EMTP

DLL Programming

info@emtp.com

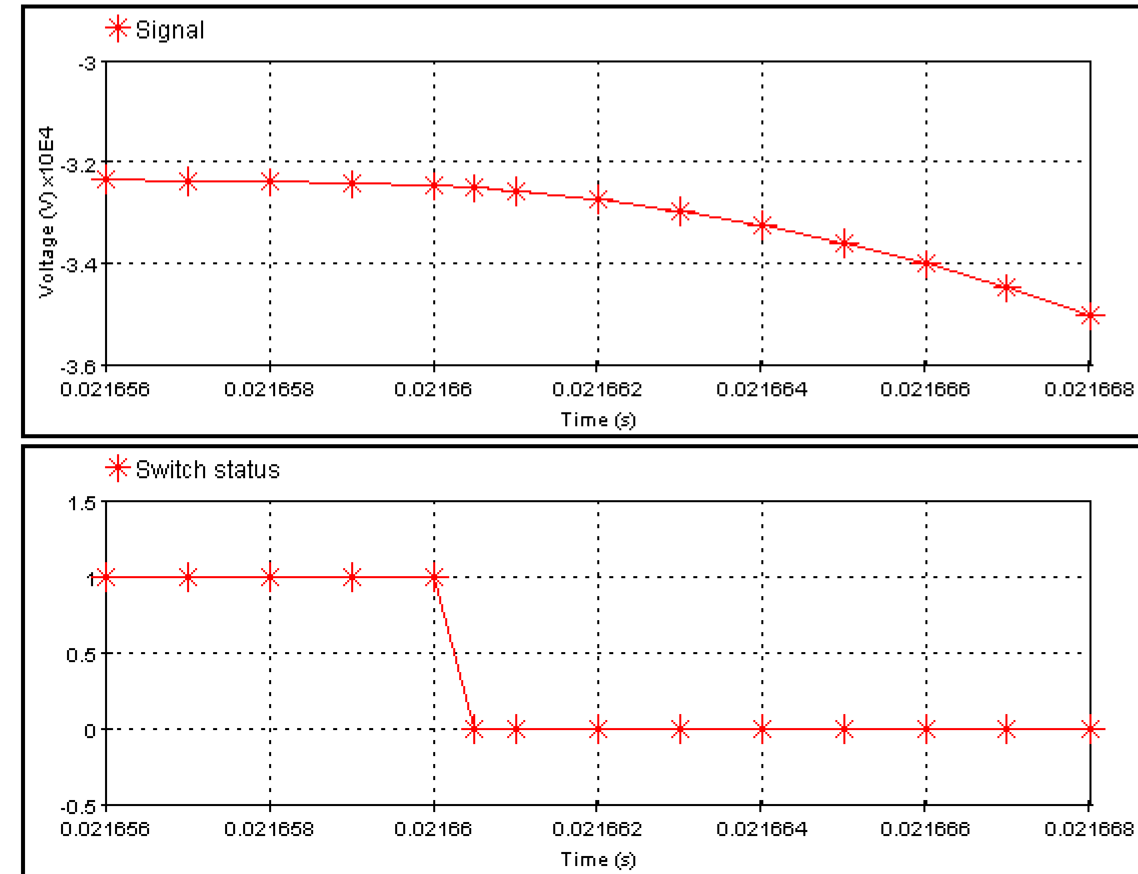
EMTP Main device attributes

- See documentation for detailed listing
- Name
  - Name.Prefix
- **Value**, Value1, Value2
  - Visual information on the device
- Device EMTP data attributes (transmitted to EMTP® engine)
  - ParamsA
  - ParamsB
  - ParamsC
  - ModelData => used to transmit parameters to DLL
- **FormData**: used for saving device data not sent to EMTP® engine
- **Script.Open.Dev**: device double-click method
- **Script.Info.Dev**: Names the script called when the user right-clicks on the device and selects "Extras/Summary".

