Best of the Discussion Forum – Part I

Changing Generator Parameters in Matlab.

Question

In openDSS i want to change the kW of generator from matlab. i have written the code as

```
genkw=[500,600,700,800,900,1000];
for i=1:6
DSSCircuit.Generators.Name='GenL70';
DSSText.Command = 'Generator.GenL70.kW = genkw(i)';
End
```

It doesn't work. plz help.

Answer

```
genkw=[500,600,700,800,900,1000];
for i=1:6

DSSText.Command = ['Generator.GenL70.kW =' num2str(genkw(i))];

%  do something with the generator ...

End
```

When you are using the Command property of the Text interface, it is expecting a string.
Now, there is a Generators interface that manages all the Generator objects. You can pass a floating-point number through it. You would do something like this:

```
DSSGenerator = DSSCircuit.Generators

genkw=[500,600,700,800,900,1000];
for i=1:6
DSSGenerator.Name ='GenL70';   % Sets the active generator
DSSGenerator.kW = genkw(i);   % Operates on the active generator

%  Do something ...

end
```

Generating OpenDSS Scripts for Large Circuits

Question

In general, what is the easiest way to build a circuit, which is consisting out of more than 1000 lines?

For example at the 3 EPRI test cases or the IEEE 8500-Node case, which are really big circuits, building the circuit data together in OpenDSS would mean work for over a month... or am I doing something completely wrong?

I would appreciate any comments or hints how to implement a circuit the easiest way possible.

Answer 1

"I assume you are not generating a new unique circuit each time, but rather have a fixed topology circuit. You mention you have most of the data in an Excel file, if I understand correctly, that defines the circuit.
I would use some text manipulation functions in Excel (in the worksheet where you have your data), and generate the OpenDSS script files that define the circuit components, In general I would generate the proper script lines in Excel and then copy/paste into a text editor and save the file(s) as .dss file(s).
For the major types of components (say primary lines), once you have developed the Excel formula(s) to generate proper syntax dss script lines (inside of Excel) that define your components, then you can just copy/paste your formulas for all primary lines (for example) in the worksheet.
This makes it easy to generate thousands of lines of script code, which as I mentioned earlier can be saved into text files and read in by the OpenDSS either directly or via the COM server (which can be called from Excel VBA, matlab, python, etc.)."

Answer 2

"Based in my personal experience the answer for that question will be: The one in which you have more experience, this es because this way you can build faster your code, find solutions to the programming issues that can appear and will not have to learn another programming languages.
So, first, you will have to ask yourself "wich is the programming language that I know best", that way you will find the way to interact with the OpenDSS using the COM interface in an easy way."

Answer 3

"If your circuit is already modeled in some other computer program, so all you can to write code to do the conversion. If you try to do the conversion by hand, you will likely end up with errors. That said, I have successfully used a text editor I am very familiar with (EditPlus) to convert text files exported from other software into DSS script. But generally, I do like Wes and write some VBA code in either ACCESS or EXCEL to generate the scripts. I have also written special converters in C#."

Stochastic Analysis

Question 1:

"when I try to run repeatedly,the information "File"XXX" is about to be overwritten……"always come out, So if I need to run many times ,that will be quite inconvenient,how to solve this problem;"

Answer 1

To get rid of the message about overwriting when using the COM server, try

```
DSSObj.AllowForms = false
```

(adapt to your programming language)
I think the program will then just overwrite the file without asking.

Question 2:

how can I get the primary and Secondary Voltage respectively through COM?

Thank you

Answer 2

You can get all the voltages through the COM interface at any time, but I think your question is how to segregate the voltages by voltage level. OpenDSS doesn't know what "primary" or "secondary" means; everything is just voltages. I would make sure I assigned voltage bases to all the buses, then cycle through the buses and check the voltage bases, putting the voltages in separate bins or arrays.

