OpenDSS Documentation

Generator UserModel DLL

Sept, 5, 2012

The OpenDSS Generator element model supports user-written DLLS for both power flow and dynamics applications.

An example of where this model is useful is the IndMach012 model supplied with OpenDSS. The model implements a simple symmetrical component model of an induction machine.

Public Data Structure

All devices in OpenDSS can have a public data structure if the programmer has provided it. A pointer to the structure may be obtained for the active circuit element through the **GetPublicDataPtr** function in the Callback routines. A pointer to the TDSSCallBacks structure is passed through argument list for the New function when a new instance of the user-written model is created.

The Pascal declaration of the public data structure for the Storage element is:

```
{Struct to pass basic data to user-written DLLs}
TGeneratorVars = packed Record

        Theta,      {Direct-Axis voltage angle}
        Pshaft,
        Speed,      {relative speed}
        w0,         { rad/sec, difference from Synchronous speed, w0}
                    {actual speed = Speed + w0}
        Hmass,      {Per unit mass constant}
        Mmass,      {Mass constant actual values (Joule-sec/rad}
        D, Dpu,     {Actual and per unit damping factors}
        kVArating,
        kVGeneratorBase,
        Xd, Xdp, Xdpp,   {machine Reactances, ohms}
        puXd, puXdp, puXdpp,   {machine Reactances, per unit}
        dTheta,
        dSpeed,   {Derivatives of Theta and Speed}
        ThetaHistory, SpeedHistory,   {history variables for integration}
        Pnominalperphase, Qnominalperphase  {Target P and Q for power flow solution,
watts, vars}

               : Double;    { All Doubles }

        {32-bit integers}
        NumPhases,       {Number of phases}
        NumConductors,   {Total Number of conductors (wye-connected will have 4)}
        Conn           :Integer;   // 0 = wye; 1 = Delta

        { Revisons (additions) to structure ...
          Later additions are appended to end of the structure so that
          previously compiled DLLs do not break
```

```
         }

        VthevMag  : Double;    {Thevinen equivalent voltage for dynamic model}
        VThevHarm : Double;    {Thevinen equivalent voltage mag reference for Harmonic
model}
        ThetaHarm : Double;    {Thevinen equivalent voltage angle reference for Harmonic
model}
        VTarget   : Double;   // Target voltage for generator with voltage control
        Zthev     : Complex;
   End;
```

Public Data Variable Definitions

Theta :double;

Generator phase angle computed during Dynamics analysis. Initialized with Vthevmag upon entering dynamics mode to yield approximately the same positive sequence power as the power flow solution.

Pshart :Double;

Shaft power, W;

Speed :Double;

Rate of change of Theta. Speed relative to synchronous speed, rad/sec.

w0 :Double;

2*pi*f. Base frequency, rad/sec

Hmass :Double;

Per unit mass constant;

Mmass :Double;

Mass constant in actual values, Joule-sec/rad.;

D, Dpu :Double;

Damping factor in actual units and in per unit. D := Dpu * kVArating *1000.0/(w0);

kVArating :Double;

kVA rating of the Generator.

kVGeneratorBase:Double;

kV rating of the generator.

Xd, Xdp, Xdpp:Double;

Machine Xd, Xd', Xd" in actual ohms on the machine base.

puXd, puXdp, puXdpp:Double;

Machine Xd, Xd', Xd" in per unit as defined by the user using the Generator object properties.

dTheta :Double;

Derivative of Theta (usually the same as Speed) at the current time step.

dSpeed :Double;

Derivative of Speed at the current time step.

ThetaHistory :Double;

Value of Theta at the previous time step.

SpeedHistory :Double;

Value of Speed at the previous time step.

PnominalPerPhase :Double;

Nominal target active power, W, per phase.

QnominalPerPhase :Double;

Nominal target reactive power, vars, per phase.

{The following are 32-bit integers}

NumPhases,

Number of phases in the Storage element terminal (only one terminal).

NumConductors,

Total Number of conductors in the Storage element terminal. 1-phase elements always have 2. If wye-connected, 3-phase, there will be 4.

Conn :Integer

Terminal connection. 0 = Wye (or L-N); 1 = Delta (or L-L)

VthevMag :Double;

Thevenin voltage magnitude, volts, used in Dynamics model for the direct axis voltage. Along with Theta, this value is initialized so that the power produced by the Generator object is approximately the same as the most recent power flow solution prior to entering Dynamics mode.

VThevHarm:Double;

Thevenin voltage magnitude, volts, used in Harmonics model.

ThetaHarm:Double;

Thevenin voltage angle, radians, used in Harmonics model.

VTarget   :Double;

Target voltage for generator with voltage control.