EMTP
The reference for power systems transients

EMTP solver

- EMTP® has a quasi-constant time step solver.
- The standard trapezoidal method (trap) may be changed into the halved time-step Backward-Euler method (EBA) to account for discontinuities and eliminate numerical oscillations (instabilities).
- The Power circuit is solved first, then the control circuit is solved. Therefore, in case of loop between the two circuits, one time-step delay is expected.



EMTP
The reference for power systems transients

DLL programming - Introduction

- The DLL (Dynamic Link Library) function is designed to allow EMTP® users to develop advanced program model modules and interface them directly and intimately with the EMTP® engine.

- To build DLLs, a compiler is required.

- EMTP® DLLs can be written in almost any language. Fortran 2003 and C++ interfaces are provided.

DLL programming – Interfacing method

- The DLL file must be placed in a valid location to become automatically located by the EMTP® engine (see 2 - DLL setup step by step).
- The extension .dll is assumed.
- The DLL file must contain a set of callable methods (functions) recognized in the EMTP® engine (see DLL setup step by step and DLL documentation).
- Any number of user-programmed DLLs can be used in a simulation case and each DLL can be used more than once.
- If a user error is experienced, the application must be killed manually using the Windows Task Manager.

DLL setup step by step

To illustrate how to create a DLL, the EMTP example will be used:
EMTPWorks X.XX\Examples\ApplicationCases\DLL\ControlsOnlyInNetworkDLL\dll

The DLL will have 2 control inputs and 1 control output.

The output will be: $o1 = i1 + K * i2$ where K is a parameter.

The DLL name is assumed to be fdll_control.dll and is located in the debug folder generated by the compiler, in the same folder of the design. Therefore, the relative path of the DLL with the EMTP design is debug/ fdll_control.dll.

EMTP
The reference for power systems transients

Warning

the DLL of this example is a Power DLL with a control section. The DLL introduces an extra time-step delay which may cause numerical or precision issues for fast dynamic controls such as converter control.

For faster control algorithms like converter controllers, it is recommended to use a control DLL such as in example:
EMTPWorks X.XX\Examples\ApplicationCases\DLL\ControlDLL

This kind of DLL participates in the perturbation algorithm used by EMTP® to solve algebraic loops, therefore the DLL may be called several times during a time step. Most of the time, this option is not necessary to run a real code. The *IF(procedure_type .NE. 2)* may be used to avoid calling the real code several times.

EMTP
The reference for power systems transients

DLL setup step by step – in EMTP®

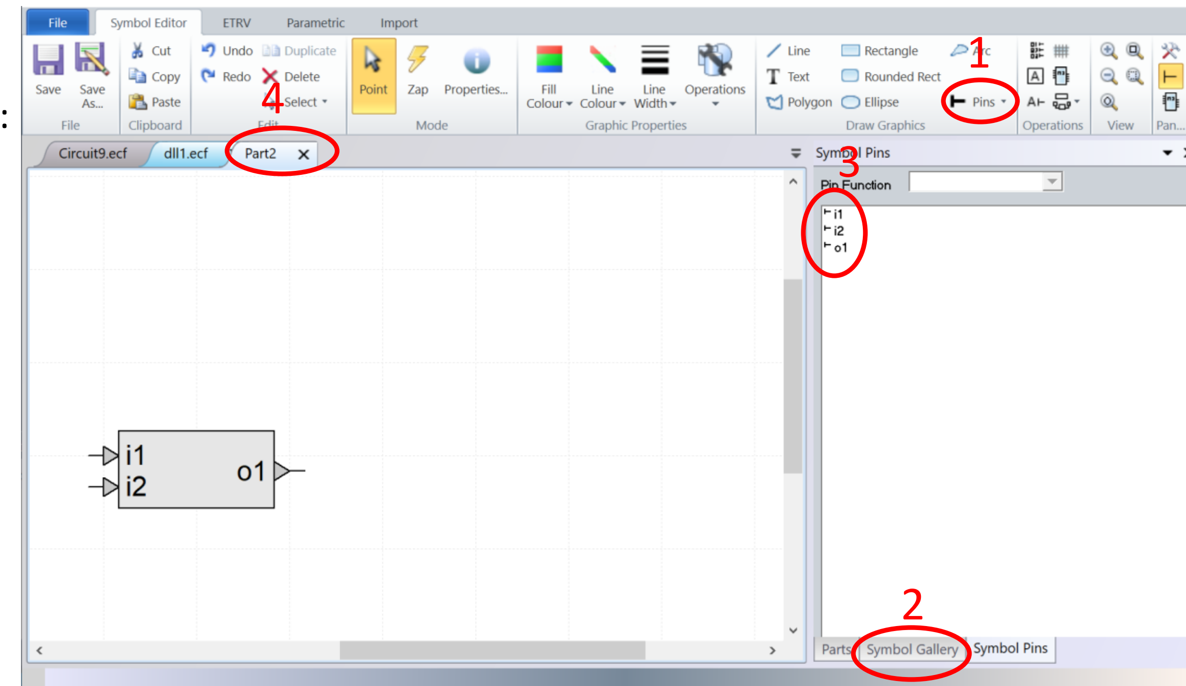
In this section, we will describe what should be done on the EMTP® side