Here is a VBA sub from the SampleDSSDriver.xls spreadsheet we provide as an example that shows one way to cycle through the buses collection in the COM interface and put the voltages on the "VoltSheet" worksheet. You can adapt this to put values from buses with different voltage bases on different worksheets:

```
Public Sub LoadVoltages()

' This Sub loads the per unit complex voltages onto Sheet3 starting in Row 2

    Dim DSSBus As OpenDSSengine.Bus
    Dim iRow As Long, iCol As Long, i As Long, j As Long
    Dim V As Variant
    Dim WorkingSheet As Worksheet

    Set WorkingSheet = Worksheets("VoltSheet") ' Sheet3

    iRow = 2
    For i = 1 To DSSCircuit.NumBuses    ' Cycle through all buses

        Set DSSBus = DSSCircuit.Buses(i)  ' Set i-th bus active

    ' Bus name goes into Column 1
        WorkingSheet.Cells(iRow, 1).Value = DSSCircuit.ActiveBus.Name

    ' Loads pu voltages (complex) at active bus as variant array of doubles
        V = DSSBus.puVoltages

    ' Put values in Variant array into cells in sequence provided by DSS
        iCol = 2
        For j = LBound(V) To UBound(V)
```

```
            WorkingSheet.Cells(iRow, iCol).Value = V(j)
            iCol = iCol + 1
        Next j

       iRow = iRow + 1
    Next i

End Sub
```

Getting Unbalanced Voltages

Question

… I don't understand why sometimes the loads are assumed unbalanced for the number of phases ;through"show powers kVA elem" ,I get the results as follows:
"
Phase kW +j kvar kVA PF
1 6.1 +j 2.5 6.6 0.9234
2 6.4 +j 1.0 6.5 0.9875
3 5.0 +j 1.5 5.2 0.9552
TERMINAL TOTAL .... 17.5 +j 5.1 18.2 0.9600"

The Load definition

:"New Load.load1 Bus1=b108 Phases=3 Conn=Delta Model=1 kV=0.4 kVA=18.2 PF=0.96"

So in the Three-phase balanced system, I get the unbalanced result. I'm quite confused.

Answer

There could be a couple of reasons for this.

1. The voltage to neutral is unbalanced (neutral shift) and the load is defined as Delta
2. I see the load is defined as kVA=30 but is computed to be 18.2. Did you change the load? Or is the voltage very low?

Accessing the COM Interface from C++

… here is a main file that following rdugan's example:

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    CoInitialize(NULL);
    try {
        OpenDSSengine::IDSSPtr DSSObj;
        DSSObj.CreateInstance(__uuidof(OpenDSSengine::DSS));
        if(DSSObj->Start(0)){
            cout<<"openDSS Loaded!"<<endl;
            // Do whatever you want, here we show the version
            cout<<DSSObj->Version<<endl;
        };
    } catch (_com_error e) {
        printf("\n");
        printf("Error: %S\n", e.Description());
        printf("Error: %S\n", e.ErrorMessage());
    }
    cout << "Press anykey to exit.";
    cin.ignore();
    cin.get();
    return 0;
}
```

Following my previous post, here is my example code solving load flow under MS Visual Studio C++
Tested on Windows 7 32bit

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
    CoInitialize(NULL);
    try {
        OpenDSSengine::IDSSPtr DSSObj;
        DSSObj.CreateInstance(__uuidof(OpenDSSengine::DSS));
        OpenDSSengine::ITextPtr DSSText;
        OpenDSSengine::ICircuitPtr DSSCircuit;
        OpenDSSengine::ISolutionPtr DSSSolution;
        if(DSSObj->Start(0)){
            cout<<"openDSS Loaded!"<<endl;
            DSSText = DSSObj->Text;
            DSSCircuit = DSSObj->ActiveCircuit;
            DSSSolution = DSSCircuit->Solution;
            cout<<DSSObj->Version<<endl;
            DSSText->Command="Compile IEEE123Master.dss";
            DSSSolution->Solve();
            DSSText->Command="Show Voltage LN Nodes";
        };
    } catch (_com_error e) {
        printf("\n");
        printf("Error: %S\n", e.Description());
        printf("Error: %S\n", e.ErrorMessage());
    }
    cout << "Press anykey to exit.";
    cin.ignore();
    cin.get();
    return 0;
}
```

--- Tao Cui

How to Do Distribution State Estimation

Put an EnergyMeter at the head of the feeder and then you can place some Sensor objects downstream. You should then populate the Currents properties of the Sensors and the PeakCurrents property of the EnergyMeter to correspond to actual measurements at those locations.

