Direct connection Shared Library (DLL) for OpenDSS

Davis Montenegro, Celso Rocha, Paulo Radatz

Last update 11-11-2021

The direct connection shared library is a DLL that implements the same classes, properties, and methods of the OpenDSS-PM COM interface. This alternative was generated to accelerate the In-process co-simulation between OpenDSS and external software when the client software does not support early bindings connection to COM servers/controls.

Normally, high level programming languages do not support early bindings, which make them use late bindings for data exchanging with COM servers. Late bindings procedures add an important overhead to the co-simulation process specially when executing loops.

So, if your programming language does not support early bindings connection with COM servers, this is the library you should use to accelerate your simulations. This library is called OpenDSSDDirect.dll and can be accessed directly without needing to register it into the OS registry.

However, if your programming language supports early bindings, keep using the COM interface, the simulation speed will be accelerated naturally when using this connection procedure instead of late bindings.

The properties implemented in this library are the same implemented in the COM interface, so, this manual can be used as a reference manual for the classes, properties and methods included in the COM interface.

ActiveClass Interface

This interface implements the ActiveClass (IActiveClass) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

ActiveClassI (Int) Interface

This interface can be used to read/modify the properties of the ActiveClass Class where the values are integers. The structure of the interface is as follows:

```
int32_t ActiveClassI(int32_t Parameter, int32_t argument) ;
```

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: ActiveClass.First
This parameter 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.
Parameter 1: ActiveClass.Next
This parameter sets next 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.
Parameter 2: ActiveClass.NumElements
This parameter gets the number of elements in this class. Same as Count Property.
Parameter 3: ActiveClass.Count
This parameter gets the number of elements in this class. Same as NumElements Property.

ActiveClassS (String) Interface

This interface can be used to read/modify the properties of the ActiveClass Class where the values are strings. The structure of the interface is as follows:

```
CStr ActiveClassI(int32_t Parameter, CStr argument) ;
```

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: ActiveClass.Name read
This parameter gets the name of the active Element of the Active class.
Parameter 1: ActiveClass.Name write
This parameter sets the name of the active Element of the Active class.
Parameter 2: ActiveClass.ActiveClassName
This parameter sets the name of the active Element of the Active class.

ActiveClassV (Variant) Interface

This interface can be used to read/modify the properties of the ActiveClass Class where the values are Variants. The structure of the interface is as follows:

```
void ActiveClassI(int32_t Parameter, VARIANT  *Argument) ;
```

This interface returns a Variant, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: ActiveClass.AllNames
This parameter gets a variant array of strings consisting of all element names in the active Class.

Alternative Interfaces

These interfaces were made to cover particular requests and can be available in future versions of this DLL. The structure of these interfaces can be different from the rest of the interfaces proposed above.

InitAndGetYparams Interface

This interface initializes the YMatrix of the system in case of being necessary. It must be executed before executing the interface *GetCompressedYMatrix* to reduce the possibility of errors. The structure of this interface is as follows:

```
Uint32 InitAndGetYparams(uint32 hY, uint32 nBus, uint32 nNZ);
```

GetCompressedYMatrix Interface

This interface gets the YMatrix of the system in a reduced format by delivering the pointers of row, column and matrix values. The structure of this interface is as follows:

```
Void GetCompressedYMatrix(uint32 hY, uint32 nBus, uint32 nNz, uint32 ColPtr, uint32 RowIdx,
                          uint32 cVals);
```

ZeroInjCurr Interface

This interface commands to OpenDSS to initialize currents vector as an array of zeros, which is basically the first step in the solution algorithm (DoNormalSolution routine). The structure of this interface is as follows:

```
Void ZeroInjCurr();
```

GetSourceInjCurrents Interface

This interface commands to OpenDSS to include the currents injected by voltage/current sources into the currents vector, following the procedure proposed in the solution algorithm (DoNormalSolution routine). The structure of this interface is as follows:

```
Void GetSourceInjCurrents();
```

GetPCInjCurr Interface

This interface commands to OpenDSS to include the currents injected by the power conversion devices into the currents vector, following the procedure proposed in the solution algorithm (DoNormalSolution routine). The structure of this interface is as follows:

Void GetPCInjCurr ();

SystemYChanged Interface

This interface reads/writes the variable *SystemYChanged* (Boolean), which is used in the DoNormalSolution routine. Using this variable, the Y matrix will be recalculated for the next solution iteration. The structure of this interface is as follows:

Int32 SystemYChanged (int32 parameter, int32 argument);

Depending on the value provided in the variable *parameter*, this interface will deliver the current value of *SystemYChanged* or will set the value specified in the variable *argument*.

Parameter 0: SystemYChanged read
The interface will deliver the value of the variable *SystemYChanged* (1=True, 0=False)
Parameter 1: SystemYChanged write
The interface will set the value of the variable *SystemYChanged* using the numeric value provided in the variable argument (1=True, 0=False).

BuildYMatrixD Interface

This interface executes the procedure BuildYMatrix, which is used in the DoNormalSolution routine to rebuild the Y Matrix. The structure of this interface is as follows:

void BuildYMatrixD (int32 BuildOps, int32 AllocateVI);

The values of BuildOps (SERIESONLY=1, WHOLEMATRIX=2) and AllocateVI (TRUE=1, FALSE=0) will have the same effects and interpretation of the function implemented within the OpenDSS code.

UseAuxCurrents Interface

This interface reads/writes the variable *UseAuxCurrents* (Boolean), which is used in the DoNormalSolution routine. The structure of this interface is as follows:

Int32 UseAuxCurrents (int32 parameter, int32 argument);

Depending on the value provided in the variable *parameter*, this interface will deliver the current value of *UseAuxCurrents* or will set the value specified in the variable *argument*.

Parameter 0: UseAuxCurrents read
The interface will deliver the value of the variable *UseAuxCurrents* (1=True, 0=False)
Parameter 1: UseAuxCurrents write
The interface will set the value of the variable *UseAuxCurrents* using the numeric value provided in the variable argument (1=True, 0=False).

AddInAuxCurrents Interface

This interface executes the procedure AddInAuxCurrents, which is used in the DoNormalSolution routine to rebuild the Y Matrix. The structure of this interface is as follows:

void AddInAuxCurrents (int32 SType);

The value of SType (NEWTONSOLVE=1, NORMALSOLVE=0) will have the same effects and interpretation of the function implemented within the OpenDSS code.

getIpointer Interface

This interface delivers the pointer to the vector of currents used within the solution routine of OpenDSS. The structure of this interface is as follows:

void getIpointer (uint32 IVectorPtr);

getVpointer Interface

This interface delivers the pointer to the vector of Voltages used within the solution routine of OpenDSS. The structure of this interface is as follows:

void getVpointer (uint32 VVectorPtr);

SolveSystem Interface

This interface executes the procedure *SolveSystem* used in the solution routine of OpenDSS. The structure of this interface is as follows:

void SolveSystem(uint32 VVectorPtr);

The variable *VVectorPtr* must contain the pointer to the vector of Voltages provided by the interface *getVpointer*.

Bus Interface

This interface implements the Bus (IBus) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

BusI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t BUSI(int32_t Parameter, int32_t Argument);

This interface returns an integer according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Bus.NumNodes

This parameter returns the number of nodes of this bus.

Parameter 1: Bus.ZscRefresh

This parameter recomputes Zsc for active bus for present circuit configuration. Return 1 if the procedure was successful.

Parameter 2: Bus.Coorddefined

This parameter returns 1 if a coordinate has been defined for this bus; otherwise, it will return 0.

Parameter 3: Bus.GetUniqueNodeNumber

This parameter returns a unique node number at the active bus to avoid node collisions and adds it to the node list for the bus. The start number can be specified in the argument.

Parameter 4: Bus.N_Customers

This parameter returns the total number of customers served down line from this bus.

Parameter 5: Bus.SectionID

This parameter returns the integer ID of the feeder section in which this bus is located.

BusF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double BUSF(int32_t Parameter, double Argument);

This interface returns a floating point number (64 bits) according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Bus.kVBase

This parameter returns the base voltage at bus in kV.

Parameter 1: Bus.X - read

This parameter returns the X coordinate for the bus.

Parameter 2: Bus.X - Write

This parameter allows to write the X coordinate for the bus. Returns 0.

Parameter 3: Bus.Y - read

This parameter returns the Y coordinate for the bus.

Parameter 4: Bus.Y - Write

This parameter allows to write the Y coordinate for the bus. Returns 0.

Parameter 5: Bus.Distance

This parameter returns the distance from the energymeter (if non-zero).

Parameter 6: Bus.Lambda

This parameter returns the accumulated failure rate downstream from this bus; faults per year.

Parameter 7: Bus.N_Interrupts

This parameter returns the number of interruptions this bus per year.

Parameter 8: Bus.Int_Duration

This parameter returns the average interruption duration in hours.

Parameter 9: Bus.Cust_interrupts

This parameter returns the annual number of customer interruptions from this bus.

Parameter 10: Bus.Cust_duration

This parameter returns the accumulated customer outage durations.

Parameter 11: Bus.Totalmiles

This parameter returns the total length of line downline from this bus, in miles. For recloser siting algorithm.

Parameter 12: Bus.Latitude read

This parameter returns the GIS latitude assigned to the active bus (if any).

Parameter 13: Bus.Latitude Write

This parameter sets the GIS latitude to the active bus using the value given at the argument.

Parameter 14: Bus.Longitude read

This parameter returns the GIS longitude assigned to the active bus (if any).

Parameter 15: Bus.Longitude Write

This parameter sets the GIS longitude to the active bus using the value given at the argument.