EMTP
The reference for power systems transients

DLL setup step by step – in EMTP®

1 - Create the symbol:

- Options/New Part => the Part editor is created
- Draw the symbol using the Draw graphics section
- Add the pins using the Draw graphics section (1). Any time a pin is placed:
  - Double click on it
  - Write its name
  - Select its function. Input and output are for control
  - The convention is to add directional triangles to indicate inputs and outputs. The triangles can be found in the Symbol Gallery on the bottom right of the window (2).
  - The pin functions can be seen and/or modified from the Symbol Pins tab (3). If a pin is to be deleted, it must be done from this tab.
  - Pins are numbered according to the order of appearance from top down
- Close the Part tab (4). A saving message will open. Save the part. To save:
  - Create a New Lib (example: myLib.clf)
  - Select the part name (example: Part2)
  - Save
- Open a design. MyLib is available in the library list.
- Drag and drop the device (Part2 in the example)



EMTP
The reference for power systems transients

DLL setup step by step – in EMTP®

2 – Setup the attributes.

- Right click on the device and go to Attributes
- In Show Fields, select All
- Setup the attributes:
    - <u>Part</u>: Must start by '*DLL*' (ex: *DLL_test*)
    - <u>ParamsA</u>: Use comma-separated format to define, in this order:
        - Number of power nodes
        - Number of voltage-defined devices
        - Number of current sources
        - Number of nonlinear nodes
        - Number of control inputs
        - Number of control outputs
        - Relative_path_flag:* 0: the DLL is searched with absolute path
          * 1: the DLL is searched with relative path
          * 2: the DLL is at the location specified by the "DLL Options" device.
          * 3:the DLL is searched in the Toolbox folder in C:\Program Files (x86)\EMTPWorks 4.0\Toolboxes.
          This Toolbox folder contains the file Toolbox.ini which specifies searched DLL folders for Toolboxes.

(ex: ParamsA is *0,0,0,0,2,1,1,* for a control DLL of 2 inputs and 1 output)

- <u>ModelData:</u> The DLL path is specified on the first line following the rule of Relative_path_flag. The .dll extension is assumed.

The following lines are used to transmit the parameter.

