

# OpenDSS Type Library Documentation

August 18, 2015

Version 7.6.4.65

## Enumerations

### enum MonitorModes

**dssVI = 0**, Monitor records Voltage and Current at the terminal (Default)  
**dssPower = 1**, Monitor records kW, kvar or kVA, angle values, etc. at the terminal to which it is connected.  
**dssSequence = 16**, Reports the monitored quantities as sequence quantities  
**dssMagnitude = 32**, Reports the monitored quantities in Magnitude Only  
**dssPosOnly = 64**, Reports the Positive Seq only or avg of all phases  
**dssTaps = 2**, For monitoring Regulator and Transformer taps  
**dssStates = 3** For monitoring State Variables (for PC Elements only)

### enum SolveModes

**dssSnapShot = 0**, Solve a single snapshot power flow  
**dssDutyCycle = 6**, Solve following Duty Cycle load shapes  
**dssDirect = 7**, Solve direct (forced admittance model)  
**dssDaily = 1**, Solve following Daily load shapes  
**dssMonte1 = 3**, Monte Carlo Mode 1  
**dssMonte2 = 10**, Monte Carlo Mode 2  
**dssMonte3 = 11**, Monte Carlo Mode 3  
**dssFaultStudy = 9**, Fault study at all buses  
**dssYearly = 2**, Solve following Yearly load shapes  
**dssMonteFault = 8**, Monte carlo Fault Study  
**dssPeakDay = 5**, Solves for Peak Day using Daily load curve  
**dssLD1 = 4**, Load-duration Mode 1  
**dssLD2 = 12**, Load-Duration Mode 2  
**dssAutoAdd = 13**, Auto add generators or capacitors  
**dssHarmonic = 15**, (no Help string available)  
**dssDynamic = 14** (no Help string available)

## enum Options

**dssPowerFlow = 1**, *Power Flow load model option*  
**dssAdmittance = 2**, *Admittance load model option*  
**dssNormalSolve = 0**, *Solution algorithm option - Normal solution mode*  
**dssNewtonSolve = 1**, *Solution algorithm option - Newton solution*  
**dssStatic = 0**, *Control Mode option - Static*  
**dssEvent = 1**, *Control Mode Option - Event driven solution mode*  
**dssTime = 2**, *Control mode option - Time driven mode*  
**dssMultiphase = 0**, *Circuit model is multiphase (default*  
**dssPositiveSeq = 1**, *Circuit model is positive sequence model only*  
**dssGaussian = 1**, *Random mode = Gaussian*  
**dssUniform = 2**, *Random mode = Uniform*  
**dssLogNormal = 3**, *Random Mode = Log normal*  
**dssAddGen = 1**, *Add generators in AutoAdd mode (AddType*  
**dssAddCap = 2**, *Add capacitors in AutoAdd mode (Addtype*  
**dssControlOFF = -1** *Control Mode OFF = -1*

## enum CapControlModes

**dssCapControlVoltage = 1**, *voltage control, ON and OFF settings on the PT secondary base*  
**dssCapControlKVAR = 2**, *kVAR control, ON and OFF settings on PT / CT base*  
**dssCapControlCurrent = 0**, *Current control, ON and OFF settings on CT secondary*  
**dssCapControlPF = 4**, *ON and OFF settings are power factor, negative for leading*  
**dssCapControlTime = 3** *Time control, ON and OFF settings are seconds from midnight*

## enum ActionCodes

**dssActionNone = 0**, *No action*  
**dssActionOpen = 1**, *Open a switch*  
**dssActionClose = 2**, *Close a switch*  
**dssActionReset = 3**, *Reset to the shelf state (unlocked, closed for a switch*  
**dssActionLock = 4**, *Lock a switch, prventing both manual and automatic operation*  
**dssActionUnlock = 5**, *Unlock a switch, permitting both manual and automatic operation*  
**dssActionTapUp = 6**, *Move a regulator tap up*  
**dssActionTapDown = 7** *Move a regulator tap down*

## enum LoadStatus

**dssLoadVariable = 0**, *(no Help string available)*

**dssLoadFixed = 1,** *(no Help string available)*  
**dssLoadExempt = 2** *(no Help string available)*

## **enum LoadModels**

**dssLoadConstPQ = 1,** *(no Help string available)*  
**dssLoadConstZ = 2,** *(no Help string available)*  
**dssLoadMotor = 3,** *(no Help string available)*  
**dssLoadCVR = 4,** *(no Help string available)*  
**dssLoadConstI = 5,** *(no Help string available)*  
**dssLoadConstPFixedQ = 6,** *(no Help string available)*  
**dssLoadConstPFixedX = 7,** *(no Help string available)*  
**dssLoadZIPV = 8** *(no Help string available)*

## **enum LineUnits**

**dssLineUnitsNone = 0,** *No line length unit.*  
**dssLineUnitsMiles = 1,** *Line length units in miles.*  
**dssLineUnitSkFt = 2,** *Line length units are in thousand feet.*  
**dssLineUnitskm = 3,** *Line length units are km.*  
**dssLineUnitsmeter = 4,** *Line length units are meters.*  
**dssLineUnitsft = 5,** *Line units in feet.*  
**dssLineUnitsinch = 6,** *Line length units are inches.*  
**dssLineUnitscm = 7,** *Line units are cm.*  
**dssLineUnitsmm = 8,** *Line length units are mm.*  
**dssLineUnitsMaxnum = 9** *Maximum number of line units constants.*

# Interfaces

## Text Interface

**Command** [out, retval] Type: BSTR\* Command; *[Property (get)];*

'value = Command' -- *Input command string for the DSS.*

**Command** [in] Type: BSTR Command; *[Property (put)];*

'Command = value' -- *Input command string for the DSS.*

**Result** [out, retval] Type: BSTR\* Result; *[Property (get)];*

'value = Result' -- *Result string for the last command.*

## DSSProperty Interface

**Name** [out, retval] Type: BSTR\* Name; *[Property (get)];*

'value = Name' -- *Name of Property*

**Description** [out, retval] Type: BSTR\* Description; *[Property (get)];*

'value = Description' -- *Description of the property.*

**Val** [out, retval] Type: BSTR\* Value; *[Property (get)];*

'value = Val' -- *(no Help string available)*

**Val** [in] Type: BSTR Value; *[Property (put)];*

'Val = value' -- *(no Help string available)*

## CktElement Interface

**Name** [out, retval] Type: BSTR\* Value; *[Property (get)];*

'value = Name' -- *Full Name of Active Circuit Element*

**NumTerminals** [out, retval] Type: long\* Value; *[Property (get)];*

'value = NumTerminals' -- *Number of Terminals this Circuit Element*

**NumConductors** [out, retval] Type: long\* Value; *[Property (get)];*

'value = NumConductors' -- *Number of Conductors per Terminal*

**NumPhases** [out, retval] Type: long\* Value; *[Property (get)];*

'value = NumPhases' -- *Number of Phases*

**BusNames** [out, retval] Type: VARIANT\* Value; *[Property (get)];*

'value = BusNames' -- *Variant array of strings. Get Bus definitions to which each terminal is connected. 0-based array.*

**BusNames** [in] Type: VARIANT Value; *[Property (put)];*

'BusNames = value' -- *Variant array of strings. Set Bus definitions for each terminal is connected.*

**Properties** [in] Type: VARIANT Indx, [out, retval] Type: IDSSProperty\*\* Value; *[Property*

```

(get));
'value = Properties ' -- Collection of Properties for this Circuit Element (0 based index, if numeric
Voltages [out, retval] Type: VARIANT* Value; [Property (get)];
'value = Voltages ' -- Complex array of voltages at terminals
Currents [out, retval] Type: VARIANT* Value; [Property (get)];
'value = Currents ' -- Complex array of currents into each conductor of each terminal
Powers [out, retval] Type: VARIANT* Value; [Property (get)];
'value = Powers ' -- Complex array of powers into each conductor of each terminal
Losses [out, retval] Type: VARIANT* Value; [Property (get)];
'value = Losses ' -- Total losses in the element: two-element complex array
PhaseLosses [out, retval] Type: VARIANT* Value; [Property (get)];
'value = PhaseLosses ' -- Complex array of losses by phase
SeqVoltages [out, retval] Type: VARIANT* Value; [Property (get)];
'value = SeqVoltages ' -- Double array of symmetrical component voltages at each 3-phase
terminal
SeqCurrents [out, retval] Type: VARIANT* Value; [Property (get)];
'value = SeqCurrents ' -- Double array of symmetrical component currents into each 3-phase
terminal
SeqPowers [out, retval] Type: VARIANT* Value; [Property (get)];
'value = SeqPowers ' -- Double array of sequence powers into each 3-phase terminal
Enabled [out, retval] Type: VARIANT_BOOL* Value; [Property (get)];
'value = Enabled ' -- Boolean indicating that element is currently in the circuit.
Enabled [in] Type: VARIANT_BOOL Value; [Property (put)];
' Enabled = value' -- Boolean indicating that element is currently in the circuit.
NormalAmps [out, retval] Type: double* Value; [Property (get)];
'value = NormalAmps ' -- Normal ampere rating for PD Elements
NormalAmps [in] Type: double Value; [Property (put)];
' NormalAmps = value' -- Normal ampere rating
EmergAmps [out, retval] Type: double* Value; [Property (get)];
'value = EmergAmps ' -- Emergency Ampere Rating for PD elements
EmergAmps [in] Type: double Value; [Property (put)];
' EmergAmps = value' -- Emergency Ampere Rating
Open [in] Type: long Term, [in] Type: long Phs; [Method];
' Open(arg list) ' -- Open the specified terminal and phase, if non-zero. Else all conductors at
terminal.
Close [in] Type: long Term, [in] Type: long Phs; [Method];
' Close(arg list) ' -- Close the specified terminal and phase, if non-zero. Else all conductors at
terminal.
IsOpen [in] Type: long Term, [in] Type: long Phs, [out, retval] Type: VARIANT_BOOL* Value;

```

[Method];

'IsOpen(arg list)' -- Boolean indicating if the specified terminal and, optionally, phase is open.

**NumProperties** [out, retval] Type: long\* Value; [Property (get)];

'value = NumProperties' -- Number of Properties this Circuit Element.

**AllPropertyNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllPropertyNames' -- Variant array containing all property names of the active device.

**Residuals** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Residuals' -- Residual currents for each terminal: (mag, angle

**Yprim** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Yprim' -- YPrim matrix, column order, complex numbers (paired

**DisplayName** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = DisplayName' -- Display name of the object (not necessarily unique

**DisplayName** [in] Type: BSTR Value; [Property (put)];

'DisplayName = value' -- Display name of the object (not necessarily unique

**Handle** [out, retval] Type: long\* Value; [Property (get)];

'value = Handle' -- Pointer to this object

**GUID** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = GUID' -- globally unique identifier for this object

**HasSwitchControl** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = HasSwitchControl' -- This element has a SwtControl attached.

**HasVoltControl** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = HasVoltControl' -- This element has a CapControl or RegControl attached.

**EnergyMeter** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = EnergyMeter' -- Name of the Energy Meter this element is assigned to.

**Controller** [in] Type: long idx, [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Controller' -- Full name of the i-th controller attached to this element. Ex: str = Controller

(2

**CplxSeqVoltages** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = CplxSeqVoltages' -- Complex double array of Sequence Voltage for all terminals of active circuit element.

**CplxSeqCurrents** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = CplxSeqCurrents' -- Complex double array of Sequence Currents for all conductors of all terminals of active circuit element.

**AllVariableNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllVariableNames' -- Variant array of strings listing all the published variable names, if a PCElement. Otherwise, null string.

**AllVariableValues** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllVariableValues' -- Variant array of doubles. Values of state variables of active element if PC element.

**Variable** [in] Type: BSTR MyVarName, [out] Type: long\* Code, [out, retval] Type: double\* Value; [Property (get)];  
'value = Variable ' -- For PCElement, get the value of a variable by name. If Code>0 Then no variable by this name or not a PCElement.

**Variablei** [in] Type: long Idx, [out] Type: long\* Code, [out, retval] Type: double\* Value; [Property (get)];  
'value = Variablei ' -- For PCElement, get the value of a variable by integer index.

**NodeOrder** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = NodeOrder ' -- Variant array of integer containing the node numbers (representing phases, for example

**HasOCPDevice** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = HasOCPDevice ' -- True if a recloser, relay, or fuse controlling this ckt element. OCP = Overcurrent Protection

**NumControls** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumControls ' -- Number of controls connected to this device. Use to determine valid range for index into Controller array.

**OCPDevIndex** [out, retval] Type: long\* Value; [Property (get)];  
'value = OCPDevIndex ' -- Index into Controller list of OCP Device controlling this CktElement

**OCPDevType** [out, retval] Type: long\* Value; [Property (get)];  
'value = OCPDevType ' -- 0=None; 1=Fuse; 2=Recloser; 3=Relay; Type of OCP controller device

**CurrentsMagAng** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = CurrentsMagAng ' -- Currents in magnitude, angle format as a variant array of doubles.

**VoltagesMagAng** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = VoltagesMagAng ' -- Voltages at each conductor in magnitude, angle form as variant array of doubles.

## Error Interface

**Number** [out, retval] Type: long\* Number; [Property (get)];  
'value = Number ' -- Error Number

**Description** [out, retval] Type: BSTR\* Description; [Property (get)];  
'value = Description ' -- Description of error for last operation

## Circuit Interface

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- Name of the active circuit.

**NumCktElements** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumCktElements ' -- Number of CktElements in the circuit.

**NumBuses** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumBuses ' -- Total number of Buses in the circuit.

**NumNodes** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumNodes ' -- Total number of nodes in the circuit.

**Buses** [in] Type: VARIANT Index, [out, retval] Type: IBus\*\* Value; [Property (get)];  
'value = Buses ' -- Collection of Buses in the circuit. Index may be string or integer index (0 based

**CktElements** [in] Type: VARIANT Idx, [out, retval] Type: ICktElement\*\* Value; [Property (get)];  
'value = CktElements ' -- Collection of CktElements in Circuit

**Losses** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Losses ' -- Total losses in active circuit, complex number (two-element array of double

**LineLosses** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = LineLosses ' -- Complex total line losses in the circuit

**SubstationLosses** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = SubstationLosses ' -- Complex losses in all transformers designated to substations.

**TotalPower** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = TotalPower ' -- Total power, watts delivered to the circuit

**AllBusVolts** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllBusVolts ' -- Complex array of all bus, node voltages from most recent solution

**AllBusVmag** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllBusVmag ' -- Array of magnitudes (doubles

**AllElementNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllElementNames ' -- Vaiant array of strings containing Full Name of all elements.

**ActiveElement** [out, retval] Type: ICktElement\*\* Value; [Property (get)];  
'value = ActiveElement ' -- Return an interface to the active circuit element

**Disable** [in] Type: BSTR Name; [Method];  
' Disable(arg list) ' -- Disable a circuit element by name (removes from circuit but leave in database

**Enable** [in] Type: BSTR Name; [Method];  
' Enable(arg list) ' -- Activate (enable

**Solution** [out, retval] Type: ISolution\*\* Value; [Property (get)];  
'value = Solution ' -- Return an interface to the Solution object.

**ActiveBus** [out, retval] Type: IBus\*\* Value; [Property (get)];  
'value = ActiveBus ' -- Return an interface to the active bus.

**FirstPCElement** [out, retval] Type: long\* Value; [Method];  
' FirstPCElement(arg list) ' -- Sets the first Power Conversion (PC

**NextPCElement** [out, retval] Type: long\* Value; [Method];  
' NextPCElement(arg list) ' -- Gets next PC Element. Returns 0 if no more.

**FirstPDElement** [out, retval] Type: long\* Value; [Method];



' FirstPDElement(*arg list*) ' -- Sets the first Power Delivery (PD

**NextPDElement** [out, retval] Type: long\* Value; [Method];

' NextPDElement(*arg list*) ' -- Gets next PD Element. Returns 0 if no more.

**AllBusNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllBusNames ' -- Array of strings containing names of all buses in circuit (see AllNodeNames

**AllElementLosses** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllElementLosses ' -- Array of total losses (complex

**Sample** [void; [Method];

' Sample(*arg list*) ' -- Force all Meters and Monitors to take a sample.

**SaveSample** [void; [Method];

' SaveSample(*arg list*) ' -- Force all meters and monitors to save their current buffers.

**Monitors** [out, retval] Type: IMonitors\*\* Value; [Property (get)];

'value = Monitors ' -- Returns interface to Monitors collection.

**Meters** [out, retval] Type: IMeters\*\* Value; [Property (get)];

'value = Meters ' -- Returns interface to Meters (EnergyMeter

**Generators** [out, retval] Type: IGenerators\*\* Value; [Property (get)];

'value = Generators ' -- Returns a Generators Object interface

**Settings** [out, retval] Type: ISettings\*\* Value; [Property (get)];

'value = Settings ' -- Returns interface to Settings interface.