Then either the Estimate command of the AllocateLoads command will perform an estimation by adjusting the estimates of the loads. Note that it may not be able to do this perfectly and it will give up after a couple of tries. It works pretty well on North American feeders with a lot of 1-phase loads (modeled as 1-phase loads) because the algorithm can adjust each phase separately. 3-phase loads are adjusted as balanced 3-phase loads. Sometimes you have to model 3-phase loads a 3 separate 1-phase loads to get a better estimate if the 3-phase loads are unbalanced.

This is a relatively simple algorithm that works pretty well most of the time, but will not work as well on all systems.

Note that the algorithm depends on the LTCs and Voltage regulators to match the voltages. You set the RegControl targets to the measured voltages. The standard bandwidth is usually within 1 percent, which is pretty good. Some users have told me you can define the regulating transformers to have 100s of small taps and set the RegControl band really tight and the program will get really close to actual voltage measurements. I have to warn you that I have never tried that approach, but it could work! 1% is usually good enough for me.

Setting the System Frequency

Question

While doing simulation, at which all places i should change the basefreq for simulating a 50Hz system. I have changed the frequency for vsource, load,line, transformer etc. But in the solution summary , the freq is displayed as 60Hz.

Answer

The easy way is to change the default base frequency

```
set defaultbasefreq=50
```

Then everything is expected in 50 Hz values and you don't have to specify any other frequency. (Unless you enter 60 Hz impedance values for a 50 Hz solution, which you can do.)

This option need be set but once if all you work on is 50 Hz. It is remembered by Windows.

Solving Twice

Question

Just running once, the result shows NOT SOLVED.

Solve again right after the first solve made it SOLVED.

What does it mean? Is it really solved and converged?

Answer

Try increasing the max number of iterations to 50 or something, and see if it solves at first run. I think it converges, but just need more iteration (the default number of iteration is 15)

```
// Code:
Set maxiter=50
// If you are using controls too:
Set maxcontroliter=50
```

You can do Solve twice and it will continue from where you left off.
It is converged? If you think it should be converged, do **Show Converged** and it will create a report of the voltage mismatches at each node.

The other way you can get lack of convergence is with controls fighting each other. Sometimes they will never converge, oscillating between two solutions. You will often find that either power flow solution is reasonable.

Accessing OpenDSS Files by SVN

It has been reported to me that some users are having trouble finding the updated material we are posting on the SourceForge.net site. If you install a SVN client, updates are very simple. Here is how I do it:

I use TortoiseSVN client form http://tortoisesvn.net/downloads.html. For Windows 7, 64-bit, be sure to install the 64-bit version.

1. To download all the OpenDSS files from SourceForge, create a clean directory such as "c:\OpenDSS" or "c:\users\MyUserID\OpenDSS".
2. Right-click on the directory and choose "SVN Checkout"
3. Set the repository URL to http://svn.code.sf.net/p/electricdss/code/trunk/

Thereafter, when you right-click on the OpenDSS folder, or any subfolder, select SVN Update and you'll get all the new files in that folder including the latest beta builds of the program in the Distrib folder. If you have the program installed by the Installer in another file location, you can simply copy the updated EXE and DLL files over the installed versions and you should then be in business with the latest version. Alternatively, if you have appropriate permissions on your computer, you can register the OpenDSSEngine.DLL you are using (X86 version for Excel and, possibly, X64 version for 64-bit Matlab) in the new folder location and it will be automatically updated when you do the SVN update.

That is essentially what I do -- except I have my development version registered, which I copy to the Distrib folder when I'm satisfied it is ready for user beta testing.