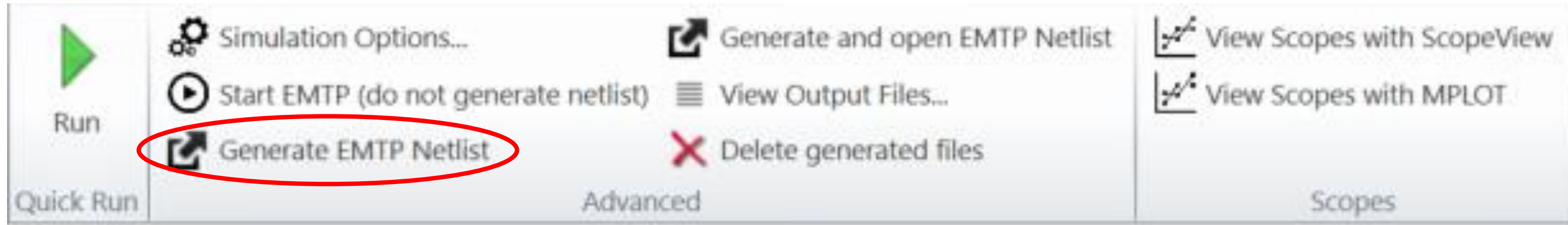
(ex: ModelData: *debug/fdll_control,* K being defined in a parent device ModelData.)
#K#

EMTP
The reference for power systems transients

DLL setup step by step – in EMTP®

3 – Netlist generation for debugging.

- At this point, the device is ready to be simulated.
- To start the simulation from the compiler in debug mode, the netlist must be generated.



DLL setup step by step – The code

In this section, we will define which subroutines are important for control DLL.

The work is assumed to start from the fdll_control.sln project.

The routines which are not described in this section should not be changed.

EMTP
The reference for power systems transients

DLL setup step by step – The code

1 – Defined the module

- Defined variables and parameters accessible throughout the simulation
- The module is called at the beginning of each subroutine with USE FDLL_DATA
- Only the part between !*Modify tags *! should be changed.
- krealhp or double precision should be used for variables

(In the example, only K is defined in the module)

```
MODULE FDLL_DATA
    USE sizelimits
    USE default_precision
    USE simulation_data
    USE variable


    TYPE, PUBLIC :: data_holder
        CHARACTER(LEN=max_len_comp_id) :: myname  !unique name for this device coming from EMTPWorks
        INTEGER                        :: idev    !unique device number from the list of all DLL type devices

        !* Modify here *!
        REAL(krealhp)                  :: K       !gain: output = i1 + K*i2
        !* End Modify here *!

        INTEGER, POINTER, DIMENSION(:) :: power_signal_nodes !will save the actual global node numbers for this device
        INTEGER, POINTER, DIMENSION(:) :: control_valindex   !will save control signal index
        LOGICAL :: Error = .FALSE.   !error flag
        TYPE(data_holder), POINTER     :: next => NULL()
   END TYPE
   TYPE(data_holder), POINTER, PUBLIC :: myDLL=>NULL()
   TYPE(data_holder), POINTER, PUBLIC :: myDLL_first=>NULL()

   LOGICAL, PUBLIC :: EXISTENCE=.FALSE.          !indicates if this device exists in EMTPWorks
   INTEGER, PUBLIC :: Total_number_of_devices=0; !keep the local count

END MODULE
```

EMTP
The reference for power systems transients

DLL setup step by step – The code

1 – DLL_INITIALIZE_NEW

- First subroutine called
- Use to transmit the parameters from ModelData to the code
- Parameter verification or pre-processing can be done in this function

- Data_Section is an array which elements are the ModelData lines, starting after the EMTP® DLL path definition
- The function READ(Data_Section(nlines),*,ERR=999,END=999) myDLL%K will read the Data_Section element number nlines and assign its value to myDLL%K.
- Several parameters can be stored on one ModelData line, separated by a space. In this case, they are called as follow:
  READ(Data_Section(nlines),*,ERR=999,END=999) myDLL%parameter1, myDLL%parameter2, myDLL%parameter3…

Warnings: There should not be more than 100 characters on one ModelData line, except from the EMTP® DLL path.
If a long parameter, like the path of a second DLL must be transmitted, see section DLL interface

- If there is an error with the READ function, the cursor jumps to line 999 which defines myDLL%Error=.TRUE.