Zthev   :Complex;

Thevenin equivalent impedance, R + jX, ohms used in dynamics model.

Generator User DLL Function Interface

The declaration of the functions and procedures in the DLL interface for a user-written DLL is for dynamics simulation only. The name of the DLL is specified by the *UserModel=MyUserWrittenDLL* statement in the description of the Generator element. The OpenDSS will then load the DLL and attempt to use it. All the functions described here will have to be found before the OpenDSS will continue.

User-written DLLs should be capable of supporting more than one instance of a model. Each of several Storage elements in the OpenDSS circuit model could invoke the same DLL. The DLL is only loaded one time, but could be called by more than one Storage element. Therefore, the DLL must be able to manage multiple instances. An integer handle is assigned to each instance by the DLL (see below).

All function calls use the Stdcall calling convention. This is also the convention used in the Windows API.

If you would like to study the DLL interface in OpenDSS, please see the StoreUserModel.Pas file in the ..\Source\PCelement folder on the source code sharing site.

Function New

(Var GenVars : TGeneratorVars;
Var DynaData : TDynamicsRec;
Var CallBacks : TDSSCallBacks) : Integer; Stdcall;// Make a new instance

This function creates a new instance of the model defined by the DLL. It returns an integer handle to the instance that may be used to specifically select this instance at a later time.

Note that for some OpenDSS objects, the New function may take a different number of parameters. It depends on when the model was written. Both the TGeneratorVars and TDynamicsRec structures can also be obtained now from the callback function.

User-written DLLs should have some mechanism for keeping track of instances of the class represented by the DLL. This could be a simple array of pointers or a linked list or some other mechanism of the programmer's choosing. There is no internal OpenDSS mechanism to support this. Each Generator object that creates a Usermodel object will keep track of only the one that belongs to it and use the Select function to instruct the DLL with instance to make active.

Arguments:

GenVars: The TGeneratorVars structure described above is passed to the New function by reference. That is, expect a pointer to the structure.

DynaData: The TDynamicsRec structure is passed by reference. This contains the time variables for a typical dynamics simulation. This is defined as follows.

```
TDynamicsRec = Packed Record

    h,     // Time step size in sec for dynamics
    t,     // sec from top of hour
```

```
       tstart,
       tstop:Double;
       IterationFlag:Integer;
{0=New Time Step; 1= Same Time Step as last iteration}
       SolutionMode :Integer;
       intHour :Integer;  // time, in hours as an integer
       dblHour :Double;   // time, in hours as a floating point
number including fractional part
   End;
```

CallBacks: The callback structure is passed by reference. See OpenDSS Callback Routines documentation. The user-written DLL would typically keep a pointer to this structure and call the functions and procedure in it to obtain specific data or to use internal OpenDSS functions such as messaging.

Procedure Delete

(var ID:Integer); Stdcall; // deletes specified instance

This procedure deletes a specific instance referenced by the ID. This is the value returned by the New function when the instance was created. Note it is passed by reference.

Function Select

(var ID:Integer):Integer; Stdcall; // Select active instance

This function selects an instance of the objects instantiated by the DLL. The ID is the value returned by the New function. ID is passed by reference (i.e., expect a pointer to an integer).

If successful, the function retuns the ID as confirmation. If the return value is 0, there has been an error.

Procedure Init

(V, I:pComplexArray);Stdcall;

This procedure is called when the OpenDSS is entering Dynamics mode from one of the power flow modes. Two pointers to complex number arrays are passed representing the present voltage, V, and current, I, at the Storage element terminals. The user-written DLL uses these values to perform whatever calculations are required to initialize the active model.

Procedure Calc

(V, I:pComplexArray); stdcall;

This procedure invokes the main electrical calculation algorithm of the model. This will usually be to compute the terminal currents, I, given the present estimate of the voltages, V, and internal state variables. For "usermodel", this function basically computes I given V. For "shaftmodel", uses V and I to calculate Pshaft, speed, etc. in the GeneratorVars data structures

V and I are pointers to complex number arrays. There will be one complex value for each conductor of the Storage element terminal. If there is any question how many conductors are present, the **GetActiveElementTerminalInfo** call back function may be called or, for the Generator element, the values are available on the public data structure (GeneratorVars).

This calc procedure should work for the various solution modes that the model will be used for. The SolutionMode field of the DynaData structure can be

checked for the present solution mode and allow appropriate action. For example, from the IndMach012 code:

```
With ActiveModel Do
  Begin
    Case DynaData^.SolutionMode of
        DYNAMICMODE: Begin
                       CalcDynamic(V012, I012);
                     End;
    Else  {All other modes are power flow modes}
          Begin
             CalcPflow(V012, I012);
          End;
    End;
  End;
```

This model has separate algorithms for power flow modes and Dynamic mode.

