PSIM

TUTORIAL
How to Use General DLL Block

October 2016

Unlike the simple DLL blocks with fixed number of inputs and outputs (such as the DLL Block with 3 inputs and 3 outputs), the general DLL block provides more flexibility and capability in interfacing PSIM with custom DLL files. This file describes the convention and basis of the general DLL block.

The general DLL block allows users to write code in C or C++, compile it as a Windows DLL, and link to PSIM. There are four exported functions. Three of them are used by the PSIM simulation engine, and one is used by the user interface.

The user interface function is **REQUESTUSERDATA.** The three simulation functions are: **SimulationStep**, **SimulationBegin**, and **SimulationEnd**. There are two additional functions: **GetPsimValue** and **GetPsimText** that can be called in the simulation functions to retrieve information such as time step, total time, schematic file path, and other circuit parameters. Each function has the non-Unicode version and the Unicode version. They are described below.

Function Description

<u>Function SimulationStep:</u>

Non-Unicode version:

```
void SimulationStep (
        double t,
        double delt,
        double *in,
        double *out,
         int *pnError,
        char * szErrorMsg,
        void ** ptrUserData,
        int     nThreadIndex,
        void * pAppPtr
        )
```

Unicode version:

```
void SimulationStepW (
        double t,
        double delt,
        double *in,
        double *out,
         int *pnError,
        wchar_t* szErrorMsg,
        void ** ptrUserData,
        int     nThreadIndex,
        void * pAppPtr
        )
```

This function is the only function in the DLL routine that is mandatory. All other functions are optional. This function is called by PSIM at each time step.

The DLL routine receives values from PSIM as inputs, performs the calculation, and sends results back to PSIM. The node assignments are: the input nodes are on the left, and the output nodes are on the right. The sequence is from the top to the bottom.

Each parameter is explained as follow.

double t :
  (Read only) Time in seconds

double delt :
  (Read only) Time step in seconds

double * in :
  (Read only) Array of input values. If the DLL block has three inputs, they are accessed by in[0], in[1], and in[2]

double *out :
  (Write only) Array of output values. After calculations are performed, outputs should be written to this array. If the DLL block has four outputs, they would be accessed by out[0], out[1], out[2], and out[3]

int *pnError:
  (Write only) On successful return, set pnError to 0. On error, set it to 1.
  Example: **pnError = 0; //success*

char * szErrorMsg: (Unicode version: wchar_t* szErrorMsg)
  (Write only) If there is an error, copy the error message to this string.

  Example: strcpy(szErrorMsg, "input 2 cannot exceed 50V");

  Unicode Example: _tcscpy(szErrorMsg, _T("input 2 cannot exceed 50V"));

void ** ptrUserData:
  (Read, Write) Pointer of the user-defined data. For more information, refer to the function **SimulationBegin**.

int nThreadIndex:
  (Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

void * pAppPtr:
  (Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

Function SimulationBegin:

Non-Unicode version:

```
void SimulationBegin(
        const char *szId,
         int nInputCount,
        int nOutputCount,
        int nParameterCount,
        const char ** pszParameters,
         int *pnError,
        char * szErrorMsg,
        void ** ptrUserData,
        int     nThreadIndex,
        void * pAppPtr
        )
```

Unicode version:

```
void SimulationBeginW(
           const wchar_t *szId,
            int nInputCount,
           int nOutputCount,
           int nParameterCount,
           const wchar_t ** pszParameters,
            int *pnError,
           wchar_t * szErrorMsg,
           void ** ptrUserData,
           int     nThreadIndex,
           void * pAppPtr
           )
```

This function is optional. It is called only once at the beginning of the simulation. It receives information from the DLL routine, and allows the DLL routine to allocate memory for its own use. The parameters are explained as below.

const char *szId: (Unicode version: const wchar_t *szId)
(Read only) string ID of the DLL block

int nInputCount:
(Read only) Number of input nodes

int nOutputCount:
(Read only) Number of output nodes

int nParameterCount:
(Read only) Number of parameters including optional input file path if present.

const char ** pszParameters: (Unicode version: const wchar_t ** pszParameters)
(Read only) Parameter values. Parameters start from index zero (pszParameters[0]).