- During the pre-processing, an error function is available. This function displays a message in the EMTP® console. If the error is critical, the function stops the simulation.

For example, assuming K must be greater than 0, this pre-processing can be done:

```
IF( myDLL%K == 0 ) THEN !*Data checking here
     !Send a device error, the message is the second argument
     !The third argument is the state of the error: TRUE means critical, FALSE is more like a Warning
     CALL device_error_(myDLL%myname,'Data is wrong',.TRUE.)
ENDIF
```

DLL setup step by step – The code

2 - DLL_LOAD_OBSERVABLES_T0

- At the beginning of the simulation, the power circuit is solved. Some controllable power devices might need control input values. This subroutine is here to define the DLL control output values for the resolution of the first power circuit time-step .
- The outputs are set by the array Returned_obs_array.

(In this example, there is 1 output, initialized at zero: Returned_obs_array(1)= zero)

EMTP
The reference for power systems transients

DLL setup step by step – The code

3 – DLL_INIT_AT_T0, DLL_EBA_INIT_AT_T0

- These functions are called right after DLL_LOAD_OBSERVABLES_T0
- For standard simulation options, where the Numerical Integration Method is Trapezoidal and Backward Euler, DLL_EBA_INIT_AT_T0 is called.
- If the Numerical Integration Method is set to Trapezoidal (trap), DLL_INIT_AT_T0 is called
- If the Numerical Integration Method is set to Backward Euler (EBA), DLL_EBA_INIT_AT_T0 is called

This differentiation is important because the time-step following will not be the same. For DLL_EBA_INIT_AT_T0, it is a half time-step, for DLL_INIT_AT_T0, a full time step.

For example, for a timer starting at the first time-step:

- DLL_INIT_AT_T0: myDLL%Timer = SimTime%Dt

- DLL_EBA_INIT_AT_T0 : myDLL%Timer = SimTime%Dton2

EMTP
The reference for power systems transients

DLL setup step by step – The code

4 – DLL_UPDATE_AT_T, DLL_TRAPTOEBA_UPDATE_AT_T, DLL_EBA_UPDATE_AT_T, DLL_EBATOTRAP_UPDATE_AT_T

- These functions are called at each time-step , respectively, for trap time-steps, time-steps from trap to EBA, EBA time-steps and EBA to trap time-steps
- In this function, the input values can be retrieved. For input *i*:

```
REAL(krealhp) :: Input_i
        :
        :
Input_i = SimData%Psys_control_signal_values(myDLL%control_valindex(i))
```

This differentiation is important because the time-step following will not be the same. For DLL_TRAPTOEBA_UPDATE_AT_T and DLL_EBA_UPDATE_AT_T, it is a half time-step, for DLL_UPDATE_AT_T and DLL_EBATOTRAP_UPDATE_AT_T, a full time-step.

For example, for the timer:

- DLL_UPDATE_AT_T and DLL_EBATOTRAP_UPDATE_AT_T : `myDLL%Timer = myDLL%Timer + SimTime%Dt`
- DLL_TRAPTOEBA_UPDATE_AT_T and DLL_EBA_UPDATE_AT_T : `myDLL%Timer = myDLL%Timer + SimTime%Dton2`

EMTP
The reference for power systems transients

DLL setup step by step – The code

5 – DLL_LOAD_OBSERVABLES

- This function is called at each time-step (trap or EBA)
- Setup the control outputs.