**Lines** [out, retval] Type: ILines\*\* Value; [Property (get)];

'value = Lines ' -- Returns Interface to Lines collection.

**SetActiveElement** [in] Type: BSTR FullName, [out, retval] Type: long\* Value; [Method];

' SetActiveElement(*arg list*) ' -- Sets the Active Circuit Element using the full object name (e.g. \i0

**Capacity** [in] Type: double Start, [in] Type: double Increment, [out, retval] Type: double\* Value; [Method];

' Capacity(*arg list*) ' -- (no Help string available)

**SetActiveBus** [in] Type: BSTR BusName, [out, retval] Type: long\* Value; [Method];

' SetActiveBus(*arg list*) ' -- Sets Active bus by name. Ignores node list. Returns bus index (zero based

**SetActiveBusi** [in] Type: long BusIndex, [out, retval] Type: long\* Value; [Method];

' SetActiveBusi(*arg list*) ' -- Sets ActiveBus by Integer value. 0-based index compatible with SetActiveBus return value and AllBusNames indexing. Returns 0 if OK.

**AllBusVmagPu** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllBusVmagPu ' -- Double Array of all bus voltages (each node

**AllNodeNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeNames ' -- Variant array of strings containing full name of each node in system in same order as returned by AllBusVolts, etc.

**SystemY** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = SystemY ' -- *System Y matrix (after a solution has been performed*

**CtrlQueue** [out, retval] Type: ICtrlQueue\*\* Value; [Property (get)];

'value = CtrlQueue ' -- *Interface to the main Control Queue*

**AllBusDistances** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllBusDistances ' -- *Returns distance from each bus to parent EnergyMeter. Corresponds to sequence in AllBusNames.*

**AllNodeDistances** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeDistances ' -- *Returns an array of distances from parent EnergyMeter for each Node. Corresponds to AllBusVMag sequence.*

**AllNodeVmagByPhase** [in] Type: long Phase, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeVmagByPhase ' -- *Returns Array of doubles represent voltage magnitudes for nodes on the specified phase.*

**AllNodeVmagPUByPhase** [in] Type: long Phase, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeVmagPUByPhase ' -- *Returns array of per unit voltage magnitudes for each node by phase*

**AllNodeDistancesByPhase** [in] Type: long Phase, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeDistancesByPhase ' -- *Returns an array of doubles representing the distances to parent EnergyMeter. Sequence of array corresponds to other node ByPhase properties.*

**AllNodeNamesByPhase** [in] Type: long Phase, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNodeNamesByPhase ' -- *Return variant array of strings of the node names for the By Phase criteria. Sequence corresponds to other ByPhase properties.*

**Loads** [out, retval] Type: ILoads\*\* Value; [Property (get)];

'value = Loads ' -- *Returns interface to Load element interface*

**FirstElement** [out, retval] Type: long\* Value; [Method];

' FirstElement(arg list) ' -- *Sets First element of active class to be the Active element in the active circuit. Returns 0 if none.*

**NextElement** [out, retval] Type: long\* Value; [Method];

' NextElement(arg list) ' -- *Sets the next element of the active class to be the active element in the active circuit. Returns 0 if no more elements.*

**SetActiveClass** [in] Type: BSTR ClassName, [out, retval] Type: long\* Value; [Method];

' SetActiveClass(arg list) ' -- *Sets the active class by name. Use FirstElement, NextElement to iterate through the class. Returns -1 if fails.*

**ActiveDSSElement** [out, retval] Type: IDSSElement\*\* Value; [Property (get)];

'value = ActiveDSSElement ' -- *Returns Interface to the Active DSS object, which could be either a circuit element or a general DSS element.*

**ActiveCktElement** [out, retval] Type: ICktElement\*\* Value; [Property (get)];  
'value = ActiveCktElement ' -- Returns interface to the Active Circuit element (same as ActiveElement

**ActiveClass** [out, retval] Type: IActiveClass\*\* Value; [Property (get)];  
'value = ActiveClass ' -- Returns interface to active class.

**Transformers** [out, retval] Type: ITransformers\*\* Value; [Property (get)];  
'value = Transformers ' -- Returns interface to Transformers collection

**SwtControls** [out, retval] Type: ISwtControls\*\* Value; [Property (get)];  
'value = SwtControls ' -- Returns interface to SwtControls collection.

**CapControls** [out, retval] Type: ICapControls\*\* Value; [Property (get)];  
'value = CapControls ' -- Returns interface to CapControls collection

**RegControls** [out, retval] Type: IRegControls\*\* Value; [Property (get)];  
'value = RegControls ' -- Returns interface to RegControls collection

**Capacitors** [out, retval] Type: ICapacitors\*\* Value; [Property (get)];  
'value = Capacitors ' -- Interface to the active circuit's Capacitors collection.

**Topology** [out, retval] Type: ITopology\*\* Value; [Property (get)];  
'value = Topology ' -- Interface to the active circuit's topology object.

**Sensors** [out, retval] Type: ISensors\*\* Value; [Property (get)];  
'value = Sensors ' -- Interface to Sensors in the Active Circuit.

**UpdateStorage** [void; [Method];  
'UpdateStorage(arg list) ' -- Forces update to all storage classes. Typically done after a solution. Done automatically in intrinsic solution modes.

**ParentPDElement** [out, retval] Type: long\* Value; [Property (get)];  
'value = ParentPDElement ' -- Sets Parent PD element, if any, to be the active circuit element and returns index>0; Returns 0 if it fails or not applicable.

**XYCurves** [out, retval] Type: IXYCurves\*\* Value; [Property (get)];  
'value = XYCurves ' -- Interface to XYCurves in active circuit.

**PDElements** [out, retval] Type: IPDElements\*\* Value; [Property (get)];  
'value = PDElements ' -- Interface to PDElements collection

**Reclosers** [out, retval] Type: IReclosers\*\* Value; [Property (get)];  
'value = Reclosers ' -- (no Help string available)

**Relays** [out, retval] Type: IRelays\*\* Value; [Property (get)];  
'value = Relays ' -- (no Help string available)

**LoadShapes** [out, retval] Type: ILoadShapes\*\* Value; [Property (get)];  
'value = LoadShapes ' -- Interface to OpenDSS Load shapes currently defined.

**Fuses** [out, retval] Type: IFuses\*\* Value; [Property (get)];  
'value = Fuses ' -- Return interface to Fuses

**ISources** [out, retval] Type: IISources\*\* Value; [Property (get)];  
'value = ISources ' -- Interface to ISOURCE devices

**YNodeVarray** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = YNodeVarray ' -- Complex array of actual node voltages in same order as SystemY matrix.

**EndOfTimeStepUpdate** [void; [Method];  
' EndOfTimeStepUpdate(arg list) ' -- Calls EndOfTimeStepCleanup in SolutionAlgs

**DSSim\_Comms** [out, retval] Type: IDSSimComs\*\* Value; [Property (get)];  
'value = DSSim\_Comms ' -- (no Help string available)

**YNodeOrder** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = YNodeOrder ' -- Variant array of strings containing the names of the nodes in the same order as the Y matrix

**YCurrents** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = YCurrents ' -- Variant array of doubles containing complex injection currents for the present solution. Is is the  $i_0$

**PVSystems** [out, retval] Type: IPVSystems\*\* Value; [Property (get)];  
'value = PVSystems ' -- Interface to PVSystems collection

## Bus Interface

**Name** [out, retval] Type: BSTR\* Name; [Property (get)];  
'value = Name ' -- Name of Bus

**NumNodes** [out, retval] Type: long\* NumNodes; [Property (get)];  
'value = NumNodes ' -- Number of Nodes this bus.

**Voltages** [out, retval] Type: VARIANT\* Voltages; [Property (get)];  
'value = Voltages ' -- Complex array of voltages at this bus.

**SeqVoltages** [out, retval] Type: VARIANT\* SeqVoltages; [Property (get)];  
'value = SeqVoltages ' -- Double Array of sequence voltages at this bus.

**Nodes** [out, retval] Type: VARIANT\* Nodes; [Property (get)];  
'value = Nodes ' -- Integer Array of Node Numbers defined at the bus in same order as the voltages.

**Voc** [out, retval] Type: VARIANT\* Voc; [Property (get)];  
'value = Voc ' -- Open circuit voltage; Complex array.

**Isc** [out, retval] Type: VARIANT\* Isc; [Property (get)];  
'value = Isc ' -- Short circuit currents at bus; Complex Array.

**puVoltages** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = puVoltages ' -- Complex Array of pu voltages at the bus.

**kVBase** [out, retval] Type: double\* Value; [Property (get)];  
'value = kVBase ' -- Base voltage at bus in kV

**ZscMatrix** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = ZscMatrix ' -- Complex array of Zsc matrix at bus. Column by column.

**Zsc1** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Zsc1 ' -- *Complex Positive-Sequence short circuit impedance at bus..*

**Zsc0** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Zsc0 ' -- *Complex Zero-Sequence short circuit impedance at bus.*

**ZscRefresh** [out, retval] Type: VARIANT\_BOOL\* Value; [Method];  
'ZscRefresh(arg list) ' -- *Recomputes Zsc for active bus for present circuit configuration.*

**YscMatrix** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = YscMatrix ' -- *Complex array of Ysc matrix at bus. Column by column.*

**Coorddefined** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = Coorddefined ' -- *False=0 else True. Indicates whether a coordinate has been defined for this bus*

**x** [out, retval] Type: double\* Value; [Property (get)];  
'value = x ' -- *X Coordinate for bus (double*

**x** [in] Type: double Value; [Property (put)];  
'x = value' -- *X Coordinate for bus (double*

**y** [out, retval] Type: double\* Value; [Property (get)];  
'value = y ' -- *Y coordinate for bus(double*

**y** [in] Type: double Value; [Property (put)];  
'y = value' -- *Y coordinate for bus(double*

**Distance** [out, retval] Type: double\* Value; [Property (get)];  
'value = Distance ' -- *Distance from energymeter (if non-zero*

**GetUniqueNodeNumber** [in] Type: long StartNumber, [out, retval] Type: long\* Value; [Method];  
'GetUniqueNodeNumber(arg list) ' -- *Returns a unique node number at the active bus to avoid node collisions and adds it to the node list for the bus.*

**CplxSeqVoltages** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = CplxSeqVoltages ' -- *Complex Double array of Sequence Voltages (0, 1, 2*

**Lambda** [out, retval] Type: double\* Value; [Property (get)];  
'value = Lambda ' -- *Accumulated failure rate downstream from this bus; faults per year*

**N\_interruptions** [out, retval] Type: double\* Value; [Property (get)];  
'value = N\_interruptions ' -- *Number of interruptions this bus per year*

**Int\_Duration** [out, retval] Type: double\* Value; [Property (get)];  
'value = Int\_Duration ' -- *Average interruption duration, hr.*

**Cust\_Interruptions** [out, retval] Type: double\* Value; [Property (get)];  
'value = Cust\_Interruptions ' -- *Annual number of customer-interruptions from this bus*

**Cust\_Duration** [out, retval] Type: double\* Value; [Property (get)];  
'value = Cust\_Duration ' -- *Accumulated customer outage durations*

**N\_Customers** [out, retval] Type: long\* Value; [Property (get)];  
'value = N\_Customers ' -- *Total numbers of customers served downline from this bus*

**VLL** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = VLL ' -- For 2- and 3-phase buses, returns variant array of complex numbers representin L-L voltages in volts. Returns -1.0 for 1-phase bus. If more than 3 phases, returns only first 3.

**puVLL** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = puVLL ' -- Returns Complex array of pu L-L voltages for 2- and 3-phase buses. Returns -1.0 for 1-phase bus. If more than 3 phases, returns only 3 phases.

**VMagAngle** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = VMagAngle ' -- Variant Array of doubles containing voltages in Magnitude (VLN

**puVmagAngle** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = puVmagAngle ' -- Variant array of doubles containig voltage magnitude, angle pairs in per unit

**TotalMiles** [out, retval] Type: double\* Value; [Property (get)];  
'value = TotalMiles ' -- Total length of line downline from this bus, in miles. For recloser siting algorithm.

**SectionID** [out, retval] Type: long\* Value; [Property (get)];  
'value = SectionID ' -- Integer ID of the feeder section in which this bus is located.

## DSS Interface

**NumCircuits** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumCircuits ' -- Number of Circuits currently defined

**Circuits** [in] Type: VARIANT Idx, [out, retval] Type: ICircuit\*\* Value; [Property (get)];  
'value = Circuits ' -- Collection of Circuit objects

**ActiveCircuit** [out, retval] Type: ICircuit\*\* Value; [Property (get)];  
'value = ActiveCircuit ' -- Returns interface to the active circuit.

**Text** [out, retval] Type: IText\*\* Value; [Property (get)];  
'value = Text ' -- Returns the DSS Text (command-result

**Error** [out, retval] Type: IError\*\* Value; [Property (get)];  
'value = Error ' -- Returns Error interface.

**NewCircuit** [in] Type: BSTR Name, [out, retval] Type: ICircuit\*\* Value; [Method];  
' NewCircuit(arg list) ' -- Make a new circuit and return interface to active circuit.

**ClearAll** [void; [Method];  
' ClearAll(arg list) ' -- Clears all circuit definitions.

**ShowPanel** [void; [Method];  
' ShowPanel(arg list) ' -- Shows non-MDI child form of the Main DSS Edit Form

**Start** [in] Type: long code, [out, retval] Type: VARIANT\_BOOL\* Value; [Method];  
' Start(arg list) ' -- Validate the user and start the DSS. Returns TRUE if successful.

**Version** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Version ' -- Get version string for the DSS.

**DSSProgress** [out, retval] Type: IDSSProgress\*\* Value; [Property (get)];  
'value = DSSProgress ' -- Gets interface to the DSS Progress Meter

**Classes** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Classes ' -- List of DSS intrinsic classes (names of the classes

**UserClasses** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = UserClasses ' -- List of user-defined classes

**NumClasses** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumClasses ' -- Number of DSS intrinsic classes

**NumUserClasses** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumUserClasses ' -- Number of user-defined classes

**DataPath** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = DataPath ' -- DSS Data File Path. Default path for reports, etc. from DSS

**DataPath** [in] Type: BSTR Value; [Property (put)];  
' DataPath = value' -- DSS Data File Path. Default path for reports, etc. from DSS

**Reset** [void; [Method];  
' Reset(arg list) ' -- Resets DSS Initialization for restarts, etc from applets

**AllowForms** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = AllowForms ' -- Default is TRUE. Use this to set to FALSE; Cannot reset to TRUE;

**AllowForms** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' AllowForms = value' -- Default is TRUE. Use this to set to FALSE; Cannot reset to TRUE;

**DefaultEditor** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = DefaultEditor ' -- Returns the path name for the default text editor.

**ActiveClass** [out, retval] Type: IActiveClass\*\* Value; [Property (get)];  
'value = ActiveClass ' -- Returns interface to the active class.

**SetActiveClass** [in] Type: BSTR ClassName, [out, retval] Type: long\* Value; [Method];  
' SetActiveClass(arg list) ' -- Sets the Active DSS Class for use with ActiveClass interface. Same as SetActiveClass in Circuit interface.

**Executive** [out, retval] Type: IDSS\_Executive\*\* Value; [Property (get)];  
'value = Executive ' -- Interface to DSS Executive commands and options

**Events** [out, retval] Type: IDSSEvents\*\* Value; [Property (get)];  
'value = Events ' -- Interface to the DSS Events

**CmathLib** [out, retval] Type: ICMATHLib\*\* Value; [Property (get)];  
'value = CmathLib ' -- Returns an interface to the complex math library.

**Parser** [out, retval] Type: IParser\*\* Value; [Property (get)];  
'value = Parser ' -- Returns interface to the OpenDSS Parser library for use by user-written programs.

**DSSim\_Coms** [out, retval] Type: IDSSimComs\*\* Value; [Property (get)];  
'value = DSSim\_Coms ' -- (no Help string available)

## Solution Interface

**Solve** [void; [Method];

'Solve(arg list)' -- Execute solution for present solution mode.

**Mode** [out, retval] Type: long\* Mode; [Property (get)];

'value = Mode' -- Set present solution mode (by a text code - see DSS Help

**Mode** [in] Type: long Mode; [Property (put)];

'Mode = value' -- Set present solution mode (by a text code - see DSS Help

**Frequency** [out, retval] Type: double\* Frequency; [Property (get)];

'value = Frequency' -- Set the Frequency for next solution

**Frequency** [in] Type: double Frequency; [Property (put)];

'Frequency = value' -- Set the Frequency for next solution

**Hour** [out, retval] Type: long\* Hour; [Property (get)];

'value = Hour' -- Set Hour for time series solutions.