If optional input file exist then it would be the last parameter (pszParameters[nParameterCount-1]).

int *pnError:
(Write only) On successful return, set pnError to 0. On error, set it to 1.
Example: **pnError = 0; //success*

char * szErrorMsg: (Unicode version: wchar_t* szErrorMsg)

> (Write only) If there is an error, copy the error message to this string.
>
> Example*:* strcpy(szErrorMsg, "input 2 can not exceed 50V");
>
> Unicode Example*:* _tcscpy(szErrorMsg, _T("input 2 can not exceed 50V"));

void ** ptrUserData:

> (Read, Write) This is a pointer to the user-defined data. The memory must be allocated in the function **SimulationBegin** and freed in **SimulationEnd**. It is passed to **SimulationStep** on every call. It allows the DLL to manage its own data during the simulation.
>
> **Note**: This pointer is not the same as the user-defined pointer in the function **REQUESTUSERDATA**. It is not possible to pass a pointer from **REQUESTUSERDATA** to any of the simulation functions. They communicate via parameters only.

int nThreadIndex:

> (Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

void * pAppPtr:

> (Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

<u>Function SimulationEnd:</u>

Non-Unicode version:

```
void SimulationEnd (
        const char *szId,
        void ** ptrUserData,
        int     nThreadIndex,
        void * pAppPtr
        )
```

Unicode version:

```
void SimulationEndW(
        const wchar_t *szId,
        void ** ptrUserData,
        int     nThreadIndex,
        void * pAppPtr
        )
```

This function is optional. It is called only once at the end of the simulation. Its main purpose is to allow DLL to free any memory or resources that it has allocated. The parameters are explained below.

const char *szId : (Unicode version: const wchar_t *szId)

> (Read only) String ID of the DLL block

void ** ptrUserData:
(Read, Write) Pointer of the user-defined data. For more information, refer to the function **SimulationBegin**.

int nThreadIndex:
(Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

void * pAppPtr:
(Read only) This is a required parameter for calling **GetPsimValue** and **GetPsimText**.

Function GetPsimValue:

Non-Unicode version:

```
double GetPsimValueA(
        int     nThreadIndex,
        void * pAppPtr,
        char * szName,
        char * szParamName,
        int & nStatus
        )
```

Unicode version:

```
double GetPsimValueW(
        int     nThreadIndex,
        void * pAppPtr,
        wchar_t * szName,
        wchar_t * szParamName,
        int & nStatus
        )
```

This function is called in **SimulationStep**, **SimulationBegin** or **SimulationEnd** to retrieve circuit information during simulation. The parameters are explained below.

int nThreadIndex:
This is a required parameter that is present in each of the simulation functions.

void * pAppPtr:
This is a required parameter that is present in each of the simulation functions.

const char * szName: (Unicode version: const wchar_t * szName)

const char * szParamName: (Unicode version: const wchar_t * szParamName)
Combination of szName and szParamName determines the parameter value retrieved by this function.

Description	szName	szParamName
Total Time	"PSIM"	"TOTALTIME"
Time Step	"PSIM"	"TIMESTEP"
Load Flag (0 or 1)	"PSIM"	"LOADFLAG"
Save Flag (0 or 1)	"PSIM"	"SAVEFLAG"
To retrieve the value of a variable from a parameter file. For example variable 'alpha'.	(Empty string) ""	Variable name: "alpha"
To retrieve an attribute of one of the circuit elements for example Resistance of the resistor 'R1' Hint: Place a C-Block in the schematic window and click on "GetPsimValue" button to get a list of elements and their attributes. C-Block code is compatible with DLL-Block code.	"R1"	"Resistance"

int & nStatus:

If function is successful, nStatus is set to 0 and function return value is the requested value. If nStatus is not zero after function returns then there was an error and the return value is zero.

Return value: (double)

Requested value if nStatus is zero.

This function requires the header file "psim.h". Below is an example of how nStatus is used.

```
	int nStatus = -1;  // If the function succeeds, the value of nStatus will be 0.
	double fTotalTime = GetPsimValueW(nThreadIndex, AppPtr, _T("PSIM"), _T("TOTALTIME"),
nStatus);
	if(nStatus == 0)
	{
		//Success
		//...
	}
	else
	{
		//Error...
		//fTotalTime is invalid.
		//...
	}
```

Function GetPsimText:

Non-Unicode version:

```
int  GetPsimTextA(
        int     nThreadIndex,
        void * pAppPtr,
        char * szName,
        char * szParamName,
        char *szReturn,
        int *pnReturnLen,
        int & nStatus
        )
```

Unicode version:

```
int GetPsimTextW(
        int     nThreadIndex,
        void * pAppPtr,
        wchar_t * szName,
        wchar_t * szParamName,
        wchar_t *szReturn,
        int *pnReturnLen,
        int & nStatus
        )
```

This function is called in **SimulationStep**, **SimulationBegin** or **SimulationEnd** to retrieve circuit information during simulation. The parameters are explained below.

int nThreadIndex:

This is a required parameter that is present in each of the simulation functions.

void * pAppPtr:

This is a required parameter that is present in each of the simulation functions.

const char * szName: (Unicode version: const wchar_t * szName)

const char * szParamName: (Unicode version: const wchar_t * szParamName)

Combination of szName and szParamName determines the parameter value retrieved by this function.