Example

```
USE FDLL_DATA

   !DEC$ ATTRIBUTES DLLEXPORT:: DLL_LOAD_OBSERVABLES
   INTEGER :: idev
   REAL(krealhp), INTENT(OUT)  :: Returned_obs_array(*) !return all observable variables
   REAL(krealhp) :: i1, i2

   !No need to perform internal testing, only devices with observables are called
   CALL find_mydev(idev); !found the right device version with observables.
   !* Modify here *!
   !Getting input values
   i1 = SimData%Psys_control_signal_values(myDLL%control_valindex(1))
   i2 = SimData%Psys_control_signal_values(myDLL%control_valindex(2))

   Returned_obs_array(1) = i1 + myDLL%K * i2
```


EMTP
The reference for power systems transients

DLL setup step by step – The code

5 – DLL_SAVE_ME, DLL_LOAD_ME

- These functions are called during statistical analysis when the time of dice roll (snapshot) is used. They are used to save and retrieve required variables between simulations. The snapshot can be done at any point in time. The DLL must save all time-dependent data to be able to restart the simulation correctly.

Use CALL save_data_into_bin_(myDLL%Variable) in DLL_SAVE_ME and ALL load_data_from_bin_(myDLL% Variable) in DLL_LOAD_ME for each time-dependent variable.

DLL debugging

In this section, we demonstrate how to debug the DLL.

It is assumed that the DLL device is built in EMTP® and the netlist is generated.

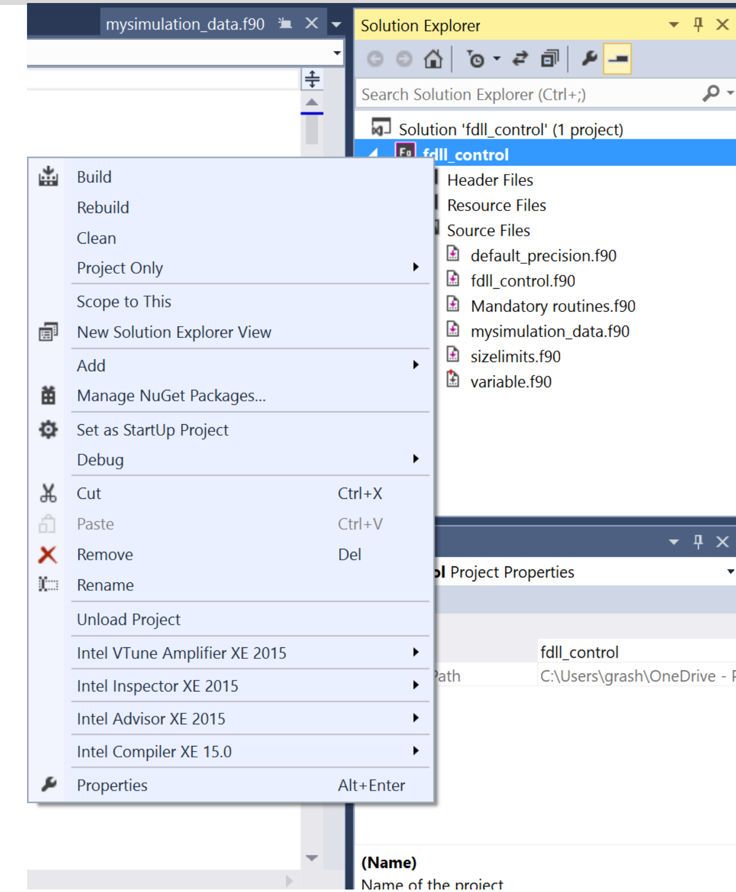
EMTP
The reference for power systems transients