**Hour** [in] Type: long Hour; [Property (put)];

'Hour = value' -- Set Hour for time series solutions.

**Seconds** [out, retval] Type: double\* Seconds; [Property (get)];

'value = Seconds' -- Seconds from top of the hour.

**Seconds** [in] Type: double Seconds; [Property (put)];

'Seconds = value' -- Seconds from top of the hour.

**StepSize** [out, retval] Type: double\* StepSize; [Property (get)];

'value = StepSize' -- Time step size in sec

**StepSize** [in] Type: double StepSize; [Property (put)];

'StepSize = value' -- Time step size in sec

**Year** [out, retval] Type: long\* Year; [Property (get)];

'value = Year' -- Set year for planning studies

**Year** [in] Type: long Year; [Property (put)];

'Year = value' -- Set year for planning studies

**LoadMult** [out, retval] Type: double\* LoadMult; [Property (get)];

'value = LoadMult' -- Default load multiplier applied to all non-fixed loads

**LoadMult** [in] Type: double LoadMult; [Property (put)];

'LoadMult = value' -- Default load multiplier applied to all non-fixed loads

**Iterations** [out, retval] Type: long\* Iterations; [Property (get)];

'value = Iterations' -- Number of iterations taken for last solution. (Same as TotalIterations

**MaxIterations** [out, retval] Type: long\* MaxIterations; [Property (get)];

'value = MaxIterations' -- Max allowable iterations.

**MaxIterations** [in] Type: long MaxIterations; [Property (put)];

'MaxIterations = value' -- Max allowable iterations.

**Tolerance** [out, retval] Type: double\* Tolerance; [Property (get)];



'value = Tolerance ' -- *Solution convergence tolerance.*

**Tolerance** [in] Type: double Tolerance; [Property (put)];

' Tolerance = value' -- *Solution convergence tolerance.*

**Number** [out, retval] Type: long\* Number; [Property (get)];

'value = Number ' -- *Number of solutions to perform for Monte Carlo and time series simulations*

**Number** [in] Type: long Number; [Property (put)];

' Number = value' -- *Number of solutions to perform for Monte Carlo and time series simulations*

**Random** [out, retval] Type: long\* Random; [Property (get)];

'value = Random ' -- *Randomization mode for random variables \i0*

**Random** [in] Type: long Random; [Property (put)];

' Random = value' -- *Randomization mode for random variables \i0*

**ModelID** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = ModelID ' -- *ID (text*

**LoadModel** [out, retval] Type: long\* Value; [Property (get)];

'value = LoadModel ' -- *Load Model: dssPowerFlow (default*

**LoadModel** [in] Type: long Value; [Property (put)];

' LoadModel = value' -- *Load Model: dssPowerFlow (default*

**LDCurve** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = LDCurve ' -- *Load-Duration Curve name for LD modes*

**LDCurve** [in] Type: BSTR Value; [Property (put)];

' LDCurve = value' -- *Load-Duration Curve name for LD modes*

**pctGrowth** [out, retval] Type: double\* Value; [Property (get)];

'value = pctGrowth ' -- *Percent default annual load growth rate*

**pctGrowth** [in] Type: double Value; [Property (put)];

' pctGrowth = value' -- *Percent default annual load growth rate*

**AddType** [out, retval] Type: long\* Value; [Property (get)];

'value = AddType ' -- *Type of device to add in AutoAdd Mode: dssGen (Default*

**AddType** [in] Type: long Value; [Property (put)];

' AddType = value' -- *Type of device to add in AutoAdd Mode: dssGen (Default*

**GenkW** [out, retval] Type: double\* Value; [Property (get)];

'value = GenkW ' -- *Generator kW for AutoAdd mode*

**GenkW** [in] Type: double Value; [Property (put)];

' GenkW = value' -- *Generator kW for AutoAdd mode*

**GenPF** [out, retval] Type: double\* Value; [Property (get)];

'value = GenPF ' -- *PF for generators in AutoAdd mode*

**GenPF** [in] Type: double Value; [Property (put)];

' GenPF = value' -- *PF for generators in AutoAdd mode*

**Capkvar** [out, retval] Type: double\* Value; [Property (get)];

'value = Capkvar ' -- *Capacitor kvar for adding capacitors in AutoAdd mode*

**Capkvar** [in] Type: double Value; [Property (put)];  
'Capkvar = value' -- Capacitor kvar for adding capacitors in AutoAdd mode

**Algorithm** [out, retval] Type: long\* Value; [Property (get)];  
'value = Algorithm ' -- Base Solution algorithm: dssNormalSolve | dssNewtonSolve

**Algorithm** [in] Type: long Value; [Property (put)];  
'Algorithm = value' -- Base Solution algorithm: dssNormalSolve | dssNewtonSolve

**ControlMode** [out, retval] Type: long\* Value; [Property (get)];  
'value = ControlMode ' -- dssStatic\* | dssEvent | dssTime Modes for control devices

**ControlMode** [in] Type: long Value; [Property (put)];  
'ControlMode = value' -- dssStatic\* | dssEvent | dssTime Modes for control devices

**GenMult** [out, retval] Type: double\* Value; [Property (get)];  
'value = GenMult ' -- Default Multiplier applied to generators (like LoadMult

**GenMult** [in] Type: double Value; [Property (put)];  
'GenMult = value' -- Default Multiplier applied to generators (like LoadMult

**DefaultDaily** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = DefaultDaily ' -- Default daily load shape (defaults to \i0

**DefaultDaily** [in] Type: BSTR Value; [Property (put)];  
'DefaultDaily = value' -- Default daily load shape (defaults to \i0

**DefaultYearly** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = DefaultYearly ' -- Default Yearly load shape (defaults to \i0

**DefaultYearly** [in] Type: BSTR Value; [Property (put)];  
'DefaultYearly = value' -- Default Yearly load shape (defaults to \i0

**EventLog** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = EventLog ' -- Array of strings containing the Event Log

**dblHour** [out, retval] Type: double\* Value; [Property (get)];  
'value = dblHour ' -- Hour as a double, including fractional part

**dblHour** [in] Type: double Value; [Property (put)];  
'dblHour = value' -- Hour as a double, including fractional part

**StepsizeMin** [in] Type: double Param1; [Property (put)];  
'StepsizeMin = value' -- Set Stepsize in minutes

**StepsizeHr** [in] Type: double Param1; [Property (put)];  
'StepsizeHr = value' -- Set Stepsize in Hr

**ControllIterations** [out, retval] Type: long\* Value; [Property (get)];  
'value = ControllIterations ' -- Value of the control iteration counter

**ControllIterations** [in] Type: long Value; [Property (put)];  
'ControllIterations = value' -- Value of the control iteration counter

**MaxControllIterations** [out, retval] Type: long\* Value; [Property (get)];  
'value = MaxControllIterations ' -- Maximum allowable control iterations

**MaxControllIterations** [in] Type: long Value; [Property (put)];

' MaxControlIterations = value' -- *Maximum allowable control iterations*

**Sample\_DoControlActions** [void; [Method];

' Sample\_DoControlActions(arg list)' -- *Sample controls and then process the control queue for present control mode and dispatch control actions*

**CheckFaultStatus** [void; [Method];

' CheckFaultStatus(arg list)' -- *Executes status check on all fault objects defined in the circuit.*

**SolveSnap** [void; [Method];

' SolveSnap(arg list)' -- *Execute the snapshot power flow routine in the DSS that solves at the present state with control actions*

**SolveDirect** [void; [Method];

' SolveDirect(arg list)' -- *Executes a direct solution from the system Y matrix, ignoring compensation currents of loads, generators (includes Yprim only*

**SolvePflow** [void; [Method];

' SolvePflow(arg list)' -- *Solves using present power flow method. Iterative solution rather than direct solution.*

**SolveNoControl** [void; [Method];

' SolveNoControl(arg list)' -- *Similar to SolveSnap except no control actions are checked or executed*

**SolvePlusControl** [void; [Method];

' SolvePlusControl(arg list)' -- *Executes a power flow solution (SolveNoControl*

**InitSnap** [void; [Method];

' InitSnap(arg list)' -- *Initializes some variables for snap shot power flow. SolveSnap does this automatically.*

**CheckControls** [void; [Method];

' CheckControls(arg list)' -- *The normal process for sampling and executing Control Actions and Fault Status and rebuilds Y if necessary.*

**SampleControlDevices** [void; [Method];

' SampleControlDevices(arg list)' -- *Executes a sampling of all intrinsic control devices, which push control actions onto the control queue.*

**DoControlActions** [void; [Method];

' DoControlActions(arg list)' -- *Pops control actions off the control queue and dispatches to the proper control element*

**BuildYMatrix** [in] Type: long BuildOption, [in] Type: long AllocateVI; [Method];

' BuildYMatrix(arg list)' -- *Force building of the System Y matrix*

**SystemYChanged** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = SystemYChanged' -- *Flag that indicates if elements of the System Y have been changed by recent activity.*

**Converged** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = Converged' -- *Flag to indicate whether the circuit solution converged*

**Converged** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
 'Converged = value' -- Flag to indicate whether the circuit solution converged

**TotalIterations** [out, retval] Type: long\* Value; [Property (get)];  
 'value = TotalIterations' -- Total iterations including control iterations for most recent solution.

**MostIterationsDone** [out, retval] Type: long\* Value; [Property (get)];  
 'value = MostIterationsDone' -- Max number of iterations required to converge at any control iteration of the most recent solution.

**ControlActionsDone** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
 'value = ControlActionsDone' -- Flag indicating the control actions are done.

**ControlActionsDone** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
 'ControlActionsDone = value' -- (no Help string available)

**FinishTimeStep** [void; [Method];  
 'FinishTimeStep(arg list)' -- Call Cleanup, sample monitors, and increment time at end of time step.

**Cleanup** [void; [Method];  
 'Cleanup(arg list)' -- Update storage, invcontrol, etc at end of time step

## Monitors Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = AllNames' -- Array of all Monitor Names

**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First' -- Sets the first Monitor active. Returns 0 if no monitors.

**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next' -- Sets next monitor active. Returns 0 if no more.

**Reset** [void; [Method];  
 'Reset(arg list)' -- Resets active Monitor object.

**ResetAll** [void; [Method];  
 'ResetAll(arg list)' -- Resets all Monitor Objects

**Sample** [void; [Method];  
 'Sample(arg list)' -- Causes active Monitor to take a sample.

**Save** [void; [Method];  
 'Save(arg list)' -- Causes active monitor to save its current sample buffer to its monitor stream. Then you can access the Bytestream or channel data. Most standard solution modes do this automatically.

**Show** [void; [Method];  
 'Show(arg list)' -- Converts monitor file to text and displays with text editor

**FileName** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = FileName' -- Name of CSV file associated with active Monitor.

**Mode** [out, retval] Type: long\* Value; [Property (get)];  
'value = Mode ' -- Set Monitor mode (bitmask integer - see DSS Help

**Mode** [in] Type: long Value; [Property (put)];  
' Mode = value' -- Set Monitor mode (bitmask integer - see DSS Help

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- Sets the active Monitor object by name

**Name** [in] Type: BSTR Value; [Property (put)];  
' Name = value' -- Sets the active Monitor object by name

**ByteStream** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = ByteStream ' -- Byte Array containing monitor stream values. Make sure a \i0

**SampleCount** [out, retval] Type: long\* Value; [Property (get)];  
'value = SampleCount ' -- Number of Samples in Monitor at Present

**SampleAll** [void; [Method];  
' SampleAll(arg list) ' -- Causes all Monitors to take a sample of the present state

**SaveAll** [void; [Method];  
' SaveAll(arg list) ' -- Save all Monitor buffers to their respective file streams.

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- Number of Monitors

**Process** [void; [Method];  
' Process(arg list) ' -- Post-process monitor samples taken so far, e.g., Pst for mode=4

**ProcessAll** [void; [Method];  
' ProcessAll(arg list) ' -- All monitors post-process the data taken so far.

**FileVersion** [out, retval] Type: long\* Value; [Property (get)];  
'value = FileVersion ' -- Monitor File Version (integer

**RecordSize** [out, retval] Type: long\* Value; [Property (get)];  
'value = RecordSize ' -- Size of each record in ByteStream (Integer

**Header** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Header ' -- Header string; Variant array of strings containing Channel names

**dblHour** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = dblHour ' -- Variant array of doubles containing in time value in hours for time-sampled monitor values; Empty if frequency-sampled values for harmonics solution (see dblFreq

**dblFreq** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = dblFreq ' -- Variant array of doubles containing frequency values for harmonics mode solutions; Empty for time mode solutions (use dblHour

**Channel** [in] Type: long Index, [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Channel ' -- Variant array of doubles for the specified channel (usage: MyArray = DSSMonitor.Channel(i

**NumChannels** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumChannels ' -- Number of Channels in the active Monitor

**Element** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Element ' -- Full object name of element being monitored.  
**Element** [in] Type: BSTR Value; [Property (put)];  
 'Element = value' -- Full object name of element being monitored.  
**Terminal** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Terminal ' -- Terminal number of element being monitored  
**Terminal** [in] Type: long Value; [Property (put)];  
 'Terminal = value' -- Terminal number of element being monitored.

## Meters Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = AllNames ' -- Array of all energy Meter names  
**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First ' -- Set the first energy Meter active. Returns 0 if none.  
**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next ' -- Sets the next energy Meter active. Returns 0 if no more.  
**RegisterNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = RegisterNames ' -- Array of strings containing the names of the registers.  
**RegisterValues** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = RegisterValues ' -- Array of all the values contained in the Meter registers for the active Meter.  
**Reset** [void; [Method];  
 'Reset(arg list) ' -- Resets registers of active Meter.  
**ResetAll** [void; [Method];  
 'ResetAll(arg list) ' -- Resets registers of all Meter objects.  
**Sample** [void; [Method];  
 'Sample(arg list) ' -- Forces active Meter to take a sample.  
**Save** [void; [Method];  
 'Save(arg list) ' -- Saves meter register values.  
**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Name ' -- Get/Set the active meter name.  
**Name** [in] Type: BSTR Value; [Property (put)];  
 'Name = value' -- Set a meter to be active by name.  
**Totals** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = Totals ' -- Totals of all registers of all meters  
**Peakcurrent** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = Peakcurrent ' -- Array of doubles to set values of Peak Current property  
**Peakcurrent** [in] Type: VARIANT Value; [Property (put)];

' Peakcurrent = value' -- Array of doubles to set values of Peak Current property

**CalcCurrent** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = CalcCurrent ' -- Set the magnitude of the real part of the Calculated Current (normally determined by solution

**CalcCurrent** [in] Type: VARIANT Value; [Property (put)];

' CalcCurrent = value' -- Set the magnitude of the real part of the Calculated Current (normally determined by solution

**AllocFactors** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllocFactors ' -- Array of doubles: set the phase allocation factors for the active meter.

**AllocFactors** [in] Type: VARIANT Value; [Property (put)];

' AllocFactors = value' -- Array of doubles: set the phase allocation factors for the active meter.

**MeteredElement** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = MeteredElement ' -- Set Name of metered element

**MeteredElement** [in] Type: BSTR Value; [Property (put)];

' MeteredElement = value' -- Set Name of metered element

**MeteredTerminal** [out, retval] Type: long\* Value; [Property (get)];

'value = MeteredTerminal ' -- set Number of Metered Terminal

**MeteredTerminal** [in] Type: long Value; [Property (put)];

' MeteredTerminal = value' -- set Number of Metered Terminal

**DIFilesAreOpen** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = DIFilesAreOpen ' -- Global Flag in the DSS to indicate if Demand Interval (DI

**SampleAll** [void; [Method];

' SampleAll(arg list) ' -- Causes all EnergyMeter objects to take a sample at the present time

**SaveAll** [void; [Method];

' SaveAll(arg list) ' -- Save All EnergyMeter objects

**OpenAllDIFiles** [void; [Method];

' OpenAllDIFiles(arg list) ' -- Open Demand Interval (DI

**CloseAllDIFiles** [void; [Method];

' CloseAllDIFiles(arg list) ' -- Close All Demand Interval Files ( Necessary at the end of a run

**CountEndElements** [out, retval] Type: long\* Value; [Property (get)];

'value = CountEndElements ' -- Number of zone end elements in the active meter zone.

**AllEndElements** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllEndElements ' -- Variant array of names of all zone end elements.

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- Number of Energy Meters in the Active Circuit

**AllBranchesInZone** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllBranchesInZone ' -- Wide string list of all branches in zone of the active energymeter object.

**CountBranches** [out, retval] Type: long\* Value; [Property (get)];

'value = CountBranches ' -- Number of branches in Active energymeter zone. (Same as sequencelist size

**SAIFI** [out, retval] Type: double\* Value; [Property (get)];

'value = SAIFI ' -- Returns SAIFI for this meter's Zone. Execute Reliability Calc method first.