Description	szName	szParamName
PSIM Folder	"PSIM"	"PSIMFOLDER"
Three part PSIM version (10.0.5)	"PSIM"	"PSIMVERSION"
Full path and file name of schematic file.	"PSIM"	"SCHPATH"
Full path and file name of simview file.	"PSIM"	"SMVPATH"
Schematic file's Path without the file name.	"PSIM"	"SCHFOLDER"

Simview file's Path without the file name.	"PSIM"	"SMVFOLDER"
Schematic file name without the folder path.	"PSIM"	"SCHNAME"

char *szReturn: (Unicode version: wchar_t *szReturn)
    Requested value if nStatus is zero.

int *pnReturnLen:
    Length of returned string.

int & nStatus:
    If function is successful, nStatus is set to 0 and function return value is the requested value. If nStatus is not zero after function returns then there was an error and the return value is zero.

Function REQUESTUSERDATA:

Non-Unicode version:

```
void REQUESTUSERDATA(
        int nRequestReason,
        int nRequestCode,
        int nRequestParam,
        void ** ptrUserData,
        int * pnParam1,
        int * pnParam2,
        char * szParam1,
        char * szParam2
        )
```

Unicode version:

```
void REQUESTUSERDATAW(
        int nRequestReason,
        int nRequestCode,
        int nRequestParam,
        void ** ptrUserData,
        int * pnParam1,
        int * pnParam2,
        wchar_t * szParam1,
        wchar_t * szParam2
        )
```

This function is optional. It handles the user interface with PSIM. It is called by PSIM when the general DLL block element is created, or its properties are modified in the property box. The parameters are explained below.

int nRequestReason :

It describes the user action when this function is called. Possible values are:

ACTION_DLL_SELECTED	A new DLL block is placed on the schematic and this DLL is selected.
ACTION_ELEMENT_LOAD	A schematic file with DLL blocks is being loaded, or being pasted as part of the copy/paste operation.
ACTION_ELEMENT_SAVE	A schematic file with DLL blocks is being saved or a buffer (Copy/paste operation)
ACTION_INPUTFILE_CHANGED	A new input file is selected.
ACTION_PARAMETERS_CHANGED	Parameters are changed in the property dialog box.
ACTION_ELEMENT_DELETE	An element is being deleted or a file is being closed.

Example: When a user places a DLL block from the library on to a schematic, after the DLL file name is defined for this block, the function **REQUESTUSERDATA** will be called and the parameter nRequestReason will be set to ACTION_DLL_SELECTED.

int nRequestCode :

It describes the information that is being requested from DLL. Possible values are:

REQUEST_BEGIN:

A new request is initiated. The values of pnParam1, pnParam2, szParam1, and szParam2 depend on the value of nRequestReason.

REQUEST_END:

Request is finished.
pnParam1: Ignored
pnParam2: Ignored
szParam1: Ignored
szParam2: Ignored

REQUEST_PARAM_COUNT:

Request the number of parameters that will be displayed on the right side of the property box, between 0 and 10.
pnParam1(Read, Write): Number of parameters.
pnParam2(Read, Write): If an input data file is required, set it to 1, otherwise set it to 0.
szParam1(Read, Write): File open dialog filter for the input data file. It must conform to the file open dialog filter format. The following is a properly formatted filter string:"All Files|*.*|Image files|*.jpg;*.bmp;*.gif|Text Files|*.txt||". The default value for the filter is: "All Files|*.*||"

szParam2(Read only): String ID associated with the element.

Example: The following code is a possible response to this request. It asks for three parameters and the input file with extension "aaa".

```
*pnParam1 = 3; // 3 parameters
*pnParam2 = 1; // Input Data File is required
strcpy(szParam1, "MyProgram Files|*.aaa|All Files|*.*||"); //File Filter
```

REQUEST_DATAFILE_INFO:

Request information of the input data file.

pnParam1(Read, Write): If set to 1, a check box will be displayed in the property box. When the check box is checked, the file path will be displayed in the schematic.

pnParam2: Ignored

szParam1(Read, Write): Label that is displayed in the property box. The length is 20 characters maximum. The default value is "Input Data File".

szParam2(Read Only): Full path and file name of the selected file.