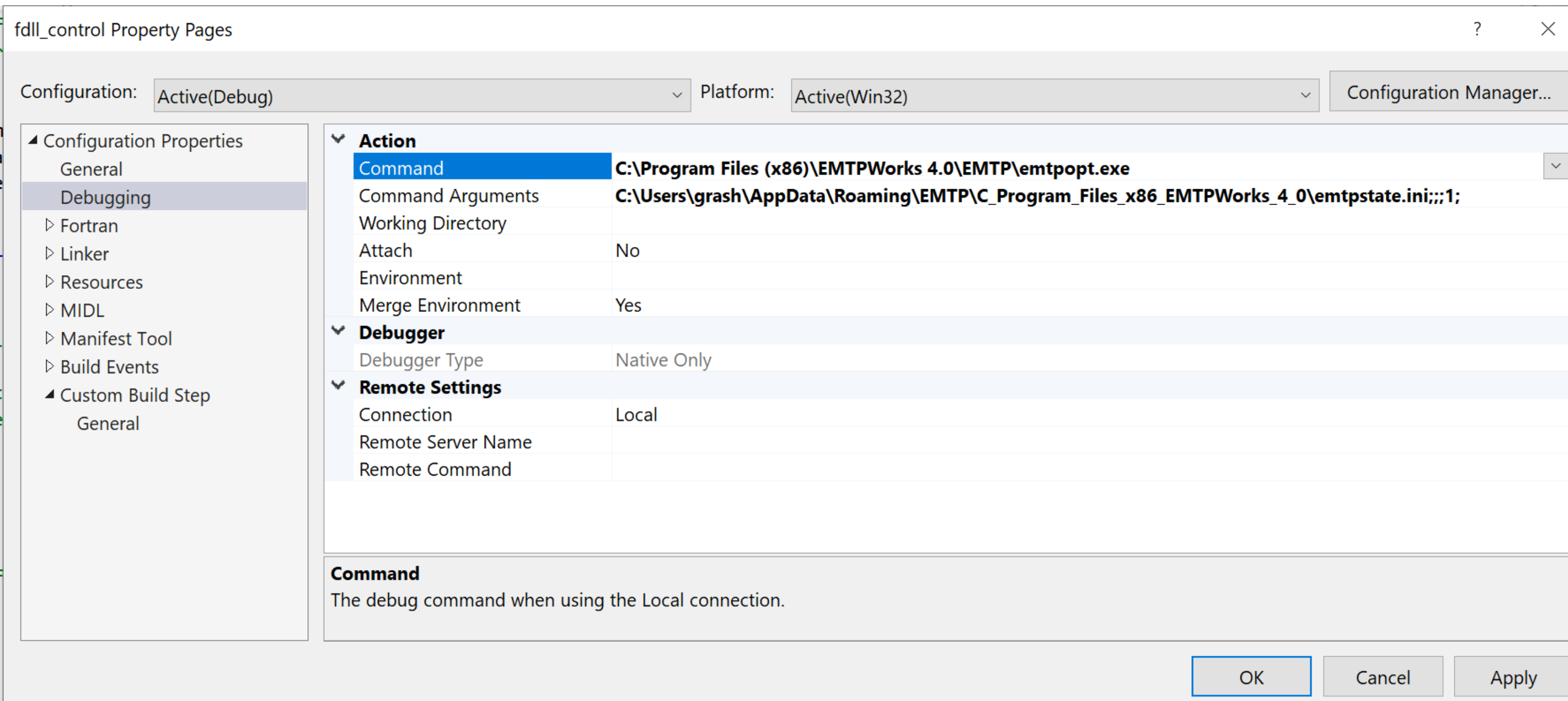
DLL debugging

- In solution explorer, right click on fdll_control and go to Properties.

- In the debug configuration, setup:

Command: C:\Program Files (x86)\EMTPWorks 4.0\EMTP\emtpopt.exe
Command arguments: C:\Users\grash\AppData\Roaming\EMTP\C_Program_Files_x86_EMTPWorks_4_0\emtpstate.ini;;;1;





EMTP
The reference for power systems transients

DLL debugging

- To start the debug:
  - Place some breakpoints
  - Build the code
  - Click on start (F5). A window will open asking to select the Netlist file.