**SequenceIndex** [out, retval] Type: long\* Value; [Property (get)];

'value = SequenceIndex ' -- Get/set Index into Meter's SequenceList that contains branch pointers in lexical order. Earlier index guaranteed to be upline from later index. Sets PDelement active.

**SequenceIndex** [in] Type: long Value; [Property (put)];

'SequenceIndex = value' -- Get/set Index into Meter's SequenceList that contains branch pointers in lexical order. Earlier index guaranteed to be upline from later index. Sets PDelement active.

**SAFIKW** [out, retval] Type: double\* Value; [Property (get)];

'value = SAFIKW ' -- SAIFI based on kW rather than number of customers. Get after reliability calcs.

**DoReliabilityCalc** [in] Type: VARIANT\_BOOL AssumeRestoration; [Method];

'DoReliabilityCalc(arg list) ' -- Calculate SAIFI, etc.

**SeqListSize** [out, retval] Type: long\* Value; [Property (get)];

'value = SeqListSize ' -- Size of Sequence List

**TotalCustomers** [out, retval] Type: long\* Value; [Property (get)];

'value = TotalCustomers ' -- Total Number of customers in this zone (downline from the EnergyMeter

**SAIDI** [out, retval] Type: double\* Value; [Property (get)];

'value = SAIDI ' -- SAIDI for this meter's zone. Execute DoReliabilityCalc first.

**CustInterrupts** [out, retval] Type: double\* Value; [Property (get)];

'value = CustInterrupts ' -- Total customer interruptions for this Meter zone based on reliability calcs.

## Generators Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames ' -- Array of names of all Generator objects.

**RegisterNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = RegisterNames ' -- Array of Names of all generator energy meter registers

**RegisterValues** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = RegisterValues ' -- Array of values in generator energy meter registers.

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First ' -- Sets first Generator to be active. Returns 0 if none.

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next ' -- Sets next Generator to be active. Returns 0 if no more.

**ForcedON** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];



'value = ForcedON ' -- Indicates whether the generator is forced ON regardless of other dispatch criteria.

**ForcedON** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' ForcedON = value' -- Indicates whether the generator is forced ON regardless of other dispatch criteria.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name ' -- Sets a generator active by name.

**Name** [in] Type: BSTR Value; [Property (put)];

' Name = value' -- Sets a generator active by name.

**kV** [out, retval] Type: double\* Value; [Property (get)];

'value = kV ' -- Voltage base for the active generator, kV

**kV** [in] Type: double Value; [Property (put)];

' kV = value' -- Voltage base for the active generator, kV

**kW** [out, retval] Type: double\* Value; [Property (get)];

'value = kW ' -- kW output for the active generator. kvar is updated for current power factor.

**kW** [in] Type: double Value; [Property (put)];

' kW = value' -- kW output for the active generator. kvar is updated for current power factor

**kvar** [out, retval] Type: double\* Value; [Property (get)];

'value = kvar ' -- kvar output for the active generator. Updates power factor based on present kW value.

**kvar** [in] Type: double Value; [Property (put)];

' kvar = value' -- kvar output for the active generator. Updates power factor based on present kW.

**PF** [out, retval] Type: double\* Value; [Property (get)];

'value = PF ' -- Power factor (pos. = producing vars

**PF** [in] Type: double Value; [Property (put)];

' PF = value' -- Power factor (pos. = producing vars

**Phases** [out, retval] Type: long\* Value; [Property (get)];

'value = Phases ' -- Number of phases

**Phases** [in] Type: long Value; [Property (put)];

' Phases = value' -- Number of phases

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- Number of Generator Objects in Active Circuit

**idx** [out, retval] Type: long\* Value; [Property (get)];

'value = idx ' -- Get/Set active Generator by index into generators list. 1..Count

**idx** [in] Type: long Value; [Property (put)];

' idx = value' -- Get/Set active Generator by index into generators list. 1..Count

## DSSProgress Interface

**PctProgress** [in] Type: long Param1; [Property (put)];  
' PctProgress = value' -- *Percent progress to indicate [0..100]*

**Caption** [in] Type: BSTR Param1; [Property (put)];  
' Caption = value' -- *Caption to appear on the bottom of the DSS Progress form.*

**Show** [void; [Method];  
' Show(arg list)' -- *Shows progress form with null caption and progress set to zero.*

**Close** [void; [Method];  
' Close(arg list)' -- *Closes (hides*

## Settings Interface

**AllowDuplicates** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = AllowDuplicates' -- *True / False\* Designates whether to allow duplicate names of objects*

**AllowDuplicates** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' AllowDuplicates = value' -- *True / False\* Designates whether to allow duplicate names of objects*

**ZoneLock** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = ZoneLock' -- *True / False\* Locks Zones on energy meters to prevent rebuilding if a circuit change occurs.*

**ZoneLock** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' ZoneLock = value' -- *True / False\* Locks Zones on energy meters to prevent rebuilding if a circuit change occurs.*

**AllocationFactors** [in] Type: double Param1; [Property (put)];  
' AllocationFactors = value' -- *Sets all load allocation factors for all loads defined by XFKVA property to this value.*

**AutoBusList** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = AutoBusList' -- *List of Buses or (File=xxxx*

**AutoBusList** [in] Type: BSTR Value; [Property (put)];  
' AutoBusList = value' -- *List of Buses or (File=xxxx*

**CktModel** [out, retval] Type: long\* Value; [Property (get)];  
'value = CktModel' -- *dssMultiphase \* / dssPositiveSeq lIndicate if the circuit model is positive sequence.*

**CktModel** [in] Type: long Value; [Property (put)];  
' CktModel = value' -- *dssMultiphase \* / dssPositiveSeq lIndicate if the circuit model is positive sequence.*

**NormVminpu** [out, retval] Type: double\* Value; [Property (get)];

'value = NormVminpu ' -- *Per Unit minimum voltage for Normal conditions.*

**NormVminpu** [in] Type: double Value; [Property (put)];

' NormVminpu = value' -- *Per Unit minimum voltage for Normal conditions.*

**NormVmaxpu** [out, retval] Type: double\* Value; [Property (get)];

'value = NormVmaxpu ' -- *Per Unit maximum voltage for Normal conditions.*

**NormVmaxpu** [in] Type: double Value; [Property (put)];

' NormVmaxpu = value' -- *Per Unit maximum voltage for Normal conditions.*

**EmergVminpu** [out, retval] Type: double\* Value; [Property (get)];

'value = EmergVminpu ' -- *Per Unit minimum voltage for Emergency conditions.*

**EmergVminpu** [in] Type: double Value; [Property (put)];

' EmergVminpu = value' -- *Per Unit minimum voltage for Emergency conditions.*

**EmergVmaxpu** [out, retval] Type: double\* Value; [Property (get)];

'value = EmergVmaxpu ' -- *Per Unit maximum voltage for Emergency conditions.*

**EmergVmaxpu** [in] Type: double Value; [Property (put)];

' EmergVmaxpu = value' -- *Per Unit maximum voltage for Emergency conditions.*

**UWeight** [out, retval] Type: double\* Value; [Property (get)];

'value = UWeight ' -- *Weighting factor applied to UE register values.*

**UWeight** [in] Type: double Value; [Property (put)];

' UWeight = value' -- *Weighting factor applied to UE register values.*

**LossWeight** [out, retval] Type: double\* Value; [Property (get)];

'value = LossWeight ' -- *Weighting factor applied to Loss register values.*

**LossWeight** [in] Type: double Value; [Property (put)];

' LossWeight = value' -- *Weighting factor applied to Loss register values.*

**UEregs** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = UEregs ' -- *Array of Integers defining energy meter registers to use for computing UE*

**UEregs** [in] Type: VARIANT Value; [Property (put)];

' UEregs = value' -- *Array of Integers defining energy meter registers to use for computing UE*

**LossRegs** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = LossRegs ' -- *Integer array defining which energy meter registers to use for computing losses*

**LossRegs** [in] Type: VARIANT Value; [Property (put)];

' LossRegs = value' -- *Integer array defining which energy meter registers to use for computing losses*

**Trapezoidal** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = Trapezoidal ' -- *True / False \* Gets value of trapezoidal integration flag in energy meters.*

**Trapezoidal** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' Trapezoidal = value' -- *True / False \* Gets value of trapezoidal integration flag in energy meters.*

**VoltageBases** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = VoltageBases ' -- *Array of doubles defining the legal voltage bases in kV L-L*

**VoltageBases** [in] Type: VARIANT Value; [Property (put)];  
 'VoltageBases = value' -- Array of doubles defining the legal voltage bases in kV L-L

**ControlTrace** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
 'value = ControlTrace' -- True / False\* Denotes whether to trace the control actions to a file.

**ControlTrace** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
 'ControlTrace = value' -- True / False\* Denotes whether to trace the control actions to a file.

**PriceSignal** [out, retval] Type: double\* Value; [Property (get)];  
 'value = PriceSignal' -- Price Signal for the Circuit

**PriceSignal** [in] Type: double Value; [Property (put)];  
 'PriceSignal = value' -- Price Signal for the Circuit

**PriceCurve** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = PriceCurve' -- Name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.

**PriceCurve** [in] Type: BSTR Value; [Property (put)];  
 'PriceCurve = value' -- Name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.

## Lines Interface

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Name' -- Specify the name of the Line element to set it active.

**Name** [in] Type: BSTR Value; [Property (put)];  
 'Name = value' -- Specify the name of the Line element to set it active.

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = AllNames' -- Names of all Line Objects

**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First' -- Invoking this property sets the first element active. Returns 0 if no lines. Otherwise, index of the line element.

**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next' -- Invoking this property advances to the next Line element active. Returns 0 if no more lines. Otherwise, index of the line element.

**New** [in] Type: BSTR Name, [out, retval] Type: long\* Value; [Method];  
 'New(arg list)' -- Creates a new Line and makes it the Active Circuit Element.

**Bus1** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Bus1' -- Name of bus for terminal 1.

**Bus1** [in] Type: BSTR Value; [Property (put)];  
 'Bus1 = value' -- Name of bus for terminal 1.

**Bus2** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Bus2' -- Name of bus for terminal 2.

**Bus2** [in] Type: BSTR Value; [Property (put)];  
'Bus2 = value' -- Name of bus for terminal 2.

**LineCode** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = LineCode' -- Name of LineCode object that defines the impedances.

**LineCode** [in] Type: BSTR Value; [Property (put)];  
'LineCode = value' -- Name of LineCode object that defines the impedances.

**Length** [out, retval] Type: double\* Value; [Property (get)];  
'value = Length' -- Length of line section in units compatible with the LineCode definition.

**Length** [in] Type: double Value; [Property (put)];  
'Length = value' -- Length of line section in units compatible with the LineCode definition.

**Phases** [out, retval] Type: long\* Value; [Property (get)];  
'value = Phases' -- Number of Phases, this Line element.

**Phases** [in] Type: long Value; [Property (put)];  
'Phases = value' -- Number of Phases, this Line element.

**R1** [out, retval] Type: double\* Value; [Property (get)];  
'value = R1' -- Positive Sequence resistance, ohms per unit length.

**R1** [in] Type: double Value; [Property (put)];  
'R1 = value' -- Positive Sequence resistance, ohms per unit length.

**X1** [out, retval] Type: double\* Value; [Property (get)];  
'value = X1' -- Positive Sequence reactance, ohms per unit length.

**X1** [in] Type: double Value; [Property (put)];  
'X1 = value' -- Positive Sequence reactance, ohms per unit length.

**R0** [out, retval] Type: double\* Value; [Property (get)];  
'value = R0' -- Zero Sequence resistance, ohms per unit length.

**R0** [in] Type: double Value; [Property (put)];  
'R0 = value' -- Zero Sequence resistance, ohms per unit length.

**X0** [out, retval] Type: double\* Value; [Property (get)];  
'value = X0' -- Zero Sequence reactance ohms per unit length.

**X0** [in] Type: double Value; [Property (put)];  
'X0 = value' -- Zero Sequence reactance ohms per unit length.

**C1** [out, retval] Type: double\* Value; [Property (get)];  
'value = C1' -- Positive Sequence capacitance, nanofarads per unit length.

**C1** [in] Type: double Value; [Property (put)];  
'C1 = value' -- Positive Sequence capacitance, nanofarads per unit length.

**C0** [out, retval] Type: double\* Value; [Property (get)];  
'value = C0' -- Zero Sequence capacitance, nanofarads per unit length.

**C0** [in] Type: double Value; [Property (put)];  
'C0 = value' -- Zero Sequence capacitance, nanofarads per unit length.

**Rmatrix** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Rmatrix ' -- *Resistance matrix (full*  
**Rmatrix** [in] Type: VARIANT Value; [Property (put)];  
' Rmatrix = value' -- *Resistance matrix (full*  
**Xmatrix** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Xmatrix ' -- *(no Help string available)*  
**Xmatrix** [in] Type: VARIANT Value; [Property (put)];  
' Xmatrix = value' -- *(no Help string available)*  
**Cmatrix** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Cmatrix ' -- *(no Help string available)*  
**Cmatrix** [in] Type: VARIANT Value; [Property (put)];  
' Cmatrix = value' -- *(no Help string available)*  
**NormAmps** [out, retval] Type: double\* Value; [Property (get)];  
'value = NormAmps ' -- *Normal ampere rating of Line.*  
**NormAmps** [in] Type: double Value; [Property (put)];  
' NormAmps = value' -- *Normal ampere rating of Line.*  
**EmergAmps** [out, retval] Type: double\* Value; [Property (get)];  
'value = EmergAmps ' -- *Emergency (maximum*  
**EmergAmps** [in] Type: double Value; [Property (put)];  
' EmergAmps = value' -- *Emergency (maximum*  
**Geometry** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Geometry ' -- *Line geometry code*  
**Geometry** [in] Type: BSTR Value; [Property (put)];  
' Geometry = value' -- *Line geometry code*  
**Rg** [out, retval] Type: double\* Value; [Property (get)];  
'value = Rg ' -- *Earth return resistance value used to compute line impedances at power frequency*  
**Rg** [in] Type: double Value; [Property (put)];  
' Rg = value' -- *Earth return resistance value used to compute line impedances at power frequency*  
**Xg** [out, retval] Type: double\* Value; [Property (get)];  
'value = Xg ' -- *Earth return reactance value used to compute line impedances at power frequency*  
**Xg** [in] Type: double Value; [Property (put)];  
' Xg = value' -- *Earth return reactance value used to compute line impedances at power frequency*  
**Rho** [out, retval] Type: double\* Value; [Property (get)];  
'value = Rho ' -- *Earth Resistivity, m-ohms*  
**Rho** [in] Type: double Value; [Property (put)];  
' Rho = value' -- *Earth Resistivity, m-ohms*  
**Yprim** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Yprim ' -- *Yprimitive: Does Nothing at present on Put; Dangerous*  
**Yprim** [in] Type: VARIANT Value; [Property (put)];  
' Yprim = value' -- *Yprimitive: Does Nothing at present on Put; Dangerous*

**NumCust** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumCust ' -- Number of customers on this line section.

**TotalCust** [out, retval] Type: long\* Value; [Property (get)];  
'value = TotalCust ' -- Total Number of customers served from this line section.

**Parent** [out, retval] Type: long\* Value; [Property (get)];  
'value = Parent ' -- Sets Parent of the active Line to be the active line. Returns 0 if no parent or action fails.

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- Number of Line objects in Active Circuit.

**Spacing** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Spacing ' -- Line spacing code

**Spacing** [in] Type: BSTR Value; [Property (put)];  
' Spacing = value' -- Line spacing code

**Units** [out, retval] Type: long\* Value; [Property (get)];  
'value = Units ' -- (no Help string available)

**Units** [in] Type: long Value; [Property (put)];  
' Units = value' -- (no Help string available)

## CtrlQueue Interface

**ClearQueue** [void; [Method];  
' ClearQueue(arg list) ' -- Clear control queue

**Delete** [in] Type: long ActionHandle; [Method];  
' Delete(arg list) ' -- Delete a control action from the DSS control queue by referencing the handle of the action

**NumActions** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumActions ' -- Number of Actions on the current actionlist (that have been popped off the control queue by CheckControlActions

**Action** [in] Type: long Param1; [Property (put)];  
' Action = value' -- Set the active action by index

**ActionCode** [out, retval] Type: long\* Value; [Property (get)];  
'value = ActionCode ' -- Code for the active action. Long integer code to tell the control device what to do

**DeviceHandle** [out, retval] Type: long\* Value; [Property (get)];  
'value = DeviceHandle ' -- Handle (User defined

**Push** [in] Type: long Hour, [in] Type: double Seconds, [in] Type: long ActionCode, [in] Type: long DeviceHandle, [out, retval] Type: long\* Value; [Method];  
' Push(arg list) ' -- Push a control action onto the DSS control queue by time, action code, and device handle (user defined

**Show** [void; [Method];

'Show(arg list)' -- Show entire control queue in CSV format

**ClearActions** [void; [Method];

'ClearActions(arg list)' -- Clear the Action list.