--- Roger Dugan

New-User Primer

Last year I asked Jason Sexauer who sometimes answers questions on this forum to write a "primer" for new users -- capturing what he learned as he started to use the program. I have just posted the pdf. You can find it by going to the Code menu above and then to

/trunk/distrib/doc

It's called "OpenDSSPrimer.pdf"

Or, if you have SVN linked to repository, just right-click and update.

I think you'll find the document quite useful.

--- Roger Dugan

Getting Solution Time

Question

Hello,
i want to take a comparison of the DSS and other tools. So I need to know the calculation time the DSS uses to get the solution. But I have no idea to set the case. Can anyone give me some advise?
Thanks!

Answer

Wes gives a good answer. I generally get this question from students who are looking for differences in solution times for a single snapshot power flow solution of relatively small circuits. It is difficult for learn anything meaningful from such tests. It is also difficult because of Windows. You never really know what it is doing.

As Wes suggests, you can do a large number of time steps and take the average. However, you need to know that we "cheat" on time sequential simulations so that they run faster. We use a simple fixed point iteration on the system Y matrix that is quick if Y is constant. We do not change the Y matrix any more than we absolutely have to. While we might rebuild the Y matrix several times on the initial solution as capacitors and regulators switch, it gets rebuilt rarely on most annual simulations. When Y doesn't change, we don't refactor, so that solution goes very quickly. In a typical utility power distribution simulation following the typical loadshape, there are only 2 iterations required to advance from one time step to the next -- one to get to the next solution and one to prove you got there. That is very quick and skews the average timing.

The other thing to take into consideration is that if there are Energymeter objects in the circuit, the program pauses at the end of each time step to write solutions to disk. We have found this can take longer than the actual calculations for the power flow solution. You are at the mercy of Windows for timing this.

You also have to decide when you are going to start counting the solution time. With OpenDSS there is almost always a little (or a lot) script processing before getting down to the actual number crunching.
A useful metric might be the average time per iteration. But again, not all iterations on OpenDSS are the same. Be sure to use a sufficiently large circuit to get some useful numbers. This can be difficult with external timing because modern computers do things so fast. For example, my Lenovo W520 with a SSD loads and executes 65 iterations on the IEEE 8500-Node Test feeder in less than 1 s. There are 5 Y matrix rebuilds in that time due to regulators and capacitors changing state. And if I do it again immediately, it is slightly quicker, presumably due to caching.

So, it is difficult to compare different tools based on speed in a meaningful way. If you wanted to compare different algorithms, you might consider programming the different algorithms on the same platform and then comparing times. Sandoval Carneiro, Jr, and is colleagues in Brazil did this a few years back and published a paper. I would not use MATLAB for this because some things are fast and some operations are very slow. Use a compiled language that is able to handle such things as complex arithmetic as efficiently as possible.

Problem Linking Matlab to OpenDSS

Question

I am working on linking OpenDSS to MATLAB, I think I have registered the COM server. MATLAB does respond to the command "actxserver('OpenDSSengine.DSS')". Then I tried to follow the codes from "OpenDSS/DOC/OpenDSS Time-based Simulations in Matlab.pdf." The problem is when I run this line

"result = dss_Command('compile master.dss')";

MATLAB told me "Undefined function 'dss_Command' for input arguments of type 'char'."

Which probably means MATLAB doesn't recognize "dss_Command". So what should I do to solve this? Thank you.

Answer

Here is one possible approach…everywhere you see dss_Command in that document try replacing it with DSSText.Command. The following code instantiates the OpenDSSengine COM server:

```
function [Start,Obj,Text] = DSSStartup(mydir)
    % Function for starting up the DSS
    % make sure we are in the proper directory
    cd(mydir);
    %
    %instantiate the DSS Object
    Obj = actxserver('OpenDSSEngine.DSS');
    %
    %Start the DSS.   Only needs to be executed the first time w/in a
    %Matlab session
    Start = Obj.Start(0);
    % Define the text interface
    Text = Obj.Text;
```

You could then call it in your main matlab function where you are working with the OpenDSS COM server. Here is some example code:

```
    [DSSStartOK, DSSObj, DSSText] = DSSStartup(myDir);
    %Check to see if the DSS started properly
    if DSSStartOK
        %Compile the DSS circuit script
        DSSText.Command = 'compile master.dss';
    % get an interface to the active circuit called "DSSCircuit"
        DSSCircuit = DSSObj.ActiveCircuit;
        % do your other stuff here
```

See the examples folder for complete examples of using the OpenDSS COM server via a number of COM clients, including Matlab.