Example: The following code changes the label from the default value of "Input Data File" to "MyProgram data File", and requests a check box for the data file.

```
strcpy(szParam1, "MyProgram data File");
*pnParam1 = 1; // Show Display check box
```

REQUEST_PARAM_INFO:

Request information on each parameter.

nRequestParam(Read only): Zero based parameter index

pnParam1: If set to 1, a check box will be displayed in the property box. When the check box is checked, the parameter value will be displayed in the schematic.

pnParam2 : ignored

szParam1(Read, Write): Label that is displayed in the property box. The length is 20 characters maximum.

szParam2(Read, Write): The parameter value. The length is 50 characters maximum.

Example: The following code sets the labels of 3 parameters and ensures that Parameter 1 is an integer between 3 and 10.

```
switch(nRequestParam)
{ //Three Parameters expected
    case 0:
    strcpy(szParam1, "Moving Avg. Samples");
```

```
                *pnParam1 = 1; // Show Display check box
                //verify the value
                nVal = atoi(szParam2);
                if( nVal < 3 )
                {
                        nVal = 3;
                }
                if( nVal > 10 )
                {
                        nVal = 10;
                }
                itoa( nVal, szParam2, 10);
                break;

                case 1:
                strcpy(szParam1, "Multiplier");
                *pnParam1 = 1; // Show Display check box
                if( strlen(szParam2) == 0 )
                {
                        strcpy(szParam2, "3.14");
                }
                break;

                case 2:
                strcpy(szParam1, "Title");
                *pnParam1 = 0; // Do not Show Display check box
                break;

                default://Only expecting three parameters
                break;
        }
```

REQUEST_IN_OUT_NODES:

Get and set the number of input and output nodes

pnParam1(Read, Write): Number of input nodes.
pnParam2(Read, Write): Number of output nodes.
szParam1: Ignored
szParam2: Ignored

These values can be modified at any time and the changes will be reflected immediately in the schematic window.

Example: The following code sets the number of input nodes to 1 and the number of output nodes to 2:

```
        *pnParam1 = 1; // one input node
```

```
*pnParam2 = 2; // two output nodes
```

REQUEST_INPUT_NODE_INFO:

Request information on each input node.

nRequestParam: Zero based node index
pnParam1: Ignored.
pnParam2: Ignored.
szParam1(Read, Write): Node label. The length is 20 characters maximum.
szParam2: Ignored

Example: The following code sets the label of the first input node.

```
switch(nRequestParam)
{
//One input node
        case 0:
        strcpy(szParam1, "in1");
        break;

        default:
        break;
}
```

REQUEST_OUTPUT_NODE_INFO

Request information on each output node.

nRequestParam: Zero based node index
pnParam1: Ignored.
pnParam2: Ignored.
szParam1(Read, Write): Node label. The length is 20 characters maximum.
szParam2: Ignored

For each output node you may change its label at any time.

Example: The following code sets the label for two output nodes.

```
switch(nRequestParam)
{ //two output nodes
        case 0:
        strcpy(szParam1, "o1");
        break;

        case 1:
        strcpy(szParam1, "o2");
        break;

        default:
        break;
}
```

int nRequestParam :

This value depends on the parameter nRequestCode.

void ** ptrUserData:

This is a pointer to the user-defined data. It is included in every function call and allows users to manage one's own data. Memory is allocated and freed by the user. Memory can be allocated, reallocated or freed at any time. However, the logical choice for allocating the memory is at

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_BEGIN,…)

and

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_BEGIN, …)

If data is loaded from an input file, one may also choose to allocate the memory at

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_BEGIN,…)

The memory is generally freed at

REQUESTUSERDATA(ACTION_ELEMENT_DELETE, REQUEST_BEGIN,…)

Example: The following code demonstrates how to use this value.

```
Struct MyStruct
{
        int myInteger;
        …
}

//Allocate memory
*ptrUserData = new MyStruct();
        …
// Use your internal data
MyStruct * pData = (MyStruct)(*ptrUserData);
PData-> myInteger = 10;
…
// Free  memory
if( *ptrUserData != NULL )
{
        delete (MyStruct)(*ptrUserData);
         *ptrUserData = NULL;
}
```

int * pnParam1, int * pnParam2 , char * szParam1, char * szParam2:

These arguments depend on the values of nRequestReason, nRequestCode and nRequestParam.

