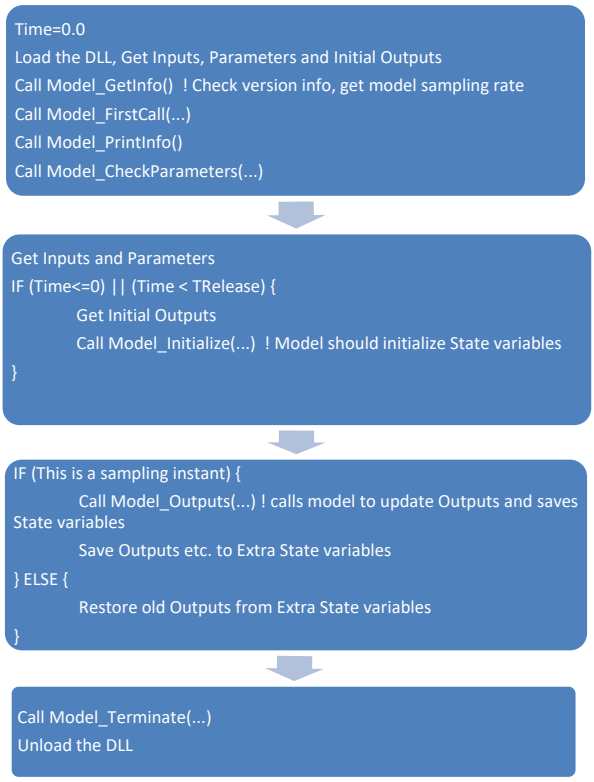

Fig. 1. Simplified Time Sequence and Calling Order of IEEE/CIGRE DLL Functions if using a snapshot is not used

VIII. Next Steps and Future Work

This IEEE/CIGRE DLL method has resolved many of the complexities and problems in power system simulation models with the use of real-code (including model accuracy, compatibility problems due to statically linked libraries, use of the same models in numerous tools etc.). The method has reached a point where it is commercially available and used in industry but will likely continue to evolve. The IEEE/CIGRE method is already supported by many simulation tools, has been used by numerous manufacturers, and is referenced by TSOs. A working example has been published [3].

Finally, it should be noted that the IEEE/CIGRE method is not (yet) formally a standard – work is ongoing in IEEE standards group P3597 [4].

IX. References

1. Cigre JWG B4.82/IEEE, "Guidelines for use of real-code in EMT models for HVDC, FACTs and inverter based generators in power systems analysis," TB 958, March 2025.
2. IEEE/Cigre DLL Interface Repository: https://www.electranix.com/ieee-cigre-dll-interface/
3. EPRI, "Code Based Generic Inverter Based Resource Model." Accessed: Sept. 25, 2024. [Online]. Available: https://www.epri.com/research/products/3002028322.
4. IEEE Standards Association, "IEEE P3597, Standard for Interfacing of Controls and Protection to Power System Simulation Tools." Accessed: July 08, 2025. [Online]. Available: https://standards.ieee.org/ieee/3597/12053/

X. Task Force Members

The task force members include simulation tool developers, consultants, university/academic experts, equipment manufacturers and utility experts. The members from IEEE (noting there are additional members from Cigre) include:

Alejandro Duque
Patrick Alizon
Chaminda Amarasinghe
Jean Belanger
Sigrid Bolik
Casey Briscoe
Christopher Burge
Tamojit Chakraborty
Ying Chen
Yunzhi Cheng
Suman Debnath
Ali Dehkordi
Jeewantha DeSilva
Graham Dudgeon
Iñigo Urra Eraso
Omar Faruque
Alex Flueck
Miguel Cervantes Martinez
Slobodan Matic
Tom Mcdermott
Eric Mellick
Parag Mitra
Allen Morinec
Kumara Mudunkotwa
Om Nayak
Christian Neumann
Taku Noda
Mohamed Osman
Slobodan Pajic
Chathura Patabandi
Krishnat Patil
Dinesh Pattabiraman
Paul Peeling
Jaime Peralta

Jens Fortmann
Tim Freiberg
Janath Geeganage
Esmaeil Ghahremani
Pramod Ghimire
Ali Goharrizi
Ani Gole
Pablo Gomez
Anupam Gopal
Jigar Gorasiya
Carlos Grande-Moran
Henry Gras
James Guest
Rantin Hadidi
Moritz Hildebrandt
Fred Huang
Roni Irnawan
Geona Jeong
Amir Reaa Kazemi
Antoine Keirsbuick
Kah Leong Koo
Neeraj Lal
Vincent Lapointe
Fernando Libonati
Bryan Lieblick
Xi Lin
Hanchao Liu
Pradeep Mahadanaarachehi
Jean Mahseredjian
Andrea Marti
Catherine Marti
Jose Marti
Sidharth Rajagopalan
Prakash Ranganathan
Praveen Reddy
Luke Robinson
Jesus Morales Roderiguez
Jeff Roesch
Jonathan Rose
Chris Rowe
Johannes Ruess
Shammya Saha
Roosa-Maria Sallinen
John Schmall
Jayapalan Senthil
Miaolei Shao
Ranjan Sharma
Andre Silva
Anton Stepanov
Ehsan Tara
Francisco Moura Teixeira
Aung Thant
Robert Thornton-Jones
Victor M. Huerta Urieta
Ramin Vakili
Nyuk-Min Vong
Ulrich Wahner
Liwei Wang
Xiaoyu Wang
Eric Wu
Sangwon Yun
Yi Zhang
Yichao Zhang

979-8-3315-8138-1