**PopAction** [out, retval] Type: long\* Value; [Property (get)];

'value = PopAction' -- Pops next action off the action list and makes it the active action. Returns zero if none.

## Loads Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames' -- Variant array of strings containing all Load names

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First' -- Set first Load element to be active; returns 0 if none.

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next' -- Sets next Load element to be active; returns 0 if none else index of active load.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name' -- Set active load by name.

**Name** [in] Type: BSTR Value; [Property (put)];

'Name = value' -- Set active load by name.

**Idx** [out, retval] Type: long\* Value; [Property (get)];

'value = Idx' -- Sets active load by index into load list. 1..Count

**Idx** [in] Type: long Value; [Property (put)];

'Idx = value' -- Sets active load by index into load list. 1..Count

**kW** [out, retval] Type: double\* Value; [Property (get)];

'value = kW' -- Set kW for active Load. Updates kvar based on present PF.

**kW** [in] Type: double Value; [Property (put)];

'kW = value' -- Set kW for active Load. Updates kvar based on present PF.

**kV** [out, retval] Type: double\* Value; [Property (get)];

'value = kV' -- Set kV rating for active Load. For 2 or more phases set Line-Line kV. Else actual kV across terminals.

**kV** [in] Type: double Value; [Property (put)];

'kV = value' -- Set kV rating for active Load. For 2 or more phases set Line-Line kV. Else actual kV across terminals.

**kvar** [out, retval] Type: double\* Value; [Property (get)];

'value = kvar' -- Set kvar for active Load. Updates PF based in present kW.

**kvar** [in] Type: double Value; [Property (put)];

'kvar = value' -- Set kvar for active Load. Updates PF based on present kW.

**PF** [out, retval] Type: double\* Value; [Property (get)];



*'value = PF ' -- Set Power Factor for Active Load. Specify leading PF as negative. Updates kvar based on kW value*

**PF** [in] Type: double Value; [Property (put)];

*' PF = value' -- Set Power Factor for Active Load. Specify leading PF as negative. Updates kvar based on present value of kW.*

**Count** [out, retval] Type: long\* Value; [Property (get)];

*'value = Count ' -- Number of Load objects in active circuit.*

**PctMean** [out, retval] Type: double\* Value; [Property (get)];

*'value = PctMean ' -- Average percent of nominal load in Monte Carlo studies; only if no loadshape defined for this load.*

**PctMean** [in] Type: double Value; [Property (put)];

*' PctMean = value' -- (no Help string available)*

**PctStdDev** [out, retval] Type: double\* Value; [Property (get)];

*'value = PctStdDev ' -- Percent standard deviation for Monte Carlo load studies; if there is no loadshape assigned to this load.*

**PctStdDev** [in] Type: double Value; [Property (put)];

*' PctStdDev = value' -- (no Help string available)*

**AllocationFactor** [out, retval] Type: double\* Value; [Property (get)];

*'value = AllocationFactor ' -- Factor for allocating loads by connected xfkva*

**AllocationFactor** [in] Type: double Value; [Property (put)];

*' AllocationFactor = value' -- (no Help string available)*

**Cfactor** [out, retval] Type: double\* Value; [Property (get)];

*'value = Cfactor ' -- Factor relates average to peak kw. Used for allocation with kwh and kwhdays/*

**Cfactor** [in] Type: double Value; [Property (put)];

*' Cfactor = value' -- (no Help string available)*

**Class** [out, retval] Type: long\* Value; [Property (get)];

*'value = Class ' -- A code number used to separate loads by class or group. No effect on the solution.*

**Class** [in] Type: long Value; [Property (put)];

*' Class = value' -- (no Help string available)*

**IsDelta** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

*'value = IsDelta ' -- Delta loads are connected line-to-line.*

**IsDelta** [in] Type: VARIANT\_BOOL Value; [Property (put)];

*' IsDelta = value' -- (no Help string available)*

**CVRcurve** [out, retval] Type: BSTR\* Value; [Property (get)];

*'value = CVRcurve ' -- Name of a loadshape with both Mult and Qmult, for CVR factors as a function of time.*

**CVRcurve** [in] Type: BSTR Value; [Property (put)];

'CVRcurve = value' -- *(no Help string available)*

**CVRwatts** [out, retval] Type: double\* Value; [Property (get)];

'value = CVRwatts' -- *Percent reduction in P for percent reduction in V. Must be used with dssLoadModelCVR.*

**CVRwatts** [in] Type: double Value; [Property (put)];

'CVRwatts = value' -- *(no Help string available)*

**CVRvars** [out, retval] Type: double\* Value; [Property (get)];

'value = CVRvars' -- *Percent reduction in Q for percent reduction in V. Must be used with dssLoadModelCVR.*

**CVRvars** [in] Type: double Value; [Property (put)];

'CVRvars = value' -- *(no Help string available)*

**daily** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = daily' -- *Name of the loadshape for a daily load profile.*

**daily** [in] Type: BSTR Value; [Property (put)];

'daily = value' -- *(no Help string available)*

**duty** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = duty' -- *Name of the loadshape for a duty cycle simulation.*

**duty** [in] Type: BSTR Value; [Property (put)];

'duty = value' -- *(no Help string available)*

**kva** [out, retval] Type: double\* Value; [Property (get)];

'value = kva' -- *Base load kva. Also defined kw and kvar or pf input, or load allocation by kwh or xfkva.*

**kva** [in] Type: double Value; [Property (put)];

'kva = value' -- *(no Help string available)*

**kwh** [out, retval] Type: double\* Value; [Property (get)];

'value = kwh' -- *kwh billed for this period. Can be used with Cfactor for load allocation.*

**kwh** [in] Type: double Value; [Property (put)];

'kwh = value' -- *(no Help string available)*

**kwhdays** [out, retval] Type: double\* Value; [Property (get)];

'value = kwhdays' -- *Length of kwh billing period for average demand calculation. Default 30.*

**kwhdays** [in] Type: double Value; [Property (put)];

'kwhdays = value' -- *(no Help string available)*

**Model** [out, retval] Type: enum LoadModels\*, [Value; [Property (get)]];

'value = Model' -- *The Load Model defines variation of P and Q with voltage.*

**Model** [in] Type: enum LoadModels, [Value; [Property (put)]];

'Model = value' -- *(no Help string available)*

**NumCust** [out, retval] Type: long\* Value; [Property (get)];

'value = NumCust' -- *Number of customers in this load, defaults to one.*

**NumCust** [in] Type: long Value; [Property (put)];

' NumCust = value' -- *(no Help string available)*

**Rneut** [out, retval] Type: double\* Value; [Property (get)];  
'value = Rneut ' -- *Neutral resistance for wye-connected loads.*

**Rneut** [in] Type: double Value; [Property (put)];  
' Rneut = value' -- *(no Help string available)*

**Spectrum** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Spectrum ' -- *Name of harmonic current spectrum shape.*

**Spectrum** [in] Type: BSTR Value; [Property (put)];  
' Spectrum = value' -- *(no Help string available)*

**Vmaxpu** [out, retval] Type: double\* Value; [Property (get)];  
'value = Vmaxpu ' -- *Maximum per-unit voltage to use the load model. Above this, constant Z applies.*

**Vmaxpu** [in] Type: double Value; [Property (put)];  
' Vmaxpu = value' -- *(no Help string available)*

**Vminemerg** [out, retval] Type: double\* Value; [Property (get)];  
'value = Vminemerg ' -- *Minimum voltage for unserved energy (UE*

**Vminemerg** [in] Type: double Value; [Property (put)];  
' Vminemerg = value' -- *(no Help string available)*

**Vminnorm** [out, retval] Type: double\* Value; [Property (get)];  
'value = Vminnorm ' -- *Minimum voltage for energy exceeding normal (EEN*

**Vminnorm** [in] Type: double Value; [Property (put)];  
' Vminnorm = value' -- *(no Help string available)*

**Vminpu** [out, retval] Type: double\* Value; [Property (get)];  
'value = Vminpu ' -- *Minimum voltage to apply the load model. Below this, constant Z is used.*

**Vminpu** [in] Type: double Value; [Property (put)];  
' Vminpu = value' -- *(no Help string available)*

**xfkVA** [out, retval] Type: double\* Value; [Property (get)];  
'value = xfkVA ' -- *Rated service transformer kVA for load allocation, using AllocationFactor. Affects kW, kvar, and pf.*

**xfkVA** [in] Type: double Value; [Property (put)];  
' xfkVA = value' -- *(no Help string available)*

**Xneut** [out, retval] Type: double\* Value; [Property (get)];  
'value = Xneut ' -- *Neutral reactance for wye-connected loads.*

**Xneut** [in] Type: double Value; [Property (put)];  
' Xneut = value' -- *(no Help string available)*

**Yearly** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Yearly ' -- *Name of yearly duration loadshape*

**Yearly** [in] Type: BSTR Value; [Property (put)];  
' Yearly = value' -- *(no Help string available)*

**Status** [out, retval] Type: enum LoadStatus\*, [Value; [Property (get)]];  
'value = Status ' -- *Response to load multipliers: Fixed (growth only)*

**Status** [in] Type: enum LoadStatus, [Value; [Property (put)]];  
' Status = value' -- *(no Help string available)*

**Growth** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Growth ' -- *Name of the growthshape curve for yearly load growth factors.*

**Growth** [in] Type: BSTR Value; [Property (put)];  
' Growth = value' -- *(no Help string available)*

**ZIPV** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = ZIPV ' -- *Array of 7 doubles with values for ZIPV property of the LOAD object*

**ZIPV** [in] Type: VARIANT Value; [Property (put)];  
' ZIPV = value' -- *(no Help string available)*

**pctSeriesRL** [out, retval] Type: double\* Value; [Property (get)];  
'value = pctSeriesRL ' -- *(no Help string available)*

**pctSeriesRL** [in] Type: double Value; [Property (put)];  
' pctSeriesRL = value' -- *Percent of Load that is modeled as series R-L for harmonics studies*

**RelWeight** [out, retval] Type: double\* Value; [Property (get)];  
'value = RelWeight ' -- *Relative Weighting factor for the active LOAD*

**RelWeight** [in] Type: double Value; [Property (put)];  
' RelWeight = value' -- *Relative Weighting factor for the active LOAD*

## DSSElement Interface

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- *Full Name of Active DSS Object (general element or circuit element)*

**Properties** [in] Type: VARIANT Indx, [out, retval] Type: IDSSProperty\*\* Value; [Property (get)];  
'value = Properties ' -- *Collection of properties for Active DSS object (general element or circuit element)*

**NumProperties** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumProperties ' -- *Number of Properties for the active DSS object.*

**AllPropertyNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllPropertyNames ' -- *Variant array of strings containing the names of all properties for the active DSS object.*

## ActiveClass Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllNames ' -- *Variant array of strings consisting of all element names in the active class.*

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- Sets first element in the active class to be the active DSS object. If object is a CktElement, ActiveCktElement also points to this element. Returns 0 if none.

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- Sets next element in active class to be the active DSS object. If object is a CktElement, ActiveCktElement also points to this element. Returns 0 if no more.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- Name of the Active Element of the Active Class

**Name** [in] Type: BSTR Value; [Property (put)];  
' Name = value' -- (no Help string available)

**NumElements** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumElements ' -- Number of elements in this class. Same as Count property.

**ActiveClassName** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = ActiveClassName ' -- Returns name of active class.

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- Number of elements in Active Class. Same as NumElements Property.

## Capacitors Interface

**kV** [out, retval] Type: double\* Value; [Property (get)];  
'value = kV ' -- Bank kV rating. Use LL for 2 or 3 phases, or actual can rating for 1 phase.

**kV** [in] Type: double Value; [Property (put)];  
' kV = value' -- Bank kV rating. Use LL for 2 or 3 phases, or actual can rating for 1 phase.

**kvar** [out, retval] Type: double\* Value; [Property (get)];  
'value = kvar ' -- Total bank KVAR, distributed equally among phases and steps.

**kvar** [in] Type: double Value; [Property (put)];  
' kvar = value' -- Total bank KVAR, distributed equally among phases and steps.

**NumSteps** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumSteps ' -- Number of steps (default 1

**NumSteps** [in] Type: long Value; [Property (put)];  
' NumSteps = value' -- Number of steps (default 1

**IsDelta** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = IsDelta ' -- Delta connection or wye?

**IsDelta** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' IsDelta = value' -- Delta connection or wye?

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllNames ' -- Variant array of strings with all Capacitor names in the circuit.

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- Sets the first Capacitor active. Returns 0 if no more.

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- Sets the next Capacitor active. Returns 0 if no more.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- Sets the active Capacitor by Name.

**Name** [in] Type: BSTR Value; [Property (put)];  
' Name = value' -- Sets the active Capacitor by Name.

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- Number of Capacitor objects in active circuit.

**AddStep** [out, retval] Type: VARIANT\_BOOL\* value; [Method];  
' AddStep(arg list) ' -- Adds one step of the capacitor if available. If successful returns TRUE.

**SubtractStep** [out, retval] Type: VARIANT\_BOOL\* Value; [Method];  
' SubtractStep(arg list) ' -- Subtract one capacitor step. Returns False if no more steps.

**AvailableSteps** [out, retval] Type: long\* Value; [Property (get)];  
'value = AvailableSteps ' -- Number of Steps available in cap bank to be switched ON.

**States** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = States ' -- A variant array of integer [0..numsteps-1] indicating state of each step. If value is -1 an error has occurred.

**States** [in] Type: VARIANT Value; [Property (put)];  
' States = value' -- Variant array of integer [0 ..numSteps-1] indicating the state of each step

**Open** [void; [Method];  
' Open(arg list) ' -- Open all steps, all phases of the Capacitor

**Close** [void; [Method];  
' Close(arg list) ' -- Close all phases of all steps of the Capacitor

## Transformers Interface

**NumWindings** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumWindings ' -- Number of windings on this transformer. Allocates memory; set or change this property first.

**NumWindings** [in] Type: long Value; [Property (put)];  
' NumWindings = value' -- Number of windings on this transformer. Allocates memory; set or change this property first.

**XfmrCode** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = XfmrCode ' -- Name of an XfmrCode that supplies electrical parameters for this Transformer.

**XfmrCode** [in] Type: BSTR Value; [Property (put)];  
' XfmrCode = value' -- Name of an XfmrCode that supplies electrical parameters for this Transformer.

**Wdg** [out, retval] Type: long\* Value; [Property (get)];

'value = Wdg ' -- Active Winding Number from 1..NumWindings. Update this before reading or setting a sequence of winding properties (R, Tap, kV, kVA, etc.

**Wdg** [in] Type: long Value; [Property (put)];

' Wdg = value' -- Active Winding Number from 1..NumWindings. Update this before reading or setting a sequence of winding properties (R, Tap, kV, kVA, etc.

**R** [out, retval] Type: double\* Value; [Property (get)];

'value = R ' -- Active Winding resistance in %

**R** [in] Type: double Value; [Property (put)];

' R = value' -- Active Winding resistance in %

**Tap** [out, retval] Type: double\* Value; [Property (get)];

'value = Tap ' -- Active Winding tap in per-unit.

**Tap** [in] Type: double Value; [Property (put)];

' Tap = value' -- Active Winding tap in per-unit.

**MinTap** [out, retval] Type: double\* Value; [Property (get)];

'value = MinTap ' -- Active Winding minimum tap in per-unit.

**MinTap** [in] Type: double Value; [Property (put)];

' MinTap = value' -- Active Winding minimum tap in per-unit.

**MaxTap** [out, retval] Type: double\* Value; [Property (get)];

'value = MaxTap ' -- Active Winding maximum tap in per-unit.

**MaxTap** [in] Type: double Value; [Property (put)];

' MaxTap = value' -- Active Winding maximum tap in per-unit.

**NumTaps** [out, retval] Type: long\* Value; [Property (get)];

'value = NumTaps ' -- Active Winding number of tap steps between MinTap and MaxTap.

**NumTaps** [in] Type: long Value; [Property (put)];

' NumTaps = value' -- Active Winding number of tap steps between MinTap and MaxTap.

**kV** [out, retval] Type: double\* Value; [Property (get)];

'value = kV ' -- Active Winding kV rating. Phase-phase for 2 or 3 phases, actual winding kV for 1 phase transformer.

**kV** [in] Type: double Value; [Property (put)];

' kV = value' -- Active Winding kV rating. Phase-phase for 2 or 3 phases, actual winding kV for 1 phase transformer.