BusS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr BUSS(int32_t Parameter, CStr Argument);
```

This interface returns a string according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Bus.Name

This parameter returns the name of the active bus.

BusV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  BUSV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a variant according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Bus.Voltages

This parameter returns a complex array of voltages at this bus.

Parameter 1: Bus.SeqVoltages

This parameter returns a complex array of Sequence voltages at this bus.

Parameter 2: Bus.Nodes

This parameter returns an integer array of node numbers defined at the bus in same order as the voltages.

Parameter 3: Bus.Voc

This parameter returns the open circuit voltage as complex array.

Parameter 4: Bus.Isc

This parameter returns the short circuit current as complex array.

Parameter 5: Bus.PuVoltages

This parameter returns the voltages in per unit at bus as complex array.

Parameter 6: Bus.ZscMatrix

This parameter returns the complex array of Zsc matrix at bus, column by column.

Parameter 7: Bus.Zsc1

This parameter returns the complex positive-sequence short circuit impedance at bus.

Parameter 8: Bus.Zsc0

This parameter returns the complex zero-sequence short circuit impedance at bus.

Parameter 9: Bus.YscMatrix

This parameter returns the complex array of Ysc matrix at bus, column by column.

Parameter 10: Bus.CplxSeqVoltages

This parameter returns the complex double array of sequence voltages (0, 1, 2) at this bus.

Parameter 11: Bus.VLL

This parameter for 2 and 3 phase buses, returns a variant array of complex numbers representing L-L voltages in volts. Returns -1.0 for 1-phase bus. If more than 3 phases, returns only first 3.

Parameter 12: Bus.PuVLL

This parameter for 2 and 3 phase buses, returns a variant array of complex numbers representing L-L voltages in per unit. Returns -1.0 for 1-phase bus. If more than 3 phases, returns only first 3.

Parameter 13: Bus.VMagAngle

This parameter returns a variant array of doubles containing voltages in magnitude (VLN), angle (deg).

Parameter 14: Bus.PuVMagAngle

This parameter returns a variant array of doubles containing voltages in per unit and angles in degrees.

Parameter 15: Bus.LineList

This parameter returns a variant array of strings containing the names of the lines connected to the active bus. The names of the lines include the class name “Line.”.

Parameter 16: Bus.LoadList

This parameter returns a variant array of strings containing the names of the loads connected to the active bus. The names of the lines include the class name “Load.”.

Parameter 17: Bus. ZSC012Matrix
Variant array of doubles (complex) containing the complete 012 Zsc matrix.
Parameter 18: Bus.AllPCEatBus
Returns an array with the names of all PCE connected to the active bus.
Parameter 19: Bus. AllPDEatBus
Returns an array with the names of all PDE connected to the active bus.

Capacitors Interface

This interface implements the Capacitors (ICapacitors) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

CapacitorsI (Int) Interface

This interface can be used to read/modify the properties of the Capacitors Class where the values are integers. The structure of the interface is as follows:

int32_t CapacitorsI(int32_t Parameter, int32_t argument) ;

This interface returns an integer (signed 32 bits), the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Capacitors.NumSteps read
This parameter gets the number of steps (defaults 1) for distributing and switching the total bank kvar.
Parameter 1: Capacitors.NumSteps write
This parameter sets the number of steps (defaults 1) for distributing and switching the total bank kvar.
Parameter 2: Capacitors.IsDelta read
This parameter gets 1 if delta connection, otherwise will return 0 for distributing and switching the total kvar.
Parameter 3: Capacitors.IsDelta write
This parameter sets (Argument) 1 if delta connection, otherwise will return 0 for distributing and switching the total kvar.
Parameter 4: Capacitors.First
This parameter sets the first capacitor active. Returns 0 if no more.
Parameter 5: Capacitors.Next
This parameter sets the next capacitor active. Returns 0 if no more.
Parameter 6: Capacitors.Count
This parameter gets the number of capacitor objects in active circuit.
Parameter 7: Capacitors.AddStep
This parameter adds one step of the capacitor if available. If successful returns 1.

Parameter 8: Capacitors.SubtractStep
This parameter subtracts one step of the capacitor if available. If no more steps, returns 0.
Parameter 9: Capacitors.AvailableSteps
This parameter gets the number of steps available in cap bank to be switched ON.
Parameter 10: Capacitors.Open
This parameter opens all steps, all phases of the capacitor.
Parameter 11: Capacitors.Close
This parameter closes all steps, all phases of the capacitor.

CapacitorsF (Float) Interface

This interface can be used to read/modify the properties of the Capacitors Class where the values are doubles. The structure of the interface is as follows:

double CapacitorsF(int32_t Parameter, double argument) ;

This interface returns a floating point number (64 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Capacitors.kV read
This parameter gets the bank rating. Use LL for 2 or 3 phases, or actual can rating for 1 phase.
Parameter 1: Capacitors.kV write
This parameter sets the bank rating. Use LL for 2 or 3 phases, or actual can rating for 1 phase.
Parameter 2: Capacitors.kvar read
This parameter gets the total bank kvar, distributed equally among phases and steps.
Parameter 3: Capacitors.kvar write
This parameter sets the total bank kvar, distributed equally among phases and steps.

CapacitorsS (String) Interface

This interface can be used to read/modify the properties of the Capacitors Class where the values are Strings. The structure of the interface is as follows:

CStr CapacitorsS(int32_t Parameter, CStr argument) ;

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Capacitors.Name read
This parameter gets the active capacitor by Name.

Parameter 1: Capacitors.Name write
This parameter sets the active capacitor by Name.

CapacitorsV (Variant) Interface

This interface can be used to read/modify the properties of the Capacitors Class where the values are Variants. The structure of the interface is as follows:

```
void CapacitorsV(int32_t Parameter, VARIANT *Argument) ;
```

This interface returns a Variant, the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Capacitors.AllNames
This parameter gets a variant array of strings with all Capacitor names in the circuit.
Parameter 1: Capacitors.States read
This parameter gets a variant array of integers [0..numsteps-1] indicating the state of each step. If value is -1 and error has occurred.
Parameter 2: Capacitors.States write
This parameter sets a variant array of integers [0..numsteps-1] indicating the state of each step. If value is -1 and error has occurred.

CapControls Interface

This interface implements the CapControls (ICapControls) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

CapControlsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t CapControlsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CapControls.First
This parameter sets the first CapControl active. Returns 0 if no more.
Parameter 1: CapControls.Next
This parameter sets the next CapControl active. Returns 0 if no more.
Parameter 2: CapControls.Mode read
This parameter gets the type of automatic controller (see manual for details).

Parameter 3: CapControls.Mode write

This parameter sets the type of automatic controller (see manual for details).

Parameter 4: CapControls.MonitoredTerm read

This parameter gets the terminal number on the element that PT and CT are connected to.

Parameter 5: CapControls.MonitoredTerm write

This parameter sets the terminal number on the element that PT and CT are connected to.

Parameter 6: CapControls.UseVoltOverride read

This parameter gets if Vmin and Vmax are enabled to override the control Mode.

Parameter 7: CapControls.UseVoltOverride write

This parameter sets if enables Vmin and Vmax to override the control Mode.

Parameter 8: CapControls.Count

This parameter gets the number of CapControls in Active Circuit.

CapControlsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double CapControlsF(int32_t Parameter, double Argument);

This interface returns a floating point number (64 bits) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CapControls.CTRatio read

This parameter gets the transducer ratio current to control current.

Parameter 1: CapControls.CTRatio write

This parameter sets the transducer ratio current to control current.

Parameter 2: CapControls.PTRatio read

This parameter gets the transducer ratio from primary feeder to control voltage.

Parameter 3: CapControls.PTRatio write

This parameter sets the transducer ratio from primary feeder to control voltage.

Parameter 4: CapControls.ONSetting read

This parameter gets the threshold to arm or switch on a step. See Mode for Units.

Parameter 5: CapControls.ONSetting write

This parameter sets the threshold to arm or switch on a step. See Mode for Units.

Parameter 6: CapControls.OFFSetting read

This parameter gets the threshold to switch off a step. See Mode for Units.

Parameter 7: CapControls.OFFSetting write

This parameter sets the threshold to switch off a step. See Mode for Units.

Parameter 8: CapControls.VMax read

This parameter gets the Vmax, this reference with VoltOverride, switch off whenever PT voltage exceeds this level.

Parameter 9: CapControls.VMax write

This parameter sets the Vmax, this reference with VoltOverride, switch off whenever PT voltage exceeds this level.

Parameter 10: CapControls.VMin read

This parameter gets the Vmin, this reference with VoltOverride, switch ON whenever PT voltage drops below this level.

Parameter 11: CapControls.VMin write

This parameter sets the Vmin, this reference with VoltOverride, switch ON whenever PT voltage drops below this level.

Parameter 12: CapControls.Delay read

This parameter gets the time delay [s] to switch on after arming. Control may reset before actually switching.

Parameter 13: CapControls.Delay write

This parameter sets the time delay [s] to switch on after arming. Control may reset before actually switching.

Parameter 14: CapControls.DelayOff read

This parameter gets the time delay [s] before switching off a step. Control may reset before actually switching.

Parameter 15: CapControls.DelayOff write

This parameter sets the time delay [s] before switching off a step. Control may reset before actually switching.

Parameter 16: CapControls.DeadTime read

This parameter gets the time delay [s] after switching off a step. Control may reset before actually switching.

Parameter 17: CapControls.DeadTime write

This parameter sets the time delay [s] after switching off a step. Control may reset before actually switching.

CapControlsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr CapControlsS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CapControls.Name read

This parameter gets the name of the active CapControl.

Parameter 1: CapControls.Name write

This parameter sets a CapControl active by name.

Parameter 2: CapControls.Capacitor read

This parameter gets the name of the capacitor that is controlled.

Parameter 3: CapControls.Capacitor write

This parameter sets the name of the capacitor that is controlled.

Parameter 4: CapControls.MonitoredObj read

This parameter gets the full name of the element that PT and CT are connected to.

Parameter 5: CapControls.MonitoredObj write

This parameter sets the full name of the element that PT and CT are connected to.

CapControlsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void CapControlsV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CapControls.AllNames

This parameter gets a variant array of string with all CapControl names.

Circuit Interface

This interface implements the Circuit (ICIrcuit) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

CircuitI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t CircuitI(int32_t Parameter, int32_t Argument);

This interface returns an integer according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Circuit.NumCktElements

This parameter will deliver the number of CktElements included in the active circuit.

Parameter 1: Circuit.NumBuses

This parameter will deliver the number of buses included in the active circuit.

Parameter 2: Circuit.NumNodes

This parameter will deliver the number of nodes included in the active circuit.

Parameter 3: Circuit.FirstPCElement

This parameter sets the first PCElement to be the active PCElement, as a result, this parameter will deliver the index of the active PCElement (ideally 1).

Parameter 4: Circuit.NextPCElement

This parameter sets the next PCElement to be the active PCElement, as a result, this parameter will deliver the index of the active PCElement (if there is no more it will return a 0).

Parameter 5: Circuit.FirstPDElement

This parameter sets the first PDElement to be the active PDElement, as a result, this parameter will deliver the index of the active PDElement (ideally 1).

Parameter 6: Circuit.NextPDElement

This parameter sets the next PDElement to be the active PDElement, as a result, this parameter will deliver the index of the active PDElement (if there is no more it will return a 0).

Parameter 7: Circuit.Sample

This parameter forces all meters and monitors to take a sample, returns 0.

Parameter 8: Circuit.SaveSample

This parameter forces all meters and monitors to save their sample buffers, returns 0.

Parameter 9: Circuit.SetActiveBusi

This parameter sets active the bus specified by index, which is compatible with the index delivered by AllBusNames, returns 0 it everything ok.

Parameter 10: Circuit.FirstElement

This parameter sets the first Element of the active class to be the active Element, as a result, this parameter will deliver the index of the active Element (0 if none).

Parameter 11: Circuit.NextElement

This parameter sets the next Element of the active class to be the active Element, as a result, this parameter will deliver the index of the active Element (0 if none).

Parameter 12: Circuit.UpdateStorage

This parameter forces all storage classes to update. Typically done after a solution.

Parameter 13: Circuit.ParentPDElement

This parameter sets parent PD Element, if any, to be the active circuit element and returns index > 0 if it fails or not applicable.

Parameter 14: Circuit.EndofTimeStepUpdate

This parameter calls end of time step cleanup routine in solutionalgs.pas. Returns 0.

CircuitF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double CircuitF(int32_t Parameter, double Argument1, double Argument2);
```

This interface returns a floating point number (IEEE754 64 bits) according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Circuit.Capacity

This parameter returns the total capacity of the active circuit. Or this parameter it is necessary to specify the start and increment of the capacity in the arguments argument1 and argument2 respectively.

CircuitS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr CircuitS(int32_t Parameter, CStr Argument);
```

This interface returns a string according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Circuit.Name

This parameter returns the name of the active circuit.

Parameter 1: Circuit.Disable

This parameter allows to disable an element of the active circuit, the element must be specified by name. As a result, this parameter will deliver the string “Ok”.

Parameter 2: Circuit.Enable

This parameter allows to enable an element of the active circuit, the element must be specified by name. As a result, this parameter will deliver the string “Ok”.

Parameter 3: Circuit.SetActiveElement

This parameter allows to activate an element of the active circuit, the element must be specified by name. As a result, this parameter will deliver a string with the index of the active element.

Parameter 4: Circuit.SetActiveBus

This parameter allows to activate a bus of the active circuit, the bus must be specified by name. As a result, this parameter will deliver a string with the index of the active Bus.

Parameter 5: Circuit.SetActiveClass

This parameter allows to activate a Class of the active circuit, the Class must be specified by name. As a result, this parameter will deliver a string with the index of the active Class.

CircuitV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  CircuitV(int32_t Parameter, VARIANT  *Argument, int32_t Argument2);
```

This interface returns a variant according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Circuit.Losses

This parameter returns an array of doubles (two doubles for representing a complex number) with the total losses of the active circuit. Argument2 must be 0.

Parameter 1: Circuit.LineLosses

This parameter returns an array of doubles (two doubles for representing a complex number) with the total Line losses of the active circuit. Argument2 must be 0.

Parameter 2: Circuit.SubstationLosses

This parameter returns an array of doubles (two doubles for representing a complex number) with the total transformer losses of the active circuit. Argument2 must be 0.

Parameter 3: Circuit.TotalPower

This parameter returns an array of doubles (two doubles for representing a complex number) with the total power in watts delivered to the active circuit. Argument2 must be 0.

Parameter 4: Circuit.AllBusVolts

This parameter returns an array of doubles (two doubles for representing a complex number) with the node voltages from the most recent solution. Argument2 must be 0.

Parameter 5: Circuit.AllBusVMag

This parameter returns an array of doubles (magnitude) with the node voltages from the most recent solution. Argument2 must be 0.

Parameter 6: Circuit.AllElementNames

This parameter returns an array of strings with the names of all the elements of the active circuit. Argument2 must be 0.

Parameter 7: Circuit.AllBusNames

This parameter returns an array of strings with the names of all the Buses of the active circuit (See AllNodeNames). Argument2 must be 0.

Parameter 8: Circuit.AllElementLosses

This parameter returns an array of doubles (two doubles for representing a complex number) with the losses in each element of the active circuit. Argument2 must be 0.

Parameter 9: Circuit.AllBusVMagPu

This parameter returns an array of doubles with the voltages in per unit of the most recent solution of the active circuit. Argument2 must be 0.

Parameter 10: Circuit.AllNodeNames

This parameter returns an array of strings containing full name of each node in system in same order as returned by AllBusVolts, etc. Argument2 must be 0.

Parameter 11: Circuit.SystemY

This parameter returns an array of doubles (two doubles for representing a complex number) containing the Y Bus Matrix of the system (after a solution has been performed). Argument2 must be 0.

Parameter 12: Circuit.AllBusDistances

This parameter returns distance from each bus to parent EnergyMeter. Corresponds to sequence in AllBusNames. Argument2 must be 0.

Parameter 13: Circuit.AllNodeDistances

This parameter returns distance from each Node to parent EnergyMeter. Corresponds to sequence in AllBusVmag. Argument2 must be 0.

Parameter 14: Circuit.AllNodeVMagbyPhase

This parameter returns array of doubles representing the voltage magnitudes for nodes on the specified phase. The phase must be specified in the Argument2.

Parameter 15: Circuit.AllNodeVMagPUByPhase

This parameter returns array of doubles representing the voltage magnitudes (in per unit) for nodes on the specified phase. The phase must be specified in the Argument2.

Parameter 16: Circuit.AllNodeDistancesByPhase

This parameter returns array of doubles representing the distances to parent EnergyMeter. Sequence of array corresponds to other node ByPhase properties. Argument2 must contain the number of the phase to return.

Parameter 17: Circuit.AllNodeNamesByPhase

This parameter returns array of strings of the node names by Phase criteria. Sequence corresponds to other ByPhase properties. Argument2 must contain the number of the phase to return.

Parameter 18: Circuit.YNodeVArray

This parameter returns a complex array of actual node voltages in same order as SystemY Matrix. Argument2 must be 0.

Parameter 19: Circuit.YNodeOrder

This parameter returns a variant array of strings containing the names of the nodes in the same order as the Y Matrix. Argument2 must be 0.

Parameter 20: Circuit.YCurrents

This parameter returns a returns a variant array of doubles containing complex injection currents for the present solution. It is the "I" vector of I=YV. Argument2 must be 0.

CktElement Interface

This interface implements the CktElement interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

CktElementI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t CktElementI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CktElement.NumTerminals

This parameter will deliver the number of terminals of the active DSS object.

Parameter 1: CktElement.NumConductors

This parameter will deliver the number of conductors of the active DSS object.

Parameter 2: CktElement.NumPhases

This parameter will deliver the number of phases of the active DSS object.

Parameter 3: CktElement.Open

This parameter will *open* the specified terminal (Argument) of the active DSS object.

Parameter 4: CktElement.Close

This parameter will *close* the specified terminal (Argument) of the active DSS object.

Parameter 5: CktElement.IsOpen

This parameter will return a *1* if any terminal of the active DSS object is open, otherwise, it will return a *0*.

Parameter 6: CktElement.NumProperties

This parameter will return the number of properties of the active DSS object.

Parameter 7: CktElement.HasSwitchControl

This parameter will return 1 if the active DSS object has a Switch Control linked; otherwise, it will return 0.

Parameter 8: CktElement.HasVoltControl

This parameter will return 1 if the active DSS object has a Volt Control linked; otherwise, it will return 0.

Parameter 9: CktElement.NumControls

This parameter will return number of controls linked to the active DSS object.

Parameter 10: CktElement.OCPDevIndex

This parameter will return the Index into Controller list of OCP Device controlling the active DSS object.

Parameter 11: CktElement.OCPDevType

This parameter will return one of the following values: 0=none; 1=Fuse; 2=Recloser; 3=Relay according to the type of active control.

Parameter 12: CktElement.Enabled - read

This parameter will return one of the following values: 0 if the active element is disabled or 1 if the active element is enabled.

Parameter 13: CktElement.Enabled - Write

This parameter allows to modify the enabled status of the active element (1=enabled, 0=disabled).

CktElementF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double CktElementF(int32_t Parameter, double Argument);

This interface returns a float (IEEE 754 64 bits) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CktElement.NormalAmps - read

This parameter will deliver the normal ampere rating for the active PDElement.

Parameter 1: CktElement.NormalAmps - Write

This parameter allows to fix the normal ampere rating for the active PDElement. The new value must be defined in the variable “Argument”.

Parameter 2: CktElement.EmergAmps - read

This parameter will deliver the Emergency ampere rating for the active PDElement.

Parameter 3: CktElement.EmergAmps - Write

This parameter allows to fix the Emergency ampere rating for the active PDElement. The new value must be defined in the variable “Argument”.

Parameter 4: CktElement.Variablei

This parameter delivers get the value of a variable by index for the active PCElement.

CktElementS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr CktElementS(int32_t Parameter, CStr Argument);
```

This interface returns a string (pAnsiChar) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CktElement.Name

This parameter delivers the full name of the active circuit element.

Parameter 1: CktElement.Display - read

This parameter displays the name of the active circuit element (not necessarily unique).

Parameter 2: CktElement.Display - write

This parameter allows to modify the name of the active circuit element (not necessarily unique).

Parameter 3: CktElement.GUID

This parameter delivers the unique name for the active circuit element.

Parameter 4: CktElement.EnergyMeter

This parameter delivers the name of the EnergyMeter linked to the active circuit element.

Parameter 5: CktElement.Controller

This parameter delivers the Full name of the i-th controller attached to the active circuit element. The i-th controller index must be specified in the argument *arg*. Ex: Str = Controller(2). See NumControls to determine valid index range.

CktElementV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void CktElementV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a variant (the format depends on the parameter) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CktElement.BusNames - read

This parameter delivers an array of strings with the names of all the buses connected to the active circuit element.

Parameter 1: CktElement.BusNames - write

This parameter allows to fix an array of strings with the names of all the buses connected to the active circuit element.

Parameter 2: CktElement.Voltages

This parameter delivers an array of doubles with the voltages at terminals of the active circuit element.

Parameter 3: CktElement.Currents

This parameter delivers an array of doubles with the currents at terminals of the active circuit element.

Parameter 4: CktElement.powers

This parameter delivers an array of doubles with the powers at terminals of the active circuit element.

Parameter 5: CktElement.Losses

This parameter delivers an array of doubles with the Losses at terminals of the active circuit element.

Parameter 6: CktElement.PhaseLosses

This parameter delivers an array of doubles with the Losses per phase at the terminals of the active circuit element.

Parameter 7: CktElement.SeqVoltages

This parameter delivers an array of doubles with the symmetrical component voltages per phase at the terminals of the active circuit element.

Parameter 8: CktElement.SeqCurrents

This parameter delivers an array of doubles with the symmetrical component Currents per phase at the terminals of the active circuit element.

Parameter 9: CktElement.SeqPowers

This parameter delivers an array of doubles with the symmetrical component powers per phase at the terminals of the active circuit element.

Parameter 10: CktElement.AllPropertyNames

This parameter delivers an array of strings with the names of all the properties of the active circuit element.

Parameter 11: CktElement.Residuals

This parameter delivers an array of doubles with the residual currents (magnitude, angle) in all the nodes of the active circuit element.

Parameter 12: CktElement.YPrim

This parameter delivers an array of doubles with the Y primitive matrix (complex) of the active circuit element.

Parameter 13: CktElement.CplxSeqVoltages

This parameter delivers an array of doubles with the complex of sequence voltages for all terminals of the active circuit element.

Parameter 14: CktElement.CplxSeqCurrents

This parameter delivers an array of doubles with the complex of sequence currents for all terminals of the active circuit element.

Parameter 15: CktElement.AllVariableNames

This parameter delivers a Variant array of strings listing all the published state variable names, if the active circuit element is a PCElement. Otherwise, null string.

Parameter 16: CktElement.AllVariableValues

This parameter delivers a Variant array of doubles listing all the values of the state variables, if the active circuit element is a PCElement. Otherwise, null string.

Parameter 17: CktElement.Nodeorder

This parameter delivers a Variant array integers variant array of integer containing the node numbers (representing phases, for example) for each conductor of each terminal.

Parameter 18: CktElement.CurrentsMagAng

This parameter delivers the currents in magnitude, angle format as a variant array of doubles of the active circuit element.

Parameter 19: CktElement.VoltagesMagAng

This parameter delivers the voltages in magnitude, angle format as a variant array of doubles of the active circuit element.

Parameter 20: CktElement.TotalPowers

Returns the total powers (complex) at ALL terminals of the active circuit element.

CmathLib Interface

This interface implements the CmathLib (ICmathLib) interface of OpenDSS by declaring 2 procedures for accessing the different properties included in this interface.

CmathLibF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double CmathLibF(int32_t Parameter, double Argument1, double Argument2);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CmathLib.Cabs

This parameter returns the absolute value of complex number given in real (Argument1) and imaginary (Argument2) doubles.

Parameter 1: CmathLib.Cdang

This parameter returns the angle, in degrees, of a complex number specified as two doubles: Real part (Argument1) and imaginary part (Argument2).

CmathLibV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void CmathLibV(int32_t Parameter, double Argument1, Argument2, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: CmathLib.cmplx

This parameter convert real (Argument1) and imaginary (Argument1) doubles to variant array of doubles.

Parameter 1: CmathLib.ctopolardeg

This parameter convert complex number (Argument1 and Argument2) to magnitude and angle, degrees. Returns variant array of two doubles.

Parameter 2: CmathLib.pdegtocomplex

This parameter convert magnitude, angle in degrees (Argument1 and Argument2) to a complex number. Returns variant array of two doubles.

CtrlQueue Interface

This interface implements the CtrlQueue (ICtrlQueue) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

CtrlQueueI (Int) Interface

This interface can be used to read/modify the properties of the CtrlQueue Class where the values are integers. The structure of the interface is as follows:

int32_t CtrlQueueI(int32_t Parameter, int32_t argument)

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: CtrlQueue.ClearQueue

This parameter clears the control queue.

Parameter 1: CtrlQueue.Delete

This parameter deletes a control action from the DSS control queue by referencing the handle of the action (Argument).

Parameter 2: CtrlQueue.NumActions

This parameter gets the number of actions on the current action list (that have been popped off the control queue by CheckControlActions).

Parameter 3: CtrlQueue.Action

This parameter sets the active action by index (argument).

Parameter 4: CtrlQueue.ActionCode

This parameter gets the code for the active action. Long integer code to tell the control device what to do.

Parameter 5: CtrlQueue.DeviceHandle

This parameter gets the handle (user defined) to device that must act on the pending action.

Parameter 6: CtrlQueue.Push

This parameter pushes a control action onto the DSS control queue by time, action code, and device handle. Returns Control Queue handle.

Parameter 7: CtrlQueue.Show

This parameter shows the entire control queue in CSV format.

Parameter 8: CtrlQueue.ClearActions
This parameter clears the action list.
Parameter 9: CtrlQueue.PopAction
This parameter pops next action off the action list and makes it the active action. Returns zero if none.
Parameter 10: CtrlQueue.QueueSize
This parameter delivers the size of the current control queue. Returns zero if none.
Parameter 11: CtrlQueue.DoAllQueue
This parameter forces the execution of all control actions stored at the control queue. Returns 0.

CtrlQueueV (Variant) Interface

This interface can be used to read/modify the properties of the CtrlQueue Class where the values are Variants. The structure of the interface is as follows:

```
void CtrlQueueV(int32_t Parameter, , VARIANT  *Argument);
```

This interface returns a Variant, the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: CtrlQueue.CtrlQueue
This parameter delivers the control actions contained in the CtrlQueue after the latest solve command.

DSS Interface

This interface implements the DSS interface (IDSS) of OpenDSS by declaring 3 procedures for accessing the different properties included in this interface.

DSSI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t DSSI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSS.NumCircuits
This parameter gets the number of circuits currently defined.

Parameter 1: DSS.ClearAll

This parameter clears all circuit definitions.

Parameter 2: DSS.ShowPanel

This parameter shows non-MDI child form of the Main DSS Edit form.

Parameter 3: DSS.Start

This parameter validates the user and start the DSS. Returns TRUE (1) if successful.

Parameter 4: DSS.NumClasses

This parameter gets the number of DSS intrinsic classes.

Parameter 5: DSS.NumUserClasses

This parameter gets the number of user-defined classes.

Parameter 6: DSS.Reset

This parameter resets DSS initialization for restarts, etc. from applets.

Parameter 7: DSS.AllowForms read

This parameter gets if the DSS allows forms (1) or not (0), default (1).

Parameter 8: DSS.AllowForms write

This parameter sets if the DSS allows forms (1) or not (0), default (1).

DSSS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr DSSS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSS.NewCircuit

This parameter makes a new circuit, the name of the circuit must be specified in the Argument.

Parameter 1: DSS.Version

This parameter gets the version string for the DSS.

Parameter 2: DSS.DataPath read

This parameter gets the Data File Path. Default for reports, etc. from DSS.

Parameter 3: DSS.DataPath write

This parameter sets the Data File Path. Default for reports, etc. from DSS.

Parameter 4: DSS.DefaultEditor

This parameter gets the path name for the default text editor.

DSSV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void DSSV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSS.Classes
This parameter gets the list of DSS intrinsic classes (names of the classes).
Parameter 1: DSS.UserClasses
This parameter gets list of user-defined classes (names of the classes).

DSSElement Interface

This interface implements the DSSElement (IDSSElement) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

DSSElementI (Int) Interface

This interface can be used to read/modify the properties of the DSSElement Class where the values are integers. The structure of the interface is as follows:

int32_t DSSElementI(int32_t Parameter, int32_t argument) ;

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: DSSElement.NumProperties
This parameter gets the number of properties for the active DSS object.

DSSElementS (String) Interface

This interface can be used to read/modify the properties of the DSSElement Class where the values are Strings. The structure of the interface is as follows:

CStr DSSElementS(int32_t Parameter, CStr argument) ;

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: DSSElement.Name
This parameter gets the full name of the active DSS object (general element or circuit element).

DSSElementV (Variant) Interface

This interface can be used to read/modify the properties of the DSSElement Class where the values are Variants. The structure of the interface is as follows:

void DSSElementV(int32_t Parameter VARIANT *Argument) ;

This interface returns a Variant, the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: DSSElement.AllPropertyNames
This parameter gets a variant array of strings containing the names of all properties for the active DSS object.

DSSProgress Interface

This interface implements the DSSProgress (IDSSProgress) interface of OpenDSS by declaring 2 procedures for accessing the different properties included in this interface.

DSSProgressI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t DSSProgressI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSSProgress.PctProgress
This parameter sets the percent progress to indicate [0..100].
Parameter 1: DSSProgress.Show
This parameter shows progress form with null caption and progress set to zero.
Parameter 2: DSSProgress.Close
This parameter closes (hides) DSS Progress form.

DSSProgressS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr DSSProgressS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSSProgress.Caption

This parameter sets the caption to appear on the bottom of the DSS Progress form.

DSSProperties Interface

This interface implements the DSSproperties (IDSSProperties) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

DSSProperties Interface

This interface can be used to read/write certain properties of DSS objects. The structure of the interface is as follows:

CStr DSSProperties(int32_t Parameter, CStr Argument);

This interface returns a string pointer (ANSI) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: DSSProperties.Name

This parameter will deliver the name of the active property. This parameter will deliver the name of the active property. The index of the property must be specified in the argument. The index minimum value is 1. This value must be entered as string.

Parameter 1: DSSProperties.Description

This parameter will deliver the description of the active property. This parameter will deliver the name of the active property. The index of the property must be specified in the argument. The index minimum value is 1. This value must be entered as string.

Parameter 2: DSSProperties.Value - read

This parameter will deliver the value of the active property. This parameter will deliver the name of the active property. The index of the property must be specified in the argument. The index minimum value is 1. This value must be entered as string.

Parameter 3: DSSProperties.Value - Write

This parameter will allow to set the value of the active property. The new value must be specified in the variable “argument” as string. This parameter will deliver the name of the active property. The index of the property must be specified in the argument. The index minimum value is 1. This value must be entered as string.

DSS_Executive Interface

This interface implements the DSS_Executive (IDSS_Executive) interface of OpenDSS by declaring 2 procedures for accessing the different properties included in this interface.

DSS_ExecutiveI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t DSSExecutiveI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Executive.NumCommands
This parameter gets the number of DSS Executive Commands.
Parameter 1: Executive.NumOptions
This parameter gets the number of DSS Executive Options.

DSS_ExecutiveS (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr DSSExecutiveS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Executive.Command
This parameter gets i-th command (specified in the argument as string).
Parameter 1: Executive.Option
This parameter gets i-th option (specified in the argument as string).
Parameter 2: Executive.CommandHelp
This parameter gets help string for i-th command (specified in the argument as string).
Parameter 3: Executive.OptionHelp
This parameter gets help string for i-th option (specified in the argument as string).
Parameter 4: Executive.OptionValue
This parameter gets present value for i-th option (specified in the argument as string).

Error Interface

This interface implements the Error interface of OpenDSS by declaring 2 procedures for accessing the different properties included in this interface.

ErrorCode (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t ErrorCode(void);

This interface returns an integer with latest error code delivered by OpenDSS.

ErrorDesc (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr ErrorDesc(void);

This interface returns a string with description of the latest error code delivered by OpenDSS.

Fuses Interface

This interface implements the Fuses (IFuses) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

FusesI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t FusesI(int32_t Parameter, int32_t Argument);

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Fuses.Count
This parameter returns the number of Fuses objects currently defined in the active circuit.
Parameter 1: Fuses.First
This parameter sets the first Fuse to be the active Fuse. Returns 0 if none.
Parameter 2: Fuses.Next
This parameter sets the next Fuse to be the active Fuse. Returns 0 if none.
Parameter 3: Fuses.MonitoredTerm read
This parameter gets the terminal number to switch the fuse is connected.
Parameter 4: Fuses.MonitoredTerm write
This parameter sets the terminal number to switch the fuse is connected.
Parameter 5: Fuses.SwitchedTerm read
This parameter gets the terminal number of the terminal containing the switch controlled by the fuse.
Parameter 6: Fuses.SwitchedTerm write
This parameter sets the terminal number of the terminal containing the switch controlled by the fuse.
Parameter 7: Fuses.Open
Manual opening of fuse.

Parameter 8: Fuses.Close

Manual closing of fuse.

Parameter 9: Fuses.IsBlown

This parameter returns the current state of the fuses. TRUE (1) if any on any phase is blown. Else FALSE (0).

Parameter 10: Fuses.Idx read

This parameter gets the active fuse by index into the list of fuses. 1 based: 1..count.

Parameter 11: Fuses.Idx write

This parameter sets the active fuse by index into the list of fuses. 1 based: 1..count.

Parameter 12: Fuses.NumPhases

This parameter gets the number of phases of the active fuse.

Parameter 13: Fuses.Reset

This parameter resets the active fuse to its normal state.

FusesF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double FusesF(int32_t Parameter, double Argument);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Fuses.RatedCurrent read

This parameter gets the multiplier or actual amps for the TCCcurve object. Defaults to 1.0, Multiply current values of TCC curve by this to get actual amps.

Parameter 1: Fuses.RatedCurrent write

This parameter sets the multiplier or actual amps for the TCCcurve object. Defaults to 1.0, Multiply current values of TCC curve by this to get actual amps.

Parameter 2: Fuses.Delay read

This parameter gets the fixed delay time in seconds added to the fuse blowing time determined by the TCC curve. Default is 0.

Parameter 3: Fuses.Delay write

This parameter sets the fixed delay time in seconds added to the fuse blowing time determined by the TCC curve. Default is 0.

FusesS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr FusesS(int32_t Parameter, CStr Argument);
```

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Fuses.Name read

This parameter gets the name of the active fuse.

Parameter 1: Fuses.Name write

This parameter sets the name of the active fuse.

Parameter 2: Fuses.MonitoredObj read

This parameter gets the name of the Monitored Object by the active fuse.

Parameter 3: Fuses.MonitoredObj write

This parameter sets the name of the Monitored Object by the active fuse.

Parameter 4: Fuses.SwitchedObj read

This parameter gets the full name of the circuit element switch that the fuse controls. Defaults to the MonitoredObj.

Parameter 5: Fuses.SwitchedObj write

This parameter sets the full name of the circuit element switch that the fuse controls. Defaults to the MonitoredObj.

Parameter 6: Fuses.TCCcurve read

This parameter gets the name of the TCCcurve object that determines fuse blowing.

Parameter 7: Fuses.TCCcurve write

This parameter sets the name of the TCCcurve object that determines fuse blowing.

FusesV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void FusesV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Fuses.AllNames

This parameter gets the variant array of string containing names of all fuses in the circuit.

Parameter 1: Fuses.States - read

This property gets a Variant array of strings representing the state of each phase of the fuse.

Parameter 2: Fuses.States - write

This property sets the current state per phase through an array of strings.

Parameter 3: Fuses.NormalState - read

This property gets the normal state of the active fuse (the state forced for the element at the beginning of the simulation).

Parameter 4: Fuses.NormalState - write

This property sets the normal state of the active fuse (the state forced for the element at the beginning of the simulation) through an array of strings per phase of the active fuse.

Generators Interface

This interface implements the Generators (IGenerators) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

GeneratorsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t GeneratorsI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Generators.First

This parameter sets first generator to be active. Returns 0 if None.

Parameter 1: Generators.Next

This parameter sets next generator to be active. Returns 0 if None.

Parameter 2: Generators.ForcedON read

This parameter returns 1 if the generator is forced ON regardless of other dispatch criteria; otherwise, returns 0.

Parameter 3: Generators.ForcedON Write

This parameter allows to force ON regardless of other dispatch criteria. To force ON put 1 in the argument, otherwise put 0.

Parameter 4: Generators.Phases read

This parameter returns the number of phases of the active generator.

Parameter 5: Generators.Phases Write

This parameter sets the number of phases (argument) of the active generator.

Parameter 6: Generators.Count

This parameter returns the number of generators Objects in Active Circuit.

Parameter 7: Generators.Idx read

This parameter gets the active generator by Index into generators list. 1..Count.

Parameter 8: Generators.Idx Write

This parameter sets the active generator (argument) by Index into generators list. 1..Count.

Parameter 9: Generators.Model read

This parameter gets the active generator Model (see Manual for details).

Parameter 10: Generators.Model Write

This parameter sets the active generator Model (see Manual for details).

GeneratorsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double GeneratorsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Generators.kV read

This parameter gets the voltage base for the active generator, kV.

Parameter 1: Generators.kV Write

This parameter sets the voltage base for the active generator, kV.

Parameter 2: Generators.kW read

This parameter gets the kW output for the active generator, kvar is updated for current power factor.

Parameter 3: Generators.kW Write

This parameter sets the kW output for the active generator, kvar is updated for current power factor.

Parameter 4: Generators.kvar read

This parameter gets the kvar output for the active generator, kW is updated for current power factor.

Parameter 5: Generators.kvar Write

This parameter sets the kvar output for the active generator, kW is updated for current power factor.

Parameter 6: Generators.Pf read

This parameter gets the power factor (pos. = producing vars). Updates kvar based on present kW value.

Parameter 7: Generators.Pf Write

This parameter sets the power factor (pos. = producing vars). Updates kvar based on present kW value.

Parameter 8: Generators.KVARated read

This parameter gets the KVA rating of the generator.

Parameter 9: Generators.KVARated Write

This parameter sets the KVA rating of the generator.

Parameter 10: Generators.Vmaxpu read

This parameter gets the Vmaxpu for Generator Model.

Parameter 11: Generators.VMaxpu Write

This parameter sets the Vmaxpu for Generator Model.

Parameter 12: Generators.Vminpu read

This parameter gets the Vminpu for Generator Model.

Parameter 13: Generators.VMinpu Write

This parameter sets the Vminpu for Generator Model.

GeneratorsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr GeneratorsS(int32_t Parameter, CStr Argument);
```

This interface returns a string as a result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Generators.Name read
This parameter returns the active generator's name.
Parameter 1: Generators.Name Write
This parameter sets the active generator's name.

GeneratorsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  GeneratorsV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a Variant as a result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Generators.AllNames
This parameter gets the array of names of all Generator objects.
Parameter 1: Generators.RegisterNames
This parameter gets the array of names of all generator Energy Meter registers.
Parameter 2: Generators.RegisterValues
This parameter gets the array of values in generator Energy Meter registers.

Isources Interface

This interface implements the ISources (IIsources) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

IsourcesI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t IsourcesI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Isources.Count
This parameter returns the number of Isource objects currently defined in the active circuit.

Parameter 1: Isources.First

This parameter sets the first ISource to be active; returns 0 if none.

Parameter 2: Isources.Next

This parameter sets the next ISource to be active; returns 0 if none.

IsourcesF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double IsourcesF(int32_t Parameter, double Argument);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Isources.Amps read

This parameter gets the magnitude of the Isource in Amps.

Parameter 1: Isources.Amps write

This parameter sets the magnitude of the Isource in Amps.

Parameter 2: Isources.AngleDeg read

This parameter gets the phase angle of the Isource in degrees.

Parameter 3: Isources.AngleDeg write

This parameter sets the phase angle of the Isource in degrees.

Parameter 4: Isources.Frequency read

This parameter gets the frequency of the Isource in Hz.

Parameter 5: Isources.Frequency write

This parameter sets the frequency of the Isource in Hz.

IsourcesS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr IsourcesS(int32_t Parameter, CStr Argument);
```

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Isources.Name read

This parameter gets the name of the active Isource object.

Parameter 1: Isources.Name write

This parameter sets the name of the active Isource object.

IsourcesV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void IsourcesV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Isources.AllNames
This parameter gets the variant array of string containing names of all ISources in the circuit.

Lines Interface

This interface implements the Lines (ILines) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

LinesI (Int) Interface

This interface can be used to read/modify the properties of the Lines Class where the values are integers. The structure of the interface is as follows:

int32_t LinesI(int32_t Parameter, int32_t argument)

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Lines.First
This parameter sets the first element active. Returns 0 if no lines. Otherwise, index of the line element.
Parameter 1: Lines.Next
This parameter sets the next element active. Returns 0 if no lines. Otherwise, index of the line element.
Parameter 2: Lines.Phases read
This parameter gets the number of phases of the active line object.
Parameter 3: Lines.Phases write
This parameter sets the number of phases of the active line object.
Parameter 4: Lines.NumCust
This parameter gets the number of customers on this line section.
Parameter 5: Lines.Parent
This parameter gets the parents of the active Line to be the active Line. Return 0 if no parent or action fails.
Parameter 6: Lines.Count
This parameter gets the number of Line Objects in Active Circuit.

Parameter 7: Lines.Units read
This parameter gets the units of the line (distance, check manual for details).
Parameter 8: Lines.Units write
This parameter sets the units of the line (distance, check manual for details).

LinesF (Float) Interface

This interface can be used to read/modify the properties of the Lines Class where the values are doubles. The structure of the interface is as follows:

double LinesF(int32_t Parameter, double argument)

This interface returns a floating point number, the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Lines.Length read
This parameter gets the length of line section in units compatible with the LineCode definition.
Parameter 1: Lines.Length write
This parameter sets the length of line section in units compatible with the LineCode definition.
Parameter 2: Lines.R1 read
This parameter gets the positive sequence resistance, ohm per unit length.
Parameter 3: Lines.R1 write
This parameter sets the positive sequence resistance, ohm per unit length.
Parameter 4: Lines.X1 read
This parameter gets the positive sequence reactance, ohm per unit length.
Parameter 5: Lines.X1 write
This parameter sets the positive sequence reactance, ohm per unit length.
Parameter 6: Lines.R0 read
This parameter gets the zero sequence resistance, ohm per unit length.
Parameter 7: Lines.R0 read
This parameter sets the zero sequence resistance, ohm per unit length.
Parameter 8: Lines.X0 read
This parameter gets the zero sequence reactance, ohm per unit length.
Parameter 9: Lines.X0 write
This parameter sets the zero sequence reactance, ohm per unit length.
Parameter 10: Lines.C1 read
This parameter gets the positive sequence capacitance, nanofarads per unit length.
Parameter 11: Lines.C1 write
This parameter sets the positive sequence capacitance, nanofarads per unit length.
Parameter 12: Lines.C0 read
This parameter gets the zero sequence capacitance, nanofarads per unit length.
Parameter 13: Lines.C0 write
This parameter sets the zero sequence capacitance, nanofarads per unit length.

Parameter 14: Lines.NormAmps read
This parameter gets the normal ampere rating of Line.
Parameter 15: Lines.NormAmps write
This parameter sets the normal ampere rating of Line.
Parameter 16: Lines.EmergAmps read
This parameter gets the emergency (maximum) ampere rating of Line.
Parameter 17: Lines.EmergAmps write
This parameter sets the emergency (maximum) ampere rating of Line.
Parameter 18: Lines.Rg read
This parameter gets the earth return value used to compute line impedances at power frequency.
Parameter 19: Lines.Rg write
This parameter sets the earth return value used to compute line impedances at power frequency.
Parameter 20: Lines.Xg read
This parameter gets the earth return reactance value used to compute line impedances at power frequency.
Parameter 21: Lines.Xg write
This parameter sets the earth return reactance value used to compute line impedances at power frequency.
Parameter 22: Lines.Rho read
This parameter gets the earth resistivity, m-ohms.
Parameter 23: Lines.Rho write
This parameter sets the earth resistivity, m-ohms.
Parameter 24: Lines.SeasonRating
This parameter returns the rating for the current season (in Amps) if the SeasonalRatings option is active.

LinesS (String) Interface

This interface can be used to read/modify the properties of the Lines Class where the values are Strings. The structure of the interface is as follows:

CStr LinesS(int32_t Parameter, CStr argument)

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Lines.Name read
This parameter gets the name of the active Line element.
Parameter 1: Lines.Name write
This parameter sets the name of the Line element to set it active.
Parameter 2: Lines.Bus1 read
This parameter gets the name of bus for terminal 1.

Parameter 3: Lines.Bus1 write
This parameter sets the name of bus for terminal 1.
Parameter 4: Lines.Bus2 read
This parameter gets the name of bus for terminal 2.
Parameter 5: Lines.Bus2 write
This parameter sets the name of bus for terminal 2.
Parameter 6: Lines.LineCode read
This parameter gets the name of LineCode object that defines the impedances.
Parameter 7: Lines.LineCode write
This parameter sets the name of LineCode object that defines the impedances.
Parameter 8: Lines.Geometry read
This parameter gets the name of the Line geometry code.
Parameter 9: Lines.Geometry write
This parameter sets the name of the Line geometry code.
Parameter 10: Lines.Spacing read
This parameter gets the name of the Line spacing code.
Parameter 11: Lines.Spacing write
This parameter sets the name of the Line spacing code.

LinesV (Variant) Interface

This interface can be used to read/modify the properties of the Lines Class where the values are Variants. The structure of the interface is as follows:

```
void LinesV(int32_t Parameter, , VARIANT  *Argument);
```

This interface returns a Variant, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Lines.AllNames
This parameter gets the name of all Line Objects.
Parameter 1: Lines.RMatrix read
This parameter gets the resistance matrix (full), ohms per unit length. Variant array of doubles.
Parameter 2: Lines.RMatrix write
This parameter sets the resistance matrix (full), ohms per unit length. Variant array of doubles.
Parameter 3: Lines.XMatrix read
This parameter gets the reactance matrix (full), ohms per unit length. Variant array of doubles.
Parameter 4: Lines.XMatrix write
This parameter sets the reactance matrix (full), ohms per unit length. Variant array of doubles.
Parameter 5: Lines.CMatrix read
This parameter gets the capacitance matrix (full), nanofarads per unit length. Variant array of doubles.

Parameter 6: Lines.CMatrix write
This parameter sets the capacitance matrix (full), nanofarads per unit length. Variant array of doubles.
Parameter 7: Lines.YPrim read
This parameter gets the YPrimitive of the active Line.
Parameter 8: Lines.YPrim write
This parameter does nothing at present.

LineCodes Interface

This interface implements the Lines (ILineCodes) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

LineCodesI (Int) Interface

This interface can be used to read/modify the properties of the LineCode Class where the values are integers. The structure of the interface is as follows:

int32_t LineCodesI(int32_t Parameter, int32_t argument)

This interface returns an integer (signed 32 bits), the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: LineCodes.Count
This parameter gets the number of Line Objects in Active Circuit.
Parameter 1: LineCodes.First
This parameter sets the first element active. Returns 0 if no lines. Otherwise, index of the line element.
Parameter 2: LineCodes.Next
This parameter sets the next element active. Returns 0 if no lines. Otherwise, index of the line element.
Parameter 3: LineCodes.Units Read
This parameter delivers the units of the active LineCode as an integer.
Parameter 4: LineCodes.Units Write
This parameter sets the units of the active LineCode. The units must be specified as an integer in the argument. Please refer to the OpenDSS User manual for more information.
Parameter 5: LineCodes.Phasess Read
This parameter delivers the number of phases of the active LineCode as an integer.
Parameter 6: LineCodes.Phasess Write
This parameter sets the number of phases of the active LineCode. The units must be specified as an integer in the argument.
Parameter 7: Lines. IsZ1Z0
This parameter gets the flag (Boolean 1/0) denoting whether the impedance data were entered in symmetrical components.

LineCodesF (Float) Interface

This interface can be used to read/modify the properties of the LineCode Class where the values are doubles. The structure of the interface is as follows:

double LineCodesF(int32_t Parameter, double argument)

This interface returns a floating point number, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: LineCodes.R1 read
This parameter gets the Positive-sequence resistance in ohms per unit length for the active LineCode.
Parameter 1: LinesCode.R1 write
This parameter sets the Positive-sequence resistance in ohms per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 2: LineCodes.X1 read
This parameter gets the Positive-sequence reactance in ohms per unit length for the active LineCode.
Parameter 3: LinesCode.X1 write
This parameter sets the Positive-sequence reactance in ohms per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 4: LineCodes.R0 read
This parameter gets the Zero-sequence resistance in ohms per unit length for the active LineCode.
Parameter 5: LinesCode.R0 write
This parameter sets the Zero-sequence resistance in ohms per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 6: LineCodes.X0 read
This parameter gets the Zero-sequence reactance in ohms per unit length for the active LineCode.
Parameter 7: LinesCode.X0 write
This parameter sets the Zero-sequence reactance in ohms per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 8: LineCodes.C1 read
This parameter gets the Positive-sequence capacitance in nanofarads per unit length for the active LineCode.
Parameter 9: LinesCode.C1 write
This parameter sets the Positive-sequence capacitance in nanofarads per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 10: LineCodes.C0 read
This parameter gets the Zero-sequence capacitance in nanofarads per unit length for the active LineCode.

Parameter 11: LinesCode.C0 write
This parameter sets the Zero-sequence capacitance in nanofarads per unit length for the active LineCode. This value must be specified in the argument as a double.
Parameter 12: LineCodes.NormAmps read
This parameter gets the normal ampere rating for the active LineCode.
Parameter 13: LinesCode.NormAmps write
This parameter sets the normal ampere rating for the active LineCode. This value must be specified in the argument as a double.
Parameter 14: LineCodes.EmergAmps read
This parameter gets the Emergency ampere rating for the active LineCode.
Parameter 15: LinesCode.EmergAmps write
This parameter sets the Emergency ampere rating for the active LineCode. This value must be specified in the argument as a double.

LineCodesS (String) Interface

This interface can be used to read/modify the properties of the LineCode Class where the values are Strings. The structure of the interface is as follows:

CStr LineCodesS(int32_t Parameter, CStr argument)

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: LineCodes.Name read
This parameter gets the name of the active LineCode element.
Parameter 1: LineCodes.Name write
This parameter sets the name of the active LineCode element. The new value must be specified in the argument as a string.

LineCodesV (Variant) Interface

This interface can be used to read/modify the properties of the LineCode Class where the values are Variants. The structure of the interface is as follows:

void LineCodesV(int32_t Parameter, , VARIANT *Argument);

This interface returns a Variant, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: LineCodes.Rmatrix read
This parameter gets the resistance matrix in ohms per unit length of the active LineCode.
Parameter 1: LineCodes.Rmatrix Write
This parameter sets the resistance matrix in ohms per unit length of the active LineCode. The new values must be entered as a vector of doubles using the argument.
Parameter 2: LineCodes.Xmatrix read
This parameter gets the reactance matrix in ohms per unit length of the active LineCode.
Parameter 3: LineCodes. Xmatrix Write
This parameter sets the reactance matrix in ohms per unit length of the active LineCode. The new values must be entered as a vector of doubles using the argument.
Parameter 4: LineCodes.Cmatrix read
This parameter gets the capacitance matrix in ohms per unit length of the active LineCode.
Parameter 5: LineCodes.Cmatrix Write
This parameter sets the capacitance matrix in ohms per unit length of the active LineCode. The new values must be entered as a vector of doubles using the argument.
Parameter 6: LineCodes.AllNames
This parameter gets the names of all the existing LineCodes in the project. This information is delivered as an array of strings.

Loads Interface

This interface implements the Loads (ILoads) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

Loads Int (Integers) Interface

This interface can be used to read/modify the properties of the Loads Class where the values are integers. The structure of the interface is as follows:

int32_t DSSLoads(int32_t Parameter, int32_t argument)

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Loads.First
This parameter allows to set the active load into the first load registered in the active circuit. As a result, this property will return the number 1. The parameter argument can be filled with a 0.
Parameter 1: Loads.Next
This parameter sets the active load into the next load registered in the active circuit. As a result, this property will deliver the index of the active load. The parameter argument can be filled with a 0.

Parameter 2: Loads.Idx – Read

This parameter allows to read the index of the active load. The parameter argument can be filled with a 0.

Parameter 3: Load.Idx – Write

This parameter allows to write the index of the active load. The parameter argument must contain the index of the desired active load. The return value will be equal to 0.

Parameter 4: Load.Count

This parameter returns the number of load elements within the active circuit. The parameter argument can be filled with a 0.

Parameter 5: Load.Class – Read

This parameter allows to read the code number used to separate loads by class or group. The parameter argument can be filled with a 0.

Parameter 6: Load.Class – Write

This parameter allows to read the code number used to separate loads by class or group. The parameter argument can be filled with a 0.

Parameter 7: Load.Model – Read

This parameter allows to read the model of the active load. The parameter argument can be filled with a 0.

Parameter 8: Load.Model – Write

This parameter allows to write the model of the active load using the parameter argument. This parameter will return a 0.

Parameter 9: Load.NumCust – Read

This parameter allows to read the number of customer of the active load. The parameter argument can be filled with a 0.

Parameter 10: Load.NumCust – Write

This parameter allows to write the number of customers of the active load using the parameter argument. This parameter will return a 0.

Parameter 11: Load.Status – Read

This parameter allows to read Response to load multipliers: Fixed (growth only –"1"), Exempt (no LD curve - "2"), Variable (all – "0"), of the active load. The parameter argument can be filled with a 0.

Parameter 12: Load.Status – Write

This parameter allows to write the response to load multipliers: Fixed (growth only – "1"), Exempt (no LD curve – "2"), Variable (all – "0"), of the active load using the parameter argument. This parameter will return a 0.

Parameter 13: Load.IsDelta – Read

This parameter allows to read if the active load is connected in delta, if the answer is positive, this function will deliver a 1; otherwise, the answer will be 0. The parameter argument can be filled with a 0.

Parameter 14: Load.IsDelta – Write

This parameter allows to write if the active load is connected in delta, if it is, the argument variable must contain a 1; otherwise, 0. The parameter argument can be filled with a 0. This parameter will return a 0.

Loads F (Float) Interface

This interface can be used to read/modify the properties of the Loads Class where the values are floating point numbers (double). The structure of the interface is as follows:

double DSSLoadsF(int32_t Parameter, double Argument);

This interface returns a Double (IEEE 754 64 bits), the variable “parameter” (Integer) is used to specify the property of the class to be used and the variable “argument” (double) can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Loads.kw – Read

This parameter allows to read the kW property of the active load. The parameter argument can be filled with a 0.

Parameter 1: Load.kw – Write

This parameter allows to write the kW property of the active load. The parameter argument must contain the new value in kW for the desired active load. The return value will be equal to 0.

Parameter 2: Loads.kV – Read

This parameter allows to read the kV property of the active load. The parameter argument can be filled with a 0.

Parameter 3: Load.kV – Write

This parameter allows to write the kV property of the active load. The parameter argument must contain the new value in kV for the desired active load. The return value will be equal to 0.

Parameter 4: Loads.kvar – Read

This parameter allows to read the kvar property of the active load. The parameter argument can be filled with a 0.

Parameter 5: Load.kvar – Write

This parameter allows to write the kvar property of the active load. The parameter argument must contain the new value in kvar for the desired active load. The return value will be equal to 0.

Parameter 6: Loads.pf – Read

This parameter allows to read the pf property of the active load. The parameter argument can be filled with a 0.

Parameter 7: Load.pf – Write

This parameter allows to write the pf property of the active load. The parameter argument must contain the new value in pf for the desired active load. The return value will be equal to 0.

Parameter 8: Loads.PctMean – Read

This parameter allows to read the PctMean property of the active load. The parameter argument can be filled with a 0.

Parameter 9: Load.PctMean – Write

This parameter allows to write the PctMean property of the active load. The parameter argument must contain the new value in PctMean for the desired active load. The return value will be equal to 0.

Parameter 10: Loads.PctStdDev – Read

This parameter allows to read the PctStdDev property of the active load. The parameter argument can be filled with a 0.

Parameter 11: Load.PctStdDev – Write

This parameter allows to write the PctStdDev property of the active load. The parameter argument must contain the new value in PctStdDev for the desired active load. The return value will be equal to 0.

Parameter 12: Loads.AllocationFactor – Read

This parameter allows to read the *AllocationFactor* property of the active load. The parameter argument can be filled with a 0.

Parameter 13: Load. AllocationFactor – Write

This parameter allows to write the *AllocationFactor* property of the active load. The parameter argument must contain the new value in *AllocationFactor* for the desired active load. The return value will be equal to 0.

Parameter 14: Loads.CFactor – Read

This parameter allows to read the *CFactor* property of the active load. The parameter argument can be filled with a 0.

Parameter 15: Load. CFactor – Write

This parameter allows to write the *CFactor* property of the active load. The parameter argument must contain the new value in *CFactor* for the desired active load. The return value will be equal to 0.

Parameter 16: Loads.CVRWatts – Read

This parameter allows to read the *CVRWatts* property of the active load. The parameter argument can be filled with a 0.

Parameter 17: Load.CVRWatts – Write

This parameter allows to write the *CVRWatts* property of the active load. The parameter argument must contain the new value in *CVRWatts* for the desired active load. The return value will be equal to 0.

Parameter 18: Loads.CVRvars – Read

This parameter allows to read the *CVRvars* property of the active load. The parameter argument can be filled with a 0.

Parameter 19: Load.CVRvars – Write

This parameter allows to write the *CVRvars* property of the active load. The parameter argument must contain the new value in *CVRvars* for the desired active load. The return value will be equal to 0.

Parameter 20: Loads.kva – Read

This parameter allows to read the *kva* property of the active load. The parameter argument can be filled with a 0.

Parameter 21: Load.kva – Write

This parameter allows to write the *kva* property of the active load. The parameter argument must contain the new value in *kva* for the desired active load. The return value will be equal to 0.

Parameter 22: Loads.kWh – Read

This parameter allows to read the *kWh* property of the active load. The parameter argument can be filled with a 0.

Parameter 23: Load.kWh – Write

This parameter allows to write the *kWh* property of the active load. The parameter argument must contain the new value in *kWh* for the desired active load. The return value will be equal to 0.

Parameter 24: Loads.kWhdays – Read

This parameter allows to read the *kWhdays* property of the active load. The parameter argument can be filled with a 0.

Parameter 25: Load.kWhdays – Write

This parameter allows to write the *kWhdays* property of the active load. The parameter argument must contain the new value in *kWhdays* for the desired active load. The return value will be equal to 0.

Parameter 26: Loads.RNeut – Read

This parameter allows to read the *RNeut (neutral resistance for wye connected loads)* property of the active load. The parameter argument can be filled with a 0.

Parameter 27: Load.RNeut – Write

This parameter allows to write the *RNeut (neutral resistance for wye connected loads)* property of the active load. The parameter argument must contain the new value in *RNeut* for the desired active load. The return value will be equal to 0.

Parameter 28: Loads.VMaxpu – Read

This parameter allows to read the *VMaxpu* property of the active load. The parameter argument can be filled with a 0.

Parameter 29: Load.VMaxpu – Write

This parameter allows to write the *VMaxpu* property of the active load. The parameter argument must contain the new value in *VMaxpu* for the desired active load. The return value will be equal to 0.

Parameter 30: Loads.VMinemerg – Read

This parameter allows to read the *VMinemerg* property of the active load. The parameter argument can be filled with a 0.

Parameter 31: Load.VMinemerg – Write

This parameter allows to write the *VMinemerg* property of the active load. The parameter argument must contain the new value in *VMinemerg* for the desired active load. The return value will be equal to 0.

Parameter 32: Loads.VMinnorm – Read

This parameter allows to read the *VMinnorm* property of the active load. The parameter argument can be filled with a 0.

Parameter 33: Load.VMinnorm – Write

This parameter allows to write the *VMinnorm* property of the active load. The parameter argument must contain the new value in *VMinnorm* for the desired active load. The return value will be equal to 0.

Parameter 34: Loads.VMinpu – Read

This parameter allows to read the *VMinpu* property of the active load. The parameter argument can be filled with a 0.

Parameter 35: Load.VMinpu – Write

This parameter allows to write the *VMinpu* property of the active load. The parameter argument must contain the new value in *VMinpu* for the desired active load. The return value will be equal to 0.

Parameter 36: Loads.xfKVA – Read

This parameter allows to read the *xfKVA (Rated service transformer KVA for load allocation, using Allocationfactor. Affects kW, kvar and pf.)* property of the active load. The parameter argument can be filled with a 0.

Parameter 37: Load.xfKVA – Write

This parameter allows to write the *xfKVA (Rated service transformer KVA for load allocation, using Allocationfactor. Affects kW, kvar and pf.)* property of the active load. The parameter argument must contain the new value in *xfKVA* for the desired active load. The return value will be equal to 0.

Parameter 38: Loads.Xneut – Read

This parameter allows to read the *Xneut* property of the active load. The parameter argument can be filled with a 0.

Parameter 39: Load. Xneut – Write

This parameter allows to write the *Xneut* property of the active load. The parameter argument must contain the new value in *Xneut* for the desired active load. The return value will be equal to 0.

Parameter 40: Loads.PctSeriesRL – Read

This parameter allows to read the *PctSeriesRL (Percent of Load that is modeled as series R-L for harmonic studies)* property of the active load. The parameter argument can be filled with a 0.

Parameter 41: Load.PctSeriesRL – Write

This parameter allows to write the *PctSeriesRL (Percent of Load that is modeled as series R-L for harmonic studies)* property of the active load. The parameter argument must contain the new value in *PctSeriesRL* for the desired active load. The return value will be equal to 0.

Parameter 42: Loads.RelWeight – Read

This parameter allows to read the *RelWeight (relative weighting factor)* property of the active load. The parameter argument can be filled with a 0.

Parameter 43: Loads.RelWeight – Write

This parameter allows to write the *RelWeight (relative weighting factor)* property of the active load. The parameter argument must contain the new value in *RelWeight* for the desired active load. The return value will be equal to 0.

Loads S (String) Interface

This interface can be used to read/modify the properties of the Loads Class where the values are strings. The structure of the interface is as follows:

```
CStr DSSLoadsS(int32_t Parameter, CStr Argument);
```

This interface returns a string, the variable “parameter” (Integer) is used to specify the property of the class to be used and the variable “argument” (string) can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

Parameter 0: Loads.Name – Read

This parameter allows to read the *Name* property of the active load. The parameter argument can be filled with an empty string.

Parameter 1: Loads.Name – Write

This parameter allows to set the active load by specifying the *Name* load. The parameter argument must contain the Name of the load to activate. The return value will be equal to empty.

Parameter 2: Loads.CVRCurve – Read

This parameter allows to read the *CVRCUrve* property of the active load. The parameter argument can be filled with an empty string.

Parameter 3: Loads.CVRCurve – Write

This parameter allows to set the *CVRCurve* property for the active load. The parameter argument must contain the Name of the new *CVRCurve* to be linked to the active load. The return value will be equal to empty.

Parameter 4: Loads.daily – Read

This parameter allows to read the *daily* property of the active load. The parameter argument can be filled with an empty string.

Parameter 5: Loads.daily – Write

This parameter allows to set the *daily* property for the active load. The parameter argument must contain the Name of the new *daily* to be linked to the active load. The return value will be equal to empty.

Parameter 6: Loads.duty – Read

This parameter allows to read the *duty* property of the active load. The parameter argument can be filled with an empty string.

Parameter 7: Loads.duty – Write

This parameter allows to set the *duty* property for the active load. The parameter argument must contain the Name of the new *duty* to be linked to the active load. The return value will be equal to empty.

Parameter 8: Loads.Spectrum – Read

This parameter allows to read the *Spectrum* property of the active load. The parameter argument can be filled with an empty string.

Parameter 9: Loads.Spectrum – Write

This parameter allows to set the *Spectrum* property for the active load. The parameter argument must contain the Name of the new *Spectrum* to be linked to the active load. The return value will be equal to empty.

Parameter 10: Loads.Yearly – Read

This parameter allows to read the *Yearly* property of the active load. The parameter argument can be filled with an empty string.

Parameter 11: Loads.Yearly – Write

This parameter allows to set the *Yearly* property for the active load. The parameter argument must contain the Name of the new *Yearly* to be linked to the active load. The return value will be equal to empty.

Parameter 12: Loads.Growth – Read

This parameter allows to read the *Growth* property of the active load. The parameter argument can be filled with an empty string.

Parameter 13: Loads.Growth – Write

This parameter allows to set the *Growth* property for the active load. The parameter argument must contain the Name of the new *Growth* to be linked to the active load. The return value will be equal to empty.

Parameter 14: Loads.Sensor

This parameter returns the name of the sensor monitoring the active load.

Loads V (Variant) Interface

This interface can be used to read/modify the properties of the Loads Class where the values are variants (the value can have different formats). The structure of the interface is as follows:

```
void  DSSLoadsV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a string, the variable “parameter” (Integer) is used to specify the property of the class to be used and the variable “argument” (Variant) is used to return the variant structure.

Parameter 0: Loads.AllNames

This parameter allows to read the names of all the loads present in the active circuit. The result is delivered as variant, however, the content of this variant is an array of strings.

Parameter 1: Loads.ZIPV - Read

This parameter allows to read the array of 7 elements (doubles) for ZIP property of the active Load object.

Parameter 2: Loads.ZIPV - Write

This parameter allows to write the array of 7 elements (doubles) for ZIP property of the active Load object.

LoadShapes Interface

This interface implements the LoadShape (ILoadShape) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

LoadShapeI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t LoadShapeI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: LoadShapes.Count

This parameter returns the number of LoadShape objects currently defined in LoadShape collection.

Parameter 1: LoadShapes.First

This parameter sets the first loadshape active and return integer index of the loadshape. Returns 0 if no more.

Parameter 2: LoadShapes.Next

This parameter sets the next loadshape active and return integer index of the loadshape. Returns 0 if no more.

Parameter 3: LoadShapes.Npts read

This parameter gets the number of points in active LoadShape.

Parameter 4: LoadShapes.Npts write

This parameter sets the number of points in active LoadShape.

Parameter 5: LoadShapes.Normalize

This parameter normalizes the P and Q curves based on either Pbase, Qbase or simply the peak value of the curve.

Parameter 6: LoadShapes.UseActual read

This parameter gets a TRUE/FALSE (1/0) to let Loads know to use the actual value in the curve rather than use the value as a multiplier.

Parameter 7: LoadShapes.UseActual write

This parameter sets a TRUE/FALSE (1/0 - Argument) to let Loads know to use the actual value in the curve rather than use the value as a multiplier.

LoadShapeF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double LoadShapeF(int32_t Parameter, double Argument);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: LoadShapes.HrInterval read

This parameter gets the fixed interval time value, hours.

Parameter 1: LoadShapes.HrInterval write

This parameter sets the fixed interval time value, hours.

Parameter 2: LoadShapes.MinInterval read

This parameter gets the fixed interval time value, in minutes.

Parameter 3: LoadShapes.MinInterval write

This parameter sets the fixed interval time value, in minutes.

Parameter 4: LoadShapes.PBase read

This parameter gets the base for normalizing P curve. If left at zero, the peak value is used.

Parameter 5: LoadShapes.PBase write

This parameter sets the base for normalizing P curve. If left at zero, the peak value is used.

Parameter 6: LoadShapes.QBase read

This parameter gets the base for normalizing Q curve. If left at zero, the peak value is used.

Parameter 7: LoadShapes.QBase write

This parameter sets the base for normalizing Q curve. If left at zero, the peak value is used.

Parameter 8: LoadShapes.Sinterval read

This parameter gets the fixed interval data time interval, seconds.

Parameter 9: LoadShapes.Sinterval write

This parameter sets the fixed interval data time interval, seconds.

LoadShapeS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr LoadShapeS(int32_t Parameter, CStr Argument);
```

This interface returns string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: LoadShapes.Name read

This parameter gets the name of the active LoadShape object.

Parameter 1: LoadShapes.Name write

This parameter sets the name of the active LoadShape object.

LoadShapeV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void LoadShapeV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: LoadShapes.AllNames

This parameter gets a variant array of strings containing names of all LoadShape objects currently defined.

Parameter 1: LoadShapes.PMult read

This parameter gets a variant array of doubles for the P multiplier in the LoadShape.

Parameter 2: LoadShapes.PMult write

This parameter sets a variant array of doubles for the P multiplier in the LoadShape.

Parameter 3: LoadShapes.QMult read

This parameter gets a variant array of doubles for the Q multiplier in the LoadShape.

Parameter 4: LoadShapes.QMult write

This parameter sets a variant array of doubles for the Q multiplier in the LoadShape.

Parameter 5: LoadShapes.TimeArray read

This parameter gets a time array in hours corresponding to P and Q multipliers when the Interval = 0.

Parameter 6: LoadShapes.TimeArray write

This parameter sets a time array in hours corresponding to P and Q multipliers when the Interval = 0.

Meters Interface

This interface implements the Meters (IMeters) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

MetersI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t MetersI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Meters.First

This parameter sets the first Energy Meter active. Returns 0 if no monitors.

Parameter 1: Meters.Next

This parameter sets the next energy Meter Active. Returns 0 if no more.

Parameter 2: Meters.Reset

This parameter resets the active Meter object.

Parameter 3: Meters.ResetAll

This parameter resets all Meter object.

Parameter 4: Meters.Sample

This parameter causes active meter to take a sample.

Parameter 5: Meters.Save

This parameter causes active meter to save its current sample buffer to its meter stream. Then you can access the Bytestream or channel data. Most standard solution modes do this automatically.

Parameter 6: Meters.MeteredTerminal read

This parameter returns the number of metered terminal by the active Energy Meter.

Parameter 7: Meters.MeterdTerminal Write

This parameter sets the number of metered terminal by the active Energy Meter.

Parameter 8: Meters.DIFilesAreopen

This parameter returns a global flag (1=true, 0=false) to indicate if Demand Interval (DI) files have been properly opened.

Parameter 9: Meters.SampleAll

This parameter causes all Energy Meters to take a sample of the present state. Returns 0.

Parameter 10: Meters.SaveAll

This parameter save all Energy Meter buffers to their respective file streams. Returns 0.

Parameter 11: Meters.OpenAllDIFiles

This parameter opens Demand Interval (DI) files. Returns 0.

Parameter 12: Meters.CloseAllDIFiles

This parameter closes all Demand Interval (DI) files. Necessary at the end of a run.

Parameter 13: Meters.CountEndElements

This parameter returns the number of zone end elements in the active meter zone.

Parameter 14: Meters.Count

This parameter returns the number of Energy Meters in the Active Circuit.

Parameter 15: Meters.CountBranches

This parameter returns the number of branches in active Energy Meter zone (same as sequencelist size).

Parameter 16: Meters.SequenceIndex read

This parameter returns the index into meter's SequenceList that contains branch pointers in lexical order. Earlier index guaranteed to be up line from later index. Sets PDElement active.

Parameter 17: Meters.SequenceIndex Write

This parameter sets the index into meter's SequenceList that contains branch pointers in lexical order. Earlier index guaranteed to be up line from later index. Sets PDElement active.

Parameter 18: Meters.DoReliabilityCalc

This parameter calculates SAIFI, etc. if the Argument is equal to 1 this parameter will assume restoration, otherwise it will not.

Parameter 19: Meters.SeqListSize

This parameter returns the size of Sequence List.

Parameter 20: Meters.TotalCustomers

This parameter returns the total number of customers in this zone (down line from the Energy Meter).

Parameter 21: Meters.NumSections

This parameter returns the number of feeder sections in this meter's zone.

Parameter 22: Meters.SetActiveSection

This parameter sets the designated section (argument) if the index is valid.

Parameter 23: Meters.OCPDeviceType

This parameter returns the type of OCP device: {1=fuse | 2+ recloser | 3= relay}.

Parameter 24: Meters.NumSectionCustomers

This parameter returns the number of customers in the active section.

Parameter 25: Meters.NumSectionBranches

This parameter returns the number of branches (lines) in the active section.

Parameter 26: Meters.SectSeqIdx

This parameter returns the Sequence Index of the branch at the head of this section.

Parameter 27: Meters.SectTotalCust

This parameter returns the total customers down line from this section.

MetersF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double MetersF(int32_t Parameter, double Argument);

This interface returns a floating point number (64 bits) according to the number sent in the variable "parameter". The parameter can be one of the following:

Parameter 0: Meters.SAIFI

This parameter returns SAIFI for this meter's zone. Execute reliability calc method first.

Parameter 1: Meters.SAIFIkW

This parameter returns the SAIFI based on kW rather than number of customers. Get after reliability calcs.

Parameter 2: Meters.SAIDI

This parameter returns the SAIDI for this meter zone. Execute DoreliabilityCalc first.

Parameter 3: Meters.CustInterrupts

This parameter returns the total customer interruptions for this meter zone based on reliability calcs.

Parameter 4: Meters.AvgRepairTime

This parameter returns the average Repair Time in this Section of the meter zone.

Parameter 5: Meters.FaultRateXRepairHrs

This parameter returns the sum