Wes Sunderman

Linking C# to OpenDSS

Here is a snippet of C# code that mimics the VBA code in SampleDSSDriver.XLS in the Examples folder. Courtesy of Lenoir Laurent. Hope this helps

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Reflection;
using System.Diagnostics;
using OpenDSSengine;
using Microsoft.Office.Interop.Excel;
namespace DemoOpenDSS
{
  public partial class DemoOpenDSSUI : Form
  {
    public DSS DSSobj;
    public Text DSSText;
    public Circuit DSSCircuit;
    public Solution DSSSolution;
    public CtrlQueue DSSControlQueue;
    public static _Application application = new Microsoft.Office.Interop.Excel.Application();
    public static string pathApp = Path.GetDirectoryName(
Assembly.GetExecutingAssembly().Location );
    public static string filePath = pathApp + @"\results.xlsx";
    public _Workbook workbook = application.Workbooks.Open( filePath, Type.Missing,
Type.Missing, Type.Missing,
                              Type.Missing, Type.Missing, Type.Missing, Type.Missing,
                              Type.Missing, Type.Missing, Type.Missing, Type.Missing,
                              Type.Missing, Type.Missing, Type.Missing );


    public DemoOpenDSSUI()
    {
      InitializeComponent();
    }
    private void starDSS_Click( object sender, EventArgs e )
    {
      DSSobj = new DSS();
      if ( !( DSSobj.Start( 0 ) ) )
      {
        MessageBox.Show( "DSS failed to start" );
```

```
            }
            else
            {
               MessageBox.Show( "DSS started sucessfully" );
               DSSText = DSSobj.Text;
            }
         }
         private void loadCircuit_Click( object sender, EventArgs e )
         {
            string circuitName = textBox1.Text;

            DSSText.Command = "clear";
            DSSText.Command = "compile " + "(" + circuitName +")";
            DSSCircuit = DSSobj.ActiveCircuit;
            DSSSolution = DSSCircuit.Solution;
            DSSControlQueue = DSSCircuit.CtrlQueue;
            MessageBox.Show( "Circuit Loaded" );
         }
      ….snip …
```

Obfuscate Command To Protect Propietary Circuit Data

One other point…if you are concerned about maintaining anonymity of the location of the circuit that you are modeling, you can execute the command:

```
Obfuscate
```

which will, according to the help change bus and circuit element names to generic values to remove identifying names. Generally, you will follow this command immediately by a

"Save Circuit Dir=MyDirName" command.

Be sure that you do NOT save the obfuscated circuit in the same subdirectory as your existing circuit as it may over-write one or more files. Also, you likely will want to look at the saved circuit files and double-check that things like the substation transformer, circuit name, and maybe some of the monitor names do not contain any identifying information.

This is one way that we exchange files, maintaining anonymity of the circuit models - with third parties.

Thanks,
Wes

Channels in Monitors Interface

Question:

Hey,
this might be a very simple question, but i'd appreciate it if any one can clear it up for me or at least direct me to where can find answers for it.
what are exactly the "channels" in Monitors? and how to define each channel and check it's output and maybe plot it?
I have read through the documentation and some examples but i couldn't clearly understand it.
thank you

Answer:

A channel can be likened to a channel on an oscilloscope. Channel contents are defined by the Monitor mode and the number of phases of the device to which the monitor is attached.

You can plot it using a script command where channels are referred to by number. You can also provide a base for the channel so that you get numbers in per unit or some other quantity.From the online Help for Plot commands:

```
Plot Type=Monitor Object=MyMonitor Channels=[1, 3, 5] Bases=[2400 2400 2400]
```

It you choose Plot Monitor from the menu of OpenDSS.EXE, it will prompt you through setting up this command.

Channels Vs Bytestream

Question

Answer 1

Well its simple, change your code for the following:

```
      DSSCircuit.monitors.Name='M1'; %Selects the monitor M1
      Freqs=DSSCircuit.monitors.ByteStream; %Request the Bytestream
      iMonitorDataSize= typecast(Freqs(9:12),'int32'); % To adjust the matrix
      VIMonitor = typecast(Freqs(273:end),'single') %Adjusts the content
      VIMonitor1 = reshape(VIMonitor, iMonitorDataSize+2, [])'
      DSSCircuit.monitors.Name='M2'; %Selects the monitor M2
      Freqs=DSSCircuit.monitors.ByteStream; %Request the Bytestream
      iMonitorDataSize= typecast(Freqs(9:12),'int32'); % To adjust the matrix
      VIMonitor = typecast(Freqs(273:end),'single') %Adjusts the content
      VIMonitor2 = reshape(VIMonitor, iMonitorDataSize+2, [])'
```

That's it. I hope it helps

Regards
Davis Montenegro

Answer 2

I am always amazed at what the Matlabbers can come up with!

There are now other ways to get the data on the bytestream. For example, this little VBA subroutine will read all the channels from all the monitors in the circuit and do something with them (you supply the DoSomething routine) ..

```
Public Sub ReadMultipleMonitors()

   Dim DSSMonitors As OpenDSSengine.Monitors
   Dim iMon As Long, iChannel As Long
   Dim ChannelData As Variant
   Dim TimeValues As Variant
   Dim strHeader As Variant

   Set DSSMonitors = DSSCircuit.Monitors

   DSSMonitors.SaveAll  ' Make sure the bytestream is completely populated

   iMon = DSSMonitors.First
   Do While iMon > 0
        strHeader = DSSMonitors.Header

        ' Get all time values -- Only have to do this once
        TimeValues = DSSMonitors.dblHour

        ' Now Process each channel
        For iChannel = 1 To DSSMonitors.NumChannels
            ChannelData = DSSMonitors.Channel(iChannel) ' This gets all values in this
channel
            ' now do something with the time and channel data, for example
            ' Dim iSample as Long
            ' For iSample = LBound(TimeValues) to UBound(TimeValues)
            '     iOK = DoSomething(strHeader(iChannel-1), Timevalues(iSample),
ChannelData(iSample))
            ' Next iSample
        Next iChannel