**kVA** [out, retval] Type: double\* Value; [Property (get)];

'value = kVA ' -- Active Winding kVA rating. On winding 1, this also determines normal and emergency current ratings for all windings.

**kVA** [in] Type: double Value; [Property (put)];

' kVA = value' -- Active Winding kVA rating. On winding 1, this also determines normal and emergency current ratings for all windings.

**Xneut** [out, retval] Type: double\* Value; [Property (get)];

'value = Xneut ' -- Active Winding neutral reactance [ohms] for wye connections.

**Xneut** [in] Type: double Value; [Property (put)];  
'Xneut = value' -- Active Winding neutral reactance [ohms] for wye connections.

**Rneut** [out, retval] Type: double\* Value; [Property (get)];  
'value = Rneut' -- Active Winding neutral resistance [ohms] for wye connections. Set less than zero for ungrounded wye.

**Rneut** [in] Type: double Value; [Property (put)];  
'Rneut = value' -- Active Winding neutral resistance [ohms] for wye connections. Set less than zero for ungrounded wye.

**IsDelta** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = IsDelta' -- Active Winding delta or wye connection?

**IsDelta** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
'IsDelta = value' -- Active Winding delta or wye connection?

**Xhl** [out, retval] Type: double\* Value; [Property (get)];  
'value = Xhl' -- Percent reactance between windings 1 and 2, on winding 1 kVA base. Use for 2-winding or 3-winding transformers.

**Xhl** [in] Type: double Value; [Property (put)];  
'Xhl = value' -- Percent reactance between windings 1 and 2, on winding 1 kVA base. Use for 2-winding or 3-winding transformers.

**Xht** [out, retval] Type: double\* Value; [Property (get)];  
'value = Xht' -- Percent reactance between windings 1 and 3, on winding 1 kVA base. Use for 3-winding transformers only.

**Xht** [in] Type: double Value; [Property (put)];  
'Xht = value' -- Percent reactance between windings 1 and 3, on winding 1 kVA base. Use for 3-winding transformers only.

**Xlt** [out, retval] Type: double\* Value; [Property (get)];  
'value = Xlt' -- Percent reactance between windings 2 and 3, on winding \_1\_ kVA base. Use for 3-winding transformers only.

**Xlt** [in] Type: double Value; [Property (put)];  
'Xlt = value' -- Percent reactance between windings 2 and 3, on winding \_1\_ kVA base. Use for 3-winding transformers only.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name' -- Sets a Transformer active by Name and 3, on winding \_1\_ kVA base. Use for 3-winding transformers only.

**Name** [in] Type: BSTR Value; [Property (put)];  
'Name = value' -- Sets a Transformer active by Name and 3, on winding \_1\_ kVA base. Use for 3-winding transformers only.

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First' -- Sets the first Transformer active. Returns 0 if no more.

**Next** [out, retval] Type: long\* Value; [Property (get)];



'value = Next ' -- Sets the next Transformer active. Returns 0 if no more.

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames ' -- Variant array of strings with all Transformer names in the active circuit.

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- (no Help string available)

## SwtControls Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames ' -- Variant array of strings with all SwtControl names in the active circuit.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name ' -- Sets a SwtControl active by Name.

**Name** [in] Type: BSTR Value; [Property (put)];

' Name = value' -- Sets a SwtControl active by Name.

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First ' -- Sets the first SwtControl active. Returns 0 if no more.

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next ' -- Sets the next SwtControl active. Returns 0 if no more.

**Action** [out, retval] Type: enum ActionCodes\*, [Value; [Property (get)]];

'value = Action ' -- Open or Close the switch. No effect if switch is locked. However, Reset removes any lock and then closes the switch (shelf state

**Action** [in] Type: enum ActionCodes, [Value; [Property (put)]];

' Action = value' -- Open or Close the switch. No effect if switch is locked. However, Reset removes any lock and then closes the switch (shelf state

**IsLocked** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = IsLocked ' -- The lock prevents both manual and automatic switch operation.

**IsLocked** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' IsLocked = value' -- The lock prevents both manual and automatic switch operation.

**Delay** [out, retval] Type: double\* Value; [Property (get)];

'value = Delay ' -- Time delay [s] between arming and opening or closing the switch. Control may reset before actually operating the switch.

**Delay** [in] Type: double Value; [Property (put)];

' Delay = value' -- Time delay [s] between arming and opening or closing the switch. Control may reset before actually operating the switch.

**SwitchedObj** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = SwitchedObj ' -- Full name of the switched element.

**SwitchedObj** [in] Type: BSTR Value; [Property (put)];

' SwitchedObj = value' -- Full name of the switched element.

**SwitchedTerm** [out, retval] Type: long\* Value; [Property (get)];

'value = SwitchedTerm ' -- *Terminal number where the switch is located on the SwitchedObj*  
**SwitchedTerm** [in] Type: long Value; [Property (put)];  
 ' SwitchedTerm = value' -- *Terminal number where the switch is located on the SwitchedObj*  
**Count** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Count ' -- *(no Help string available)*

## CapControls Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = AllNames ' -- *Variant array of strings with all CapControl names.*  
**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Name ' -- *Sets a CapControl active by name.*  
**Name** [in] Type: BSTR Value; [Property (put)];  
 ' Name = value' -- *Sets a CapControl active by name.*  
**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First ' -- *Sets the first CapControl as active. Return 0 if none.*  
**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next ' -- *Gets the next CapControl in the circuit. Returns 0 if none.*  
**Mode** [out, retval] Type: enum CapControlModes\*, [Value; [Property (get)]];  
 'value = Mode ' -- *Type of automatic controller.*  
**Mode** [in] Type: enum CapControlModes, [Value; [Property (put)]];  
 ' Mode = value' -- *Type of automatic controller.*  
**Capacitor** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Capacitor ' -- *Name of the Capacitor that is controlled.*  
**Capacitor** [in] Type: BSTR Value; [Property (put)];  
 ' Capacitor = value' -- *Name of the Capacitor that is controlled.*  
**MonitoredObj** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = MonitoredObj ' -- *Full name of the element that PT and CT are connected to.*  
**MonitoredObj** [in] Type: BSTR Value; [Property (put)];  
 ' MonitoredObj = value' -- *Full name of the element that PT and CT are connected to.*  
**MonitoredTerm** [out, retval] Type: long\* Value; [Property (get)];  
 'value = MonitoredTerm ' -- *Terminal number on the element that PT and CT are connected to.*  
**MonitoredTerm** [in] Type: long Value; [Property (put)];  
 ' MonitoredTerm = value' -- *Terminal number on the element that PT and CT are connected to.*  
**CTratio** [out, retval] Type: double\* Value; [Property (get)];  
 'value = CTratio ' -- *Transducer ratio from primary current to control current.*  
**CTratio** [in] Type: double Value; [Property (put)];  
 ' CTratio = value' -- *Transducer ratio from primary current to control current.*  
**PTratio** [out, retval] Type: double\* Value; [Property (get)];

'value = PTratio ' -- *Transducer ratio from primary feeder to control voltage.*

**PTratio** [in] Type: double Value; [Property (put)];

' PTratio = value' -- *Transducer ratio from primary feeder to control voltage.*

**ONSetting** [out, retval] Type: double\* Value; [Property (get)];

'value = ONSetting ' -- *Threshold to arm or switch on a step. See Mode for units.*

**ONSetting** [in] Type: double Value; [Property (put)];

' ONSetting = value' -- *Threshold to arm or switch on a step. See Mode for units.*

**OFFSetting** [out, retval] Type: double\* Value; [Property (get)];

'value = OFFSetting ' -- *Threshold to switch off a step. See Mode for units.*

**OFFSetting** [in] Type: double Value; [Property (put)];

' OFFSetting = value' -- *Threshold to switch off a step. See Mode for units.*

**Vmax** [out, retval] Type: double\* Value; [Property (get)];

'value = Vmax ' -- *With VoltOverride, switch off whenever PT voltage exceeds this level.*

**Vmax** [in] Type: double Value; [Property (put)];

' Vmax = value' -- *With VoltOverride, switch off whenever PT voltage exceeds this level.*

**Vmin** [out, retval] Type: double\* Value; [Property (get)];

'value = Vmin ' -- *With VoltOverride, switch ON whenever PT voltage drops below this level.*

**Vmin** [in] Type: double Value; [Property (put)];

' Vmin = value' -- *With VoltOverride, switch ON whenever PT voltage drops below this level.*

**UseVoltOverride** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = UseVoltOverride ' -- *Enables Vmin and Vmax to override the control Mode*

**UseVoltOverride** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' UseVoltOverride = value' -- *Enables Vmin and Vmax to override the control Mode*

**Delay** [out, retval] Type: double\* Value; [Property (get)];

'value = Delay ' -- *Time delay [s] to switch on after arming. Control may reset before actually switching.*

**Delay** [in] Type: double Value; [Property (put)];

' Delay = value' -- *Time delay [s] to switch on after arming. Control may reset before actually switching.*

**DelayOff** [out, retval] Type: double\* Value; [Property (get)];

'value = DelayOff ' -- *Time delay [s] before swithcing off a step. Control may reset before actually switching.*

**DelayOff** [in] Type: double Value; [Property (put)];

' DelayOff = value' -- *Time delay [s] before swithcing off a step. Control may reset before actually switching.*

**DeadTime** [out, retval] Type: double\* Value; [Property (get)];

'value = DeadTime ' -- *(no Help string available)*

**DeadTime** [in] Type: double Value; [Property (put)];

' DeadTime = value' -- *(no Help string available)*

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- *Number of CapControls in Active Circuit*

## RegControls Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllNames ' -- *Variant array of strings containing all RegControl names*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- *Get/set Active RegControl name*

**Name** [in] Type: BSTR Value; [Property (put)];  
' Name = value' -- *Sets a RegControl active by name*

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- *Sets the first RegControl active. Returns 0 if none.*

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- *Sets the next RegControl active. Returns 0 if none.*

**MonitoredBus** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = MonitoredBus ' -- *Name of a remote regulated bus, in lieu of LDC settings*

**MonitoredBus** [in] Type: BSTR Value; [Property (put)];  
' MonitoredBus = value' -- *Name of a remote regulated bus, in lieu of LDC settings*

**Transformer** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Transformer ' -- *Name of the transformer this regulator controls*

**Transformer** [in] Type: BSTR Value; [Property (put)];  
' Transformer = value' -- *Name of the transformer this regulator controls*

**TapWinding** [out, retval] Type: long\* Value; [Property (get)];  
'value = TapWinding ' -- *Tapped winding number*

**TapWinding** [in] Type: long Value; [Property (put)];  
' TapWinding = value' -- *Tapped winding number*

**Winding** [out, retval] Type: long\* Value; [Property (get)];  
'value = Winding ' -- *Winding number for PT and CT connections*

**Winding** [in] Type: long Value; [Property (put)];  
' Winding = value' -- *Winding number for PT and CT connections*

**CTPrimary** [out, retval] Type: double\* Value; [Property (get)];  
'value = CTPrimary ' -- *CT primary ampere rating (secondary is 0.2 amperes*

**CTPrimary** [in] Type: double Value; [Property (put)];  
' CTPrimary = value' -- *CT primary ampere rating (secondary is 0.2 amperes*

**PTratio** [out, retval] Type: double\* Value; [Property (get)];  
'value = PTratio ' -- *PT ratio for voltage control settings*

**PTratio** [in] Type: double Value; [Property (put)];  
' PTratio = value' -- *PT ratio for voltage control settings*

**ForwardR** [out, retval] Type: double\* Value; [Property (get)];  
'value = ForwardR ' -- LDC R setting in Volts

**ForwardR** [in] Type: double Value; [Property (put)];  
' ForwardR = value' -- LDC R setting in Volts

**ForwardX** [out, retval] Type: double\* Value; [Property (get)];  
'value = ForwardX ' -- LDC X setting in Volts

**ForwardX** [in] Type: double Value; [Property (put)];  
' ForwardX = value' -- LDC X setting in Volts

**ReverseR** [out, retval] Type: double\* Value; [Property (get)];  
'value = ReverseR ' -- Reverse LDC R setting in Volts.

**ReverseR** [in] Type: double Value; [Property (put)];  
' ReverseR = value' -- Reverse LDC R setting in Volts.

**ReverseX** [out, retval] Type: double\* Value; [Property (get)];  
'value = ReverseX ' -- Reverse LDC X setting in volts.

**ReverseX** [in] Type: double Value; [Property (put)];  
' ReverseX = value' -- Reverse LDC X setting in volts.

**IsReversible** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = IsReversible ' -- Regulator can use different settings in the reverse direction. Usually not applicable to substation transformers.

**IsReversible** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' IsReversible = value' -- Regulator can use different settings in the reverse direction. Usually not applicable to substation transformers.

**IsInverseTime** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
'value = IsInverseTime ' -- Time delay is inversely adjusted, proportional to the amount of voltage outside the regulating band.

**IsInverseTime** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
' IsInverseTime = value' -- Time delay is inversely adjusted, proportional to the amount of voltage outside the regulating band.

**Delay** [out, retval] Type: double\* Value; [Property (get)];  
'value = Delay ' -- Time delay [s] after arming before the first tap change. Control may reset before actually changing taps.

**Delay** [in] Type: double Value; [Property (put)];  
' Delay = value' -- Time delay [s] after arming before the first tap change. Control may reset before actually changing taps.

**TapDelay** [out, retval] Type: double\* Value; [Property (get)];  
'value = TapDelay ' -- Time delay [s] for subsequent tap changes in a set. Control may reset before actually changing taps.

**TapDelay** [in] Type: double Value; [Property (put)];  
' TapDelay = value' -- Time delay [s] for subsequent tap changes in a set. Control may reset before

*actually changing taps.*

**MaxTapChange** [out, retval] Type: long\* Value; [Property (get)];

'value = MaxTapChange ' -- *Maximum tap change per iteration in STATIC solution mode. 1 is more realistic, 16 is the default for a faster solution.*

**MaxTapChange** [in] Type: long Value; [Property (put)];

'MaxTapChange = value' -- *Maximum tap change per iteration in STATIC solution mode. 1 is more realistic, 16 is the default for a faster solution.*

**VoltageLimit** [out, retval] Type: double\* Value; [Property (get)];

'value = VoltageLimit ' -- *First house voltage limit on PT secondary base. Setting to 0 disables this function.*

**VoltageLimit** [in] Type: double Value; [Property (put)];

'VoltageLimit = value' -- *First house voltage limit on PT secondary base. Setting to 0 disables this function.*

**ForwardBand** [out, retval] Type: double\* Value; [Property (get)];

'value = ForwardBand ' -- *Regulation bandwidth in forward direction, centered on Vreg*

**ForwardBand** [in] Type: double Value; [Property (put)];

'ForwardBand = value' -- *Regulation bandwidth in forward direction, centered on Vreg*

**ForwardVreg** [out, retval] Type: double\* Value; [Property (get)];

'value = ForwardVreg ' -- *Target voltage in the forward direction, on PT secondary base.*

**ForwardVreg** [in] Type: double Value; [Property (put)];

'ForwardVreg = value' -- *Target voltage in the forward direction, on PT secondary base.*

**ReverseBand** [out, retval] Type: double\* Value; [Property (get)];

'value = ReverseBand ' -- *Bandwidth in reverse direction, centered on reverse Vreg.*

**ReverseBand** [in] Type: double Value; [Property (put)];

'ReverseBand = value' -- *Bandwidth in reverse direction, centered on reverse Vreg.*

**ReverseVreg** [out, retval] Type: double\* Value; [Property (get)];

'value = ReverseVreg ' -- *Target voltage in the reverse direction, on PT secondary base.*

**ReverseVreg** [in] Type: double Value; [Property (put)];

'ReverseVreg = value' -- *Target voltage in the reverse direction, on PT secondary base.*

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- *Number of RegControl objects in Active Circuit*

**TapNumber** [out, retval] Type: long\* Value; [Property (get)];

'value = TapNumber ' -- *(no Help string available)*

**TapNumber** [in] Type: long Value; [Property (put)];

'TapNumber = value' -- *Integer number of the tap that the controlled transformer winding is currently on.*

## Topology Interface

**NumLoops** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumLoops ' -- *Number of loops*

**NumIsolatedBranches** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumIsolatedBranches ' -- *Number of isolated branches (PD elements and capacitors)*

**AllLoopedPairs** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllLoopedPairs ' -- *Variant array of all looped element names, by pairs.*

**AllIsolatedBranches** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllIsolatedBranches ' -- *Variant array of all isolated branch names.*

**NumIsolatedLoads** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumIsolatedLoads ' -- *Number of isolated loads*

**AllIsolatedLoads** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllIsolatedLoads ' -- *Variant array of all isolated load names.*