Call Sequences

To write a DLL that works with PSIM, It is essential to understand the way PSIM interacts with the DLL. The parameter nRequestReason of the function **REQUESTUSERDATA** describes the user action. The following is a description of the **REQUESTUSERDATA** function calls, organized based on the value of nRequestReason.

ACTION_DLL_SELECTED

The user places a general DLL block on the PSIM schematic, double clicks on the image to bring up the property box, and defines the “DLL File”.

At this point, PSIM loads the DLL and looks for the function **RUNSIMUSER**. If this function does not exist, DLL will be rejected. The **RUNSIMUSER** function is mandatory for the general DLL block.

Next, PSIM will look for the function **REQUESTUSERDATA** in the DLL. If DLL exports this function, PSIM calls it several times to gather information on the DLL. Here is the description of each of these calls:

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1: Ignored.
pnParam2: Ignored.
szParam1(Read only): Full path and file name of the DLL.
szParam2(Write only): Full path and file name of the default input data file.

This is a good place to allocate memory for ptrUserData. For example,

*ptrUserData = new Internal_DLL_RuntimeData();

If a default file for the input data file is required, copy its full path and file name to szParam2. The file must exist, otherwise it will be ignored. Also, at the next call REQUEST_PARAM_COUNT, the input data file must be requested. If a file path is set at this point, after this sequence ends, a new sequence with ACTION_INPUTFILE_CHANGED will be executed.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_PARAM_COUNT, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

For more information, refer to the section on nRequestCode.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_DATAFILE_INFO, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

For more information, refer to the section on nRequestCode.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_PARAM_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

This function is called several times with the value of “nRequestParam” set from 0 to “number of parameters – 1” (the number of parameters was set in previous calls). For each parameter you may change its label or value at any time.

For more information, refer to the section on nRequestCode.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_IN_OUT_NODES, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

This function is called to get and set the number of input and output nodes. You may change the number of nodes at any time. In some cases, number of nodes depends on the input data file. In that case you can ignore this call here and handle it at ACTION_INPUTFILE_CHANGED.
For more information, refer to the section on nRequestCode.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_INPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

For more information, refer to the section on nRequestCode and REQUEST_INPUT_NODE_INFO.

This function is called several times with the value of nRequestParam set from 0 to “number of input nodes – 1”.

REQUESTUSERDATA(ACTION_DLL_SELECTED, EQUEST_OUTPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

This function is called several times with the value of nRequestParam set from 0 to “number of output nodes – 1”.

For more information, refer to the section on nRequestCode.

REQUESTUSERDATA(ACTION_DLL_SELECTED, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

For more information, refer to the section on nRequestCode.

ACTION_ELEMENT_LOAD

There are two situations where this sequence of calls would be executed. One is when a schematic file is being loaded and a general DLL block is present in the file. Another is during a copy/paste operation when a general DLL block is being pasted.

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1(Read only): Number of valid bytes in szParam1
pnParam2: Ignored.
szParam1(Read only): Binary data that is provided by DLL at save time. Refer to ACTION_ELEMENT_SAVE for more information.
szParam2(Read, Write): Selected input file path. One may modify the file path here.

This is a good place to allocate memory for ptrUserData. For example,

```
*ptrUserData = new Internal_DLL_RuntimeData();
```

This is the first call to the DLL from PSIM for each DLL block. If the DLL block is new (selected from the **Elements** menu and placed in the schematic window), the sequence of calls with ACTION_ELEMENT_LOAD would not be executed. Instead, the sequence of calls with ACTION_DLL_SELECTED is executed.

You may decide to ignore the rest of this sequence, in which case the values that were selected in the previous session and saved in a file, or values that are copied (in case of copy/paste operation) would be used.

The following calls are the rest of this sequence. For more information, refer to the nRequestCode section.

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_PARAM_COUNT, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_DATAFILE_INFO, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_PARAM_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_IN_OUT_NODES, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_INPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_OUTPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_ELEMENT_LOAD, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

ACTION_ELEMENT_SAVE

This sequence is called when saving an element to a file or copying an element as part of a copy/paste operation.

A buffer with the maximum size of 100 bytes is provided for saving the DLL specific information. This could be the DLL version or any additional parameter that is not one of the displayed parameters. This buffer will be passed back to the DLL during the sequence ACTION_ELEMENT_LOAD.