Procedure Integrate;

stdcall; // Integrates any state vars

This procedure is called by the OpenDSS in Dynamics mode when it is time to integrate the state variables if the model uses differential equations.

OpenDSS currently uses a two step predictor-corrector integration process. The IterationFlag variable of the TDynamicsRec structure indicates with step is currently being executed. If IterationFlag=0, the predictor step is being executed. If it is =1 the corrector step is being executed.

Most other models in the OpenDSS use a forward Euler for the predictor step and trapezoidal for the corrector step. The derivatives from the previous time step – necessary for the trapezoidal integration step – are captured during the predictor step.

Note that the Integrate function is called prior to the Calc function.

Procedure Edit

(EditStr:pAnsichar; Maxlen:Cardinal); Stdcall;

The Edit procedure is called when the OpenDSS encounters the UserData=(*MyString*) property definition in the Generator script. *MyString* is passed to this procedure in the EditStr argument as a pointer to a null-terminated Ansi character string. It is up to the Edit procedure in the DLL to interpret the string and properly set any variables. The callbacks to the OpenDSS parser may be used to help interpret the string if an OpenDSS-like syntax is used.

The length of the string is passed in the *Maxlen* argument, which is pushed onto the calling stack in case it is expected by the language in which the DLL is implemented (some languages expect this argument).

Note that it is not necessary to support OpenDSS syntax. Any format may be used for the data required to define the model. If the model requires an extensive amount of data, one approach is to pass a file name in *MyString* and have the DLL's Edit procedure read the data from the file.

Procedure UpdateModel;

StdCall;

This procedure is called by OpenDSS when it thinks it is necessary to recalculate model parameters from the present values of the data used to describe the

models. Many models will not need this, but it is provided. It is typically called after calling the Edit procedure.

Procedure Save;
StdCall;

This procedure is called by the OpenDSS to allow a user model to save its state variables to a file somewhere for a quick restart. The Restore procedure retrieves the values.

Procedure Restore;
StdCall;

This procedure is called by the OpenDSS to allow a user model to recover data written to a file by the Save procedure.

Note that as of this writing (2012) the Save .. Restore functionality is not used in the Generator model. However, the function interfaces must exist in the DLL so that the DLL will load without error.

MONITOR VARIABLES / STATE VARIABLES

The following API functions in the DLL allow the user to get and set selected variables from within the model as determined by the programmer implementing the DLL.

The values returned by these functions are automatically captured by **Monitor** elements assigned to Power Conversion (PC) elements and defined with **Mode=3**. The variables exposed by the programmer are appended to the default list of variables in the parent PC element (i.e., a Storage element). A Monitor will put the names of the variables in the header record and then capture the numeric values for each sample.

The user may return a number of double-precision floating point values for monitoring. These are frequently values of the state variables, but can be any number the programmer desires.

Function NumVars:
Integer;Stdcall; // Number of variables that can be returned for monitoring

This function returns the number of double-precision variables that can be accessed.

Procedure GetAllVars
(Vars:pDoubleArray);StdCall;

This Procedure is called by OpenDSS monitor elements and is expected to return the values of all the exposed model variables in an array of doubles. The OpenDSS will allocate the necessary space and pass a pointer to the buffer in the Vars argument.

Function GetVariable
(var i:Integer):double;StdCall; // returns a i-th variable value only

This function returns the value of the *i*-th variable. Note that *i* is passed by reference, That is, the user-written program should expect to receive a pointer to this index.

Procedure SetVariable
(var i:Integer;var value:Double); StdCall;

This procedure is provided to allow the user to set the value of a variable internal to the model. Both arguments are passed by reference. That is, the user-written program should expect to receive pointers to the values of these arguments.

The DLL is not obligated to set the internal variable to the value and should not permit it if it will break the model.

Procedure GetVarName
(var VarNum:Integer; VarName:pAnsiChar; maxlen:Cardinal); StdCall;

This procedure is provided to obtain the name assigned to a monitor or state variable. Monitor elements defined as Mode=3 will automatically query the names using this function.

VarNum is the index of the variable name requested and will be passed by reference (i.e., expect a pointer to an integer).

The OpenDSS will allocate space for *VarName*, a null-terminated Ansi character string, and pass the size of the allocation in the *maxlen* argument, which is an unsigned integer passed by value. The DLL implementation of this procedure will copy the string name of the requested variable into the buffer up to the limit given by *maxlen*.

Example

The entire source code for the IndMach012 DLL is supplied on line at the OpenDSS source sharing site.

You may use this as a pattern for writing a DLL.