**BranchName** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = BranchName ' -- *Name of the active branch.*

**BranchName** [in] Type: BSTR Value; [Property (put)];  
' BranchName = value ' -- *(no Help string available)*

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- *Sets the first branch active, returns 0 if none.*

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- *Sets the next branch active, returns 0 if no more.*

**ActiveBranch** [out, retval] Type: long\* Value; [Property (get)];  
'value = ActiveBranch ' -- *Returns index of the active branch*

**ForwardBranch** [out, retval] Type: long\* Value; [Property (get)];  
'value = ForwardBranch ' -- *Move forward in the tree, return index of new active branch or 0 if no more*

**BackwardBranch** [out, retval] Type: long\* Value; [Property (get)];  
'value = BackwardBranch ' -- *MOve back toward the source, return index of new active branch, or 0 if no more.*

**LoopedBranch** [out, retval] Type: long\* Value; [Property (get)];  
'value = LoopedBranch ' -- *Move to looped branch, return index or 0 if none.*

**ParallelBranch** [out, retval] Type: long\* Value; [Property (get)];  
'value = ParallelBranch ' -- *Move to directly parallel branch, return index or 0 if none.*

**FirstLoad** [out, retval] Type: long\* Value; [Property (get)];  
'value = FirstLoad ' -- *First load at the active branch, return index or 0 if none.*

**NextLoad** [out, retval] Type: long\* Value; [Property (get)];  
'value = NextLoad ' -- *Next load at the active branch, return index or 0 if no more.*

**ActiveLevel** [out, retval] Type: long\* Value; [Property (get)];  
'value = ActiveLevel ' -- *Topological depth of the active branch*

**BusName** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = BusName ' -- *(no Help string available)*  
**BusName** [in] Type: BSTR Value; *[Property (put)]*;  
 ' BusName = value' -- *Set the active branch to one containing this bus, return index or 0 if not found*

## DSS\_Executive Interface

**NumCommands** [out, retval] Type: long\* Value; *[Property (get)]*;  
 'value = NumCommands ' -- *Number of DSS Executive Commands*  
**NumOptions** [out, retval] Type: long\* Value; *[Property (get)]*;  
 'value = NumOptions ' -- *Number of DSS Executive Options*  
**Command** [in] Type: long i, [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = Command ' -- *Get i-th command*  
**Option** [in] Type: long i, [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = Option ' -- *Get i-th option*  
**CommandHelp** [in] Type: long i, [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = CommandHelp ' -- *Get help string for i-th command*  
**OptionHelp** [in] Type: long i, [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = OptionHelp ' -- *Get help string for i-th option*  
**OptionValue** [in] Type: long i, [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = OptionValue ' -- *Get present value of i-th option*

## DSSEvents Interface

### Sensors Interface

**Name** [out, retval] Type: BSTR\* Value; *[Property (get)]*;  
 'value = Name ' -- *Name of the active sensor.*  
**Name** [in] Type: BSTR Value; *[Property (put)]*;  
 ' Name = value' -- *Set the active Sensor by name.*  
**Count** [out, retval] Type: long\* Value; *[Property (get)]*;  
 'value = Count ' -- *Number of Sensors in Active Circuit.*  
**First** [out, retval] Type: long\* Value; *[Property (get)]*;  
 'value = First ' -- *Sets the first sensor active. Returns 0 if none.*  
**Next** [out, retval] Type: long\* Value; *[Property (get)]*;  
 'value = Next ' -- *Sets the next Sensor active. Returns 0 if no more.*  
**AllNames** [out, retval] Type: VARIANT\* Value; *[Property (get)]*;  
 'value = AllNames ' -- *Variant array of Sensor names.*  
**IsDelta** [out, retval] Type: VARIANT\_BOOL\* Value; *[Property (get)]*;



'value = IsDelta ' -- *True if measured voltages are line-line. Currents are always line currents.*

**IsDelta** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' IsDelta = value' -- *(no Help string available)*

**ReverseDelta** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = ReverseDelta ' -- *True if voltage measurements are 1-3, 3-2, 2-1.*

**ReverseDelta** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' ReverseDelta = value' -- *(no Help string available)*

**PctError** [out, retval] Type: double\* Value; [Property (get)];

'value = PctError ' -- *Assumed percent error in the Sensor measurement. Default is 1.*

**PctError** [in] Type: double Value; [Property (put)];

' PctError = value' -- *(no Help string available)*

**Weight** [out, retval] Type: double\* Value; [Property (get)];

'value = Weight ' -- *Weighting factor for this Sensor measurement with respect to other Sensors. Default is 1.*

**Weight** [in] Type: double Value; [Property (put)];

' Weight = value' -- *(no Help string available)*

**MeteredElement** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = MeteredElement ' -- *Full Name of the measured element*

**MeteredElement** [in] Type: BSTR Value; [Property (put)];

' MeteredElement = value' -- *(no Help string available)*

**MeteredTerminal** [out, retval] Type: long\* Value; [Property (get)];

'value = MeteredTerminal ' -- *Number of the measured terminal in the measured element.*

**MeteredTerminal** [in] Type: long Value; [Property (put)];

' MeteredTerminal = value' -- *(no Help string available)*

**Reset** [void; [Method];

' Reset(arg list) ' -- *Clear the active Sensor.*

**ResetAll** [void; [Method];

' ResetAll(arg list) ' -- *Clear all Sensors in the Active Circuit.*

**kVbase** [out, retval] Type: double\* Value; [Property (get)];

'value = kVbase ' -- *Voltage base for the sensor measurements. LL for 2 and 3-phase sensors, LN for 1-phase sensors.*

**kVbase** [in] Type: double Value; [Property (put)];

' kVbase = value' -- *(no Help string available)*

**Currents** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Currents ' -- *Array of doubles for the line current measurements; don't use with kWS and kVARS.*

**Currents** [in] Type: VARIANT Value; [Property (put)];

' Currents = value' -- *(no Help string available)*

**kVS** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = kVS ' -- Array of doubles for the LL or LN (depending on Delta connection  
**kVS** [in] Type: VARIANT Value; [Property (put)];  
 ' kVS = value' -- (no Help string available)  
**kVARS** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = kVARS ' -- Array of doubles for Q measurements. Overwrites Currents with a new estimate using kWS.  
**kVARS** [in] Type: VARIANT Value; [Property (put)];  
 ' kVARS = value' -- (no Help string available)  
**kWS** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = kWS ' -- Array of doubles for P measurements. Overwrites Currents with a new estimate using kVARS.  
**kWS** [in] Type: VARIANT Value; [Property (put)];  
 ' kWS = value' -- (no Help string available)

## XYCurves Interface

**Count** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Count ' -- Number of XYCurve Objects  
**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First ' -- Sets first XYcurve object active; returns 0 if none.  
**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next ' -- Advances to next XYCurve object; returns 0 if no more objects of this class  
**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Name ' -- Name of active XYCurve Object  
**Name** [in] Type: BSTR Value; [Property (put)];  
 ' Name = value' -- Get Name of active XYCurve Object  
**Npts** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Npts ' -- Get/Set Number of points in X-Y curve  
**Npts** [in] Type: long Value; [Property (put)];  
 ' Npts = value' -- Get/Set Number of Points in X-Y curve  
**Xarray** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = Xarray ' -- Get/Set X values as a Variant array of doubles. Set Npts to max number expected if setting  
**Xarray** [in] Type: VARIANT Value; [Property (put)];  
 ' Xarray = value' -- Get/Set X values as a Variant array of doubles. Set Npts to max number expected if setting  
**Yarray** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = Yarray ' -- Get/Set Y values in curve; Set Npts to max number expected if setting  
**Yarray** [in] Type: VARIANT Value; [Property (put)];

' Yarray = value' -- *Get/Set Y values in curve; Set Npts to max number expected if setting*

**x** [out, retval] Type: double\* Value; [Property (get)];

'value = x ' -- *Set X value or get interpolated value after setting Y*

**x** [in] Type: double Value; [Property (put)];

' x = value' -- *(no Help string available)*

**y** [out, retval] Type: double\* Value; [Property (get)];

'value = y ' -- *Y value for present X or set this value then get corresponding X*

**y** [in] Type: double Value; [Property (put)];

' y = value' -- *Set Y value or get interpolated Y value after setting X*

**Xshift** [out, retval] Type: double\* Value; [Property (get)];

'value = Xshift ' -- *Amount to shift X value from original curve*

**Xshift** [in] Type: double Value; [Property (put)];

' Xshift = value' -- *(no Help string available)*

**Yshift** [out, retval] Type: double\* Value; [Property (get)];

'value = Yshift ' -- *amount to shift Y value from original curve*

**Yshift** [in] Type: double Value; [Property (put)];

' Yshift = value' -- *(no Help string available)*

**Xscale** [out, retval] Type: double\* Value; [Property (get)];

'value = Xscale ' -- *Factor to scale X values from original curve*

**Xscale** [in] Type: double Value; [Property (put)];

' Xscale = value' -- *Factor to scale X values from original curve*

**Yscale** [out, retval] Type: double\* Value; [Property (get)];

'value = Yscale ' -- *Factor to scale Y values from original curve*

**Yscale** [in] Type: double Value; [Property (put)];

' Yscale = value' -- *Amount to scale Y values from original curve. Represents a curve shift.*

## PDElements Interface

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- *Number of PD elements (including disabled elements)*

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First ' -- *Set the first enabled PD element to be the active element. Returns 0 if none found.*

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next ' -- *Advance to the next PD element in the circuit. Enabled elements only. Returns 0 when no more elements.*

**IsShunt** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = IsShunt ' -- *Variant boolean indicating of PD element should be treated as a shunt element rather than a series element. Applies to Capacitor and Reactor elements in particular.*

**FaultRate** [out, retval] Type: double\* Value; [Property (get)];  
'value = FaultRate ' -- *Get/Set Number of failures per year. For LINE elements: Number of failures per unit length per year.*

**FaultRate** [in] Type: double Value; [Property (put)];  
' FaultRate = value' -- *(no Help string available)*

**pctPermanent** [out, retval] Type: double\* Value; [Property (get)];  
'value = pctPermanent ' -- *Get/Set percent of faults that are permanent (require repair*

**pctPermanent** [in] Type: double Value; [Property (put)];  
' pctPermanent = value' -- *(no Help string available)*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- *Get/Set name of active PD Element. Returns null string if active element is not PDElement type.*

**Name** [in] Type: BSTR Value; [Property (put)];  
' Name = value' -- *(no Help string available)*

**Lambda** [out, retval] Type: double\* Value; [Property (get)];  
'value = Lambda ' -- *Failure rate for this branch. Faults per year including length of line.*

**AccumulatedL** [out, retval] Type: double\* Value; [Property (get)];  
'value = AccumulatedL ' -- *accumulated failure rate for this branch on downline*

**RepairTime** [out, retval] Type: double\* Value; [Property (get)];  
'value = RepairTime ' -- *Average time to repair a permanent fault on this branch, hours.*

**Numcustomers** [out, retval] Type: long\* Value; [Property (get)];  
'value = Numcustomers ' -- *Number of customers, this branch*

**Totalcustomers** [out, retval] Type: long\* Value; [Property (get)];  
'value = Totalcustomers ' -- *Total number of customers from this branch to the end of the zone*

**ParentPDElement** [out, retval] Type: long\* Value; [Property (get)];  
'value = ParentPDElement ' -- *Sets the parent PD element to be the active circuit element. Returns 0 if no more elements upline.*

**FromTerminal** [out, retval] Type: long\* Value; [Property (get)];  
'value = FromTerminal ' -- *Number of the terminal of active PD element that is on the \i0*

**TotalMiles** [out, retval] Type: double\* Value; [Property (get)];  
'value = TotalMiles ' -- *Total miles of line from this element to the end of the zone. For recloser siting algorithm.*

**SectionID** [out, retval] Type: long\* Value; [Property (get)];  
'value = SectionID ' -- *Integer ID of the feeder section that this PDElement branch is part of*

## Reclosers Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllNames ' -- *Variant array of strings with names of all Reclosers in Active Circuit*

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- *Number of Reclosers in active circuit.*

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- *Set First Recloser to be Active Ckt Element. Returns 0 if none.*

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- *Iterate to the next recloser in the circuit. Returns zero if no more.*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = Name ' -- *Get Name of active Recloser or set the active Recloser by name.*

**Name** [in] Type: BSTR Value; [Property (put)];  
'Name = value' -- *(no Help string available)*

**MonitoredObj** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = MonitoredObj ' -- *Full name of object this Recloser is monitoring.*

**MonitoredObj** [in] Type: BSTR Value; [Property (put)];  
'MonitoredObj = value' -- *Set monitored object by full name.*

**MonitoredTerm** [out, retval] Type: long\* Value; [Property (get)];  
'value = MonitoredTerm ' -- *Terminal number of Monitored object for the Recloser*

**MonitoredTerm** [in] Type: long Value; [Property (put)];  
'MonitoredTerm = value' -- *(no Help string available)*

**SwitchedObj** [out, retval] Type: BSTR\* Value; [Property (get)];  
'value = SwitchedObj ' -- *Full name of the circuit element that is being switched by the Recloser.*

**SwitchedObj** [in] Type: BSTR Value; [Property (put)];  
'SwitchedObj = value' -- *(no Help string available)*

**SwitchedTerm** [out, retval] Type: long\* Value; [Property (get)];  
'value = SwitchedTerm ' -- *Terminal number of the controlled device being switched by the Recloser*

**SwitchedTerm** [in] Type: long Value; [Property (put)];  
'SwitchedTerm = value' -- *(no Help string available)*

**NumFast** [out, retval] Type: long\* Value; [Property (get)];  
'value = NumFast ' -- *Number of fast shots*

**NumFast** [in] Type: long Value; [Property (put)];  
'NumFast = value' -- *(no Help string available)*

**Shots** [out, retval] Type: long\* Value; [Property (get)];  
'value = Shots ' -- *Number of shots to lockout (fast + delayed*

**Shots** [in] Type: long Value; [Property (put)];  
'Shots = value' -- *(no Help string available)*

**RecloseIntervals** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = RecloseIntervals ' -- *Variant Array of Doubles: reclose intervals, s, between shots.*

**PhaseTrip** [out, retval] Type: double\* Value; [Property (get)];  
'value = PhaseTrip ' -- *Phase trip curve multiplier or actual amps*

**PhaseTrip** [in] Type: double Value; [Property (put)];  
 'PhaseTrip = value' -- *Phase Trip multiplier or actual amps*

**PhaseInst** [out, retval] Type: double\* Value; [Property (get)];  
 'value = PhaseInst' -- *Phase instantaneous curve multiplier or actual amps*

**PhaseInst** [in] Type: double Value; [Property (put)];  
 'PhaseInst = value' -- *(no Help string available)*

**GroundTrip** [out, retval] Type: double\* Value; [Property (get)];  
 'value = GroundTrip' -- *Ground (3I0)*

**GroundTrip** [in] Type: double Value; [Property (put)];  
 'GroundTrip = value' -- *(no Help string available)*

**GroundInst** [out, retval] Type: double\* Value; [Property (get)];  
 'value = GroundInst' -- *Ground (3I0)*

**GroundInst** [in] Type: double Value; [Property (put)];  
 'GroundInst = value' -- *Ground (3I0)*

**Open** [void; [Method];  
 'Open(arg list)' -- *Open recloser's controlled element and lock out the recloser*

**Close** [void; [Method];  
 'Close(arg list)' -- *Close the switched object controlled by the recloser. Resets recloser to first operation.*

**idx** [out, retval] Type: long\* Value; [Property (get)];  
 'value = idx' -- *Get/Set the active Recloser by index into the recloser list. 1..Count*

**idx** [in] Type: long Value; [Property (put)];  
 'idx = value' -- *Get/Set the Active Recloser by index into the recloser list. 1..Count*

## Relays Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = AllNames' -- *Variant array of strings containing names of all Relay elements*

**Count** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Count' -- *Number of Relays in circuit*

**First** [out, retval] Type: long\* Value; [Property (get)];  
 'value = First' -- *Set First Relay active. If none, returns 0.*

**Next** [out, retval] Type: long\* Value; [Property (get)];  
 'value = Next' -- *Advance to next Relay object. Returns 0 when no more relays.*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = Name' -- *Get name of active relay.*

**Name** [in] Type: BSTR Value; [Property (put)];  
 'Name = value' -- *Set Relay active by name*

**MonitoredObj** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = MonitoredObj ' -- *Full name of object this Relay is monitoring.*

**MonitoredObj** [in] Type: BSTR Value; [Property (put)];

' MonitoredObj = value' -- *(no Help string available)*

**MonitoredTerm** [out, retval] Type: long\* Value; [Property (get)];

'value = MonitoredTerm ' -- *Number of terminal of monitored element that this Relay is monitoring.*