       iMon = DSSMonitors.Next
   Loop

End Sub
```

Tech Note on Monitors Interface from the Wiki

Updates to Monitors Interface

Users have been reporting difficulties in decoding the ByteStream property. This is an exact copy of the in-memory file stream used by the Monitor objects to save the Monitor samples. The file stream technique is used because it has been found to be much faster than writing to disk. In some languages it is fairly straightforward to write a subroutine that decodes the variant array quickly. However, it doesn't seem to work as well with other languages.

The ByteStream property was added to the COM interface when users found it inefficient to use the Export Monitor command to write a CSV file that read back into their application. There is probably no faster way to move the Monitor file stream across the COM interface, but some languages are slow at looping through the ByteStream array to decode it. (The code is also rather cryptic in languages like Matlab.)

Therefore, a different approach has been made available. The decoding is done on the OpenDSS side and the data arrays are passed back as arrays of doubles. This will be much faster and more efficient. Several read-only properties have been added to the Monitors interface. They are described below.

Note: To guarantee the **ByteStream** and related properties are populated, the **SaveAll** method should be executed so that the temporary Monitor buffer is dumped to the Monitor object's memory stream. This is not necessary for standard solution modes such as Daily, Yearly, or Dutycycle. However, it is necessary if you simply do a snapshot solution followed by a Sample command. The Show and Export commands automatically save the buffer. If the ByteStream or Channel properties do not contain the expected number of samples, execute the Saveall function from the Monitors interface.

Header Property

This is a variant array of strings containing the names of the channels. The array contains basically the same contents as the first record in the CSV file created by the Export command, except the first two columns are deleted.

The first two columns are different for sequential-time simulations and harmonic simulations. The new properties return these values differently:

dblHour Property

This is the time array that corresponds to the channel values, in units of hours, as a variant array of doubles. This combines the first two channels in the monitor file stream to return time as a single value. Most users should find this more convenient.

This property is populated only for sequential-time simulations. If the solution was a harmonics solution, a single value is returned for this property. You can check the upper bound of the variant array, which will be zero.

dblFreq Property

This is the frequency array that corresponds to the channel values. It is populated only if the solution was in Harmonics mode.

There are two channels in the Monitor object file stream: Frequency, Hz, and harmonic number. Only the frequency is returned in this array.

NumChannels Property

This returns an integer equal to the number of data channels in the present file stream. Should be the same as the number of strings in the Header Property.

Same as RecordSize Property.

Channel(Index) Property

This property returns the values of the channel indicated by Index as a variant array of doubles. The valid range for Index is 1..NumChannels.

The number of elements in the array should be the same as the SampleCount property. Of course, you should check the lower and upper bounds of the variant array when looping through the array. Generally, OpenDSS returns variant arrays with a range of 0..SampleCount unless there is an error.

Recordsize Property

Integer value of the Recordsize field in the Monitor file stream. Same as NumChannels.

FileVersion Property

Integer value of the FileVersion Property in the Monitor file stream. For future use in case the file format changes.

Example

This is a VBA subroutine from Excel that illustrates the use of the new properties. You can load this into Excel and then using the COM interface execute a script from Excel containing Monitor objects. Run some sort of sequential time simulation. While OpenDSS is still running (DSSObj is still valid), execute this subroutine.

(Do not clear the circuit or set DSSobj=Nothing before running it -- that will dispose of the Monitor file streams; just solve a bunch of time steps and then run this.)

It will create a file in the same folder as the data with some excerpts from the Monitors. Use this as a pattern for other languages.

```
    Public Sub TestMonitorInterface()

    Dim DSSMonitors As OpenDSSengine.Monitors

    ' set a variable for easy access to the monitors interface
    Set DSSMonitors = DSSobj.ActiveCircuit.Monitors

    ' Make a couple of variants to receive arrays
    Dim V As Variant, V2 As Variant

    Dim i As Long, imon As Long

    Open "MonitorInterfaceTest.Txt" For Output As #1

    ' Print a list of the monitors in the circuit
    V = DSSMonitors.AllNames
    Print #1, "Number of Monitors = "; DSSMonitors.Count
    For i = LBound(V) To UBound(V)
         Print #1, "Monitor ("; CStr(i); ")="; V(i)
    Next i

    ' go through each monitor and print out some stuff
    imon = DSSMonitors.First
    Do While imon > 0
        Print #1, " "
        Print #1, "Monitor=", DSSMonitors.Name

          ' get the names of the channels
        V = DSSMonitors.Header
        Print #1, "Header:"
        For i = LBound(V) To UBound(V)
            Print #1, V(i); ", ";
        Next i
        Print #1, " "

        Print #1, "NumChannels="; DSSMonitors.NumChannels; " SampleCount=";
DSSMonitors.SampleCount

        V = DSSMonitors.dblHour
        V2 = DSSMonitors.Channel(1)

        ' Excerpt from the channels ...
        Print #1, " "
        Print #1, "--------------- Excerpt -------------------"
        Print #1, " time and channel 1"
        Print #1, " "
        For i = 1 To 20
            Print #1, V(i); ", ";
            Print #1, V2(i)
        Next i
        Print #1, " "
        Print #1, "--------------- Excerpt -------------------"
        Print #1, " "

        imon = DSSMonitors.Next
    Loop

    Close #1

    End Sub
```