REQUESTUSERDATA(ACTION_ELEMENT_SAVE, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1(Write only): Number of valid bytes in szParam1
pnParam2: Ignored.
szParam1(Write only): Copy binary data to be saved in the .sch file (DLL version, File Version, ...) (maximum 100 bytes)
szParam2(Read only): Selected input file path

For example, the following code copies 25 bytes of user defined binary data:

//copy 25 bytes of binary data.

memcpy(szParam1, mydata, 25);
**pnParam1 = 25;*

REQUESTUSERDATA(ACTION_ELEMENT_SAVE, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

ACTION_INPUTFILE_CHANGED

This sequence is executed every time a new input data file is selected in the DLL block's property box.

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1(Write only): By setting pnParam1 to 0, you can reject this file. If the file is rejected, REQUEST_END is called immediately and the value of input file path is reverted back to the previous value.
pnParam2: Ignored.
szParam1(Read, Write): Full path and file name of the selected input file. This parameter is both readable and writeable. You may modify the path and file name.
szParam2: Ignored.

The following calls are the rest of this sequence if the file is not rejected.

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_PARAM_COUNT, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_DATAFILE_INFO, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_PARAM_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_IN_OUT_NODES, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_INPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_OUTPUT_NODE_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

REQUESTUSERDATA(ACTION_INPUTFILE_CHANGED, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

ACTION_PARAMETERS_CHANGED

This sequence is executed every time the data in the property box is modified.

REQUESTUSERDATA(ACTION_PARAMETERS_CHANGED, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1: Ignored.

pnParam2: Ignored.
zzParam1: Ignored.
szParam2: Ignored.

REQUESTUSERDATA(ACTION_PARAMETERS_CHANGED, REQUEST_PARAM_INFO, nRequestParam, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

This function is called several times with nRequestParam set from 0 to 'number of parameters – 1'
nRequestParam: Zero based parameter index.
pnParam1: Ignored.
pnParam2: Ignored.
szParam1(Read Only): Parameter label.
szParam2(Read, Write): Parameter value. The value can be changed, but it can not be rejected and the message box can not be popped up in this function.

REQUESTUSERDATA(ACTION_PARAMETERS_CHANGED, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

ACTION_ELEMENT_DELETE

This sequence is executed when an element is being deleted or the schematic file is being closed. Any memory or resources that have been created must be freed.

REQUESTUSERDATA(ACTION_ELEMENT_DELETE, REQUEST_BEGIN, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2):

pnParam1: Ignored.
pnParam2: Ignored.
zzParam1: Ignored.
szParam2: Ignored.
This is the proper place to free any allocated memory. For example,

```
delete (Internal_DLL_Block_RuntimeData *)(*ptrUserData);
*ptrUserData = NULL;
```

REQUESTUSERDATA(ACTION_ELEMENT_DELETE, REQUEST_END, 0, ptrUserData, pnParam1, pnParam2, szParam1, szParam2);

Examples

Three examples are provided to illustrate the use of the general DLL block, as described below.

TestBlock (Files: general_dll_block_test1.sch, TestBlock.dll):

In this example, the number of input/output nodes and the node labels are read from an input data file. When the DLL block is loaded from the schematic file (*ACTION_ELEMENT_LOAD*) or the input file is changed (*ACTION_INPUTFILE_CHANGED*), the DLL loads the file and reallocates the memory for its internal data. The general DLL block has ten parameters. It sets the number of parameters when a new element is created by handling *ACTION_DLL_SELECTED: REQUEST_PARAM_COUNT*. It ignores all other calls to *REQUEST_PARAM_COUNT* , thus retaining the original value of the number of parameters through the life of the element and through multiple simulation sessions.

TestBlock2 (Files: general_dll_block_test2.sch, TestBlock2.dll):

In this example, the general DLL block has one input node and two output nodes. It has an input data file and three parameters (an integer, a floating-point value, and a string). The string parameter is used as the output file name. During the simulation, the DLL opens and reads the input data file. It performs calculations using the values from the input data file and the parameters. The DLL writes results to the output data file.

TestBlock3 (Files: general_dll_block_test3.sch, TestBlock3.dll):

In this example, the DLL does not export the user interface function *REQUESTUSERDATA*. The property box of the element allows the user to enter the number of input and output nodes. There are also five parameters and an input data file. This example demonstrates how the DLL obtains the parameter values and the input data file path from PSIM. These parameters and the input data file, however, are not used inside the DLL and are left blank. The example assumes that the general DLL block has at least one input node and one output node.