**MonitoredTerm** [in] Type: long Value; [Property (put)];

' MonitoredTerm = value' -- *(no Help string available)*

**SwitchedObj** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = SwitchedObj ' -- *Full name of element that will be switched when relay trips.*

**SwitchedObj** [in] Type: BSTR Value; [Property (put)];

' SwitchedObj = value' -- *(no Help string available)*

**SwitchedTerm** [out, retval] Type: long\* Value; [Property (get)];

'value = SwitchedTerm ' -- *(no Help string available)*

**SwitchedTerm** [in] Type: long Value; [Property (put)];

' SwitchedTerm = value' -- *Terminal number of the switched object that will be opened when the relay trips.*

**idx** [out, retval] Type: long\* Value; [Property (get)];

'value = idx ' -- *Get/Set active Relay by index into the Relay list. 1..Count*

**idx** [in] Type: long Value; [Property (put)];

' idx = value' -- *Get/Set Relay active by index into relay list. 1..Count*

## CmathLib Interface

**cmplx** [in] Type: double RealPart, [in] Type: double ImagPart, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = cmplx ' -- *Convert real and imaginary doubles to Variant array of doubles*

**cabs** [in] Type: double realpart, [in] Type: double imagpart, [out, retval] Type: double\* Value; [Property (get)];

'value = cabs ' -- *Return abs value of complex number given in real and imag doubles*

**cdang** [in] Type: double RealPart, [in] Type: double ImagPart, [out, retval] Type: double\* Value; [Property (get)];

'value = cdang ' -- *Returns the angle, in degrees, of a complex number specified as two doubles: Realpart and imagpart.*

**ctopolardeg** [in] Type: double RealPart, [in] Type: double ImagPart, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = ctopolardeg ' -- *Convert complex number to magnitude and angle, degrees. Returns variant array of two doubles.*

**pdegtocomplex** [in] Type: double magnitude, [in] Type: double angle, [out, retval] Type:

VARIANT\* Value; [Property (get)];  
 'value = pdegtocomplex ' -- Convert magnitude, angle in degrees to a complex number. Returns Variant array of two doubles.  
**cmul** [in] Type: double a1, [in] Type: double b1, [in] Type: double a2, [in] Type: double b2, [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = cmul ' -- Multiply two complex numbers: (a1, b1  
**cdiv** [in] Type: double a1, [in] Type: double b1, [in] Type: double a2, [in] Type: double b2, [out, retval] Type: VARIANT\* Value; [Property (get)];  
 'value = cdiv ' -- Divide two complex number: (a1, b1

## Parser Interface

**CmdString** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = CmdString ' -- String to be parsed. Loading this string resets the Parser to the beginning of the line. Then parse off the tokens in sequence.  
**CmdString** [in] Type: BSTR Value; [Property (put)];  
 ' CmdString = value' -- String to be parsed. Loading this string resets the Parser to the beginning of the line. Then parse off the tokens in sequence.  
**NextParam** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = NextParam ' -- Get next token and return tag name (before = sign  
**AutoIncrement** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];  
 'value = AutoIncrement ' -- Default is FALSE. If TRUE parser automatically advances to next token after DblValue, IntValue, or StrValue. Simpler when you don't need to check for parameter names.  
**AutoIncrement** [in] Type: VARIANT\_BOOL Value; [Property (put)];  
 ' AutoIncrement = value' -- Default is FALSE. If TRUE parser automatically advances to next token after DblValue, IntValue, or StrValue. Simpler when you don't need to check for parameter names.  
**DblValue** [out, retval] Type: double\* Value; [Property (get)];  
 'value = DblValue ' -- Return next parameter as a double.  
**IntValue** [out, retval] Type: long\* Value; [Property (get)];  
 'value = IntValue ' -- Return next parameter as a long integer.  
**StrValue** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = StrValue ' -- Return next parameter as a string  
**WhiteSpace** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = WhiteSpace ' -- Get the characters used for White space in the command string. Default is blank and Tab.  
**WhiteSpace** [in] Type: BSTR Value; [Property (put)];  
 ' WhiteSpace = value' -- Set the characters used for White space in the command string. Default



is blank and Tab.

**BeginQuote** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = BeginQuote' -- Get String containing the the characters for Quoting in OpenDSS scripts. Matching pairs defined in EndQuote. Default is \i0

**BeginQuote** [in] Type: BSTR Value; [Property (put)];

'BeginQuote = value' -- Set String containing the the characters for Quoting in OpenDSS scripts. Matching pairs defined in EndQuote. Default is \i0

**EndQuote** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = EndQuote' -- String containing characters, in order, that match the beginning quote characters in BeginQuote. Default is \i0

**EndQuote** [in] Type: BSTR Value; [Property (put)];

'EndQuote = value' -- String containing characters, in order, that match the beginning quote characters in BeginQuote. Default is \i0

**Delimiters** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Delimiters' -- String defining hard delimiters used to separate token on the command string. Default is , and =. The = separates token name from token value. These override whitespace to separate tokens.

**Delimiters** [in] Type: BSTR Value; [Property (put)];

'Delimiters = value' -- String defining hard delimiters used to separate token on the command string. Default is , and =. The = separates token name from token value. These override whitespace to separate tokens.

**ResetDelimiters** [void; [Method];

'ResetDelimiters(arg list)' -- Reset delimiters to their default values.

**Vector** [in] Type: long ExpectedSize, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Vector' -- Returns token as variant array of doubles. For parsing quoted array syntax.

**Matrix** [in] Type: long ExpectedOrder, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = Matrix' -- Use this property to parse a Matrix token in OpenDSS format. Returns square matrix of order specified. Order same as default Fortran order: column by column.

**SymMatrix** [in] Type: long ExpectedOrder, [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = SymMatrix' -- Use this property to parse a matrix token specified in lower triangle form. Symmetry is forced.

## LoadShapes Interface

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name' -- Get the Name of the active Loadshape

**Name** [in] Type: BSTR Value; [Property (put)];

'Name = value' -- Set the active Loadshape by name

**Count** [out, retval] Type: long\* Value; [Property (get)];  
'value = Count ' -- Number of Loadshape objects currently defined in Loadshape collection

**First** [out, retval] Type: long\* Value; [Property (get)];  
'value = First ' -- Set the first loadshape active and return integer index of the loadshape. Returns 0 if none.

**Next** [out, retval] Type: long\* Value; [Property (get)];  
'value = Next ' -- Advance active Loadshape to the next on in the collection. Returns 0 if no more loadshapes.

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = AllNames ' -- Variant array of strings containing names of all Loadshape objects currently defined.

**Npts** [out, retval] Type: long\* Value; [Property (get)];  
'value = Npts ' -- Get Number of points in active Loadshape.

**Npts** [in] Type: long Value; [Property (put)];  
' Npts = value' -- Set number of points to allocate for active Loadshape.

**Pmult** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Pmult ' -- Variant array of Doubles for the P multiplier in the Loadshape.

**Pmult** [in] Type: VARIANT Value; [Property (put)];  
' Pmult = value' -- Variant array of doubles containing the P array for the Loadshape.

**Qmult** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = Qmult ' -- Variant array of doubles containing the Q multipliers.

**Qmult** [in] Type: VARIANT Value; [Property (put)];  
' Qmult = value' -- Variant array of doubles containing the Q multipliers.

**Normalize** [void; [Method];  
' Normalize(arg list) ' -- Normalize the P and Q curves based on either Pbase, Qbase or simply the peak value of the curve.

**TimeArray** [out, retval] Type: VARIANT\* Value; [Property (get)];  
'value = TimeArray ' -- Time array in hours corresponding to P and Q multipliers when the Interval=0.

**TimeArray** [in] Type: VARIANT Value; [Property (put)];  
' TimeArray = value' -- Time array in hours corresponding to P and Q multipliers when the Interval=0.

**HrInterval** [out, retval] Type: double\* Value; [Property (get)];  
'value = HrInterval ' -- Fixed interval time value, hours

**HrInterval** [in] Type: double Value; [Property (put)];  
' HrInterval = value' -- Fixed interval time value, hours.

**MinInterval** [out, retval] Type: double\* Value; [Property (get)];  
'value = MinInterval ' -- Fixed Interval time value, in minutes

**MinInterval** [in] Type: double Value; [Property (put)];

' MinInterval = value' -- *Fixed Interval time value, in minutes*

**New** [in] Type: BSTR Name; [Method];

' New(arg list)' -- *Make a new Loadshape*

**Pbase** [out, retval] Type: double\* Value; [Property (get)];

'value = Pbase' -- *Base for normalizing P curve. If left at zero, the peak value is used.*

**Pbase** [in] Type: double Value; [Property (put)];

' Pbase = value' -- *Base for normalizing P curve. If left at zero, the peak value is used.*

**Qbase** [out, retval] Type: double\* Value; [Property (get)];

'value = Qbase' -- *Base for normalizing Q curve. If left at zero, the peak value is used.*

**Qbase** [in] Type: double Value; [Property (put)];

' Qbase = value' -- *Base for normalizing Q curve. If left at zero, the peak value is used.*

**UseActual** [out, retval] Type: VARIANT\_BOOL\* Value; [Property (get)];

'value = UseActual' -- *T/F flag to let Loads know to use the actual value in the curve rather than use the value as a multiplier.*

**UseActual** [in] Type: VARIANT\_BOOL Value; [Property (put)];

' UseActual = value' -- *T/F flag to let Loads know to use the actual value in the curve rather than use the value as a multiplier.*

**Sinterval** [out, retval] Type: double\* Value; [Property (get)];

'value = Sinterval' -- *Fixed interval data time interval, seconds*

**Sinterval** [in] Type: double Value; [Property (put)];

' Sinterval = value' -- *Fixed interval data time interval, seconds*

## Fuses Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames' -- *Variant array of strings containing names of all Fuses in the circuit*

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count' -- *Number of Fuse elements in the circuit*

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First' -- *Set the first Fuse to be the active fuse. Returns 0 if none.*

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next' -- *Advance the active Fuse element pointer to the next fuse. Returns 0 if no more fuses.*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name' -- *Get the name of the active Fuse element*

**Name** [in] Type: BSTR Value; [Property (put)];

' Name = value' -- *Set the active Fuse element by name.*

**MonitoredObj** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = MonitoredObj' -- *Full name of the circuit element to which the fuse is connected.*

**MonitoredObj** [in] Type: BSTR Value; [Property (put)];  
 ' MonitoredObj = value' -- *Full name of the circuit element to which the fuse is connected.*

**MonitoredTerm** [out, retval] Type: long\* Value; [Property (get)];  
 'value = MonitoredTerm ' -- *Terminal number to which the fuse is connected.*

**MonitoredTerm** [in] Type: long Value; [Property (put)];  
 ' MonitoredTerm = value' -- *Number of the terminal to which the fuse is connected*

**SwitchedObj** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = SwitchedObj ' -- *Full name of the circuit element switch that the fuse controls. Defaults to the MonitoredObj.*

**SwitchedObj** [in] Type: BSTR Value; [Property (put)];  
 ' SwitchedObj = value' -- *Full name of the circuit element switch that the fuse controls. Defaults to MonitoredObj.*

**SwitchedTerm** [out, retval] Type: long\* Value; [Property (get)];  
 'value = SwitchedTerm ' -- *Number of the terminal containing the switch controlled by the fuse.*

**SwitchedTerm** [in] Type: long Value; [Property (put)];  
 ' SwitchedTerm = value' -- *Number of the terminal of the controlled element containing the switch controlled by the fuse.*

**TCCcurve** [out, retval] Type: BSTR\* Value; [Property (get)];  
 'value = TCCcurve ' -- *Name of the TCCcurve object that determines fuse blowing.*

**TCCcurve** [in] Type: BSTR Value; [Property (put)];  
 ' TCCcurve = value' -- *Name of the TCCcurve object that determines fuse blowing.*

**RatedCurrent** [out, retval] Type: double\* Value; [Property (get)];  
 'value = RatedCurrent ' -- *Multiplier or actual amps for the TCCcurve object. Defaults to 1.0. Multiply current values of TCC curve by this to get actual amps.*

**RatedCurrent** [in] Type: double Value; [Property (put)];  
 ' RatedCurrent = value' -- *Multiplier or actual fuse amps for the TCC curve. Defaults to 1.0. Has to correspond to the Current axis of TCCcurve object.*

**Delay** [out, retval] Type: double\* Value; [Property (get)];  
 'value = Delay ' -- *A fixed delay time in seconds added to the fuse blowing time determined by the TCC curve. Default is 0.*

**Delay** [in] Type: double Value; [Property (put)];  
 ' Delay = value' -- *Fixed delay time in seconds added to the fuse blowing time to represent fuse clear or other delay.*

**Open** [void; [Method];  
 ' Open(arg list) ' -- *Manual opening of fuse*

**Close** [void; [Method];  
 ' Close(arg list) ' -- *Close the fuse back in and reset.*

**IsBlown** [void; [Method];  
 ' IsBlown(arg list) ' -- *Current state of the fuses. TRUE if any fuse on any phase is blown. Else*

*FALSE.*

**idx** [out, retval] Type: long\* Value; [Property (get)];

'value = idx' -- Get/set active fuse by index into the list of fuses. 1 based: 1..count

**idx** [in] Type: long Value; [Property (put)];

'idx = value' -- Set Fuse active by index into the list of fuses. 1..count

**NumPhases** [out, retval] Type: long\* Value; [Property (get)];

'value = NumPhases' -- Number of phases, this fuse.

## ISources Interface

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames' -- Variant array of strings containing names of all ISOURCE elements.

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count' -- Count: Number of ISOURCE elements.

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First' -- Set the First ISOURCE to be active; returns Zero if none.

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next' -- Sets the next ISOURCE element to be the active one. Returns Zero if no more.

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name' -- Get name of active ISOURCE

**Name** [in] Type: BSTR Value; [Property (put)];

'Name = value' -- Set Active ISOURCE by name

**Amps** [out, retval] Type: double\* Value; [Property (get)];

'value = Amps' -- Get the magnitude of the ISOURCE in amps

**Amps** [in] Type: double Value; [Property (put)];

'Amps = value' -- Set the magnitude of the ISOURCE, amps

**AngleDeg** [out, retval] Type: double\* Value; [Property (get)];

'value = AngleDeg' -- Phase angle for ISOURCE, degrees

**AngleDeg** [in] Type: double Value; [Property (put)];

'AngleDeg = value' -- Phase angle for ISOURCE, degrees

**Frequency** [out, retval] Type: double\* Value; [Property (get)];

'value = Frequency' -- The present frequency of the ISOURCE, Hz

**Frequency** [in] Type: double Value; [Property (put)];

'Frequency = value' -- Set the present frequency for the ISOURCE

## DSSimComs Interface

**BusVoltagepu** [in] Type: unsigned int, [Index] Type: out, retval VARIANT\*, [Vpu]; [Method];

'BusVoltagepu(arg list)' -- This method delivers the voltage pu of the specified bus, this

*specification must be with a number (index*

**BusVoltage** [in] Type: unsigned int, [Index] Type: out, retval VARIANT\*, [Voltages; [Method];  
' BusVoltage(arg list) ' -- *This method delivers the voltage (complex*

## **PVSystems Interface**

**AllNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = AllNames ' -- *Vairant array of strings with all PVSystem names*

**RegisterNames** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = RegisterNames ' -- *Variant Array of PVSYSYSTEM energy meter register names*

**RegisterValues** [out, retval] Type: VARIANT\* Value; [Property (get)];

'value = RegisterValues ' -- *Variant array of doubles containing values in PVSystem registers.*

**First** [out, retval] Type: long\* Value; [Property (get)];

'value = First ' -- *Set first PVSystem active; returns 0 if none.*

**Next** [out, retval] Type: long\* Value; [Property (get)];

'value = Next ' -- *Sets next PVSystem active; returns 0 if no more.*

**Count** [out, retval] Type: long\* Value; [Property (get)];

'value = Count ' -- *Number of PVSystems*

**idx** [out, retval] Type: long\* Value; [Property (get)];

'value = idx ' -- *Get/set active PVSystem by index; 1..Count*

**idx** [in] Type: long Value; [Property (put)];

' idx = value ' -- *Get/Set Active PVSystem by index: 1.. Count*

**Name** [out, retval] Type: BSTR\* Value; [Property (get)];

'value = Name ' -- *Get the name of the active PVSystem*

**Name** [in] Type: BSTR Value; [Property (put)];

' Name = value ' -- *Set the name of the active PVSystem*

**Irradiance** [out, retval] Type: double\* Value; [Property (get)];

'value = Irradiance ' -- *Get the present value of the Irradiance property in W/sq-m*

**Irradiance** [in] Type: double Value; [Property (put)];

' Irradiance = value ' -- *Set the present Irradiance value in W/sq-m*