Note: If, after several runs, the debug start does not work, it might be because while manipulating the DLL, the user blocked the EMTPWorks process. In this case, try the following:

- Close the EMTP console
- Kill the EMTPWorks process using the Task manager.

```
SUBROUTINE DLL_INITIALIZE_NEW(myname,idev,Data_Section,DLL_NAME)
  USE FDLL_DATA
  !DEC$ ATTRIBUTES DLLEXPORT:: DLL_INITIALIZE_NEW

  CHARACTER(LEN=*), INTENT(IN) :: myname          !device name in EMTPWorks
  INTEGER,          INTENT(IN) :: idev            !device number in the list c
  CHARACTER(LEN=*), INTENT(IN) :: Data_Section(*) !Complete data section of th
  CHARACTER(LEN=*), INTENT(IN) :: DLL_NAME        !the name of this DLL as ref
  INTEGER nlines


  IF( EXISTENCE ) THEN !at least one device already exists, allocated a new one
    ALLOCATE(myDLL%next);
    myDLL=>myDLL%next
```


EMTP
The reference for power systems transients

DLL interface

In this section, it is explained how to call a third party DLL from the EMTP® DLL

EMTP
The reference for power systems transients

DLL interface

For some applications, the EMTP® DLL is an interface to call another DLL (like a manufacturer code). In this case, the full directory of the manufacturer DLL must be provided to the EMTP® interface DLL through ModelData and it might exceed 100 characters.
In this case, EMTP® automatically wraps the directory string on several lines.

For example, if the manufacturer DLL is placed in the same folder as the design, the full directory can be found using the EMTP® JavaScript command: var *directory = currentDesign().getAttribute("CctPath")+'DLL_name.dll';*

The actual path should then be assembled in the EMTP® interface DLL. The following piece of code can be used:

```
!Parameter definition
CHARACTER(1000) :: Control_Dll_Path    !It is assumed that the path length is less than 1000 characters
CHARACTER(100)  :: line_Dll_Path
INTEGER         :: indexDLL, nlines,nlinesPath
             :
             :
!Read the first line
READ(Data_Section(nlines),'(A100)',ERR=999, END=999) Control_Dll_Path(1:100) != Data_Section(nlines)
nlines = nlines + 1
nlinesPath=1
!Read all lines until .dll is found
DO WHILE(INDEX(Control_Dll_Path, '.dll') == 0)
       READ(Data_Section(nlines),'(A100)',ERR=999, END=999) line_Dll_Path
       Control_Dll_Path = Control_Dll_Path(:100*(nlinesPath))//line_Dll_Path  !Concatenate the lines
       nlines = nlines + 1
       nlinesPath = nlinesPath + 1
END DO
indexDLL = INDEX(Control_Dll_Path, '.dll')
Control_Dll_Path = Control_Dll_Path(1:(indexDLL+3))   !Clean Control_Dll_Path.
Control_Dll_Path = trim(Control_Dll_Path)
```


EMTP
The reference for power systems transients

DLL interface

Once the full directory is retrieved, the manufacturer DLL can be retrieved with loadLibrary, getprocadress and C_F_PROCPOINTER functions. `USE, INTRINSIC :: iso_c_binding` should be added to the module definition if the manufacturer DLL is coded in C.

```
myDLL%lib_handle=loadlibrary((TRIM(Control_Dll_Path))//char(0));
if(myDLL%lib_handle == 0) then
        myDLL%dllFound = .false.
        GO TO 999  !Error line
else
       myDLL%dllFound = .true.
endif

myDLL%initProcedure= getprocaddress(myDLL%lib_handle, “initProcedure"//CHAR(0) )

myDLL%update_at_t_Procedure  = getprocaddress(myDLL%lib_handle, “update_at_t_Procedure"//CHAR(0) )
```

Then, to call the function:

```
USE kernel32
USE, INTRINSIC :: iso_c_binding
INTEGER(C_INTPTR_T), intent(in) ::  sub_handle
PROCEDURE(), POINTER :: initProcedure_
DOUBLE PRECISION :: Parameter1, ...

CALL C_F_PROCPOINTER (TRANSFER(sub_handle, C_NULL_FUNPTR), initProcedure_)
CALL initProcedure_(Parameter1, ...)
```

EMTP
The reference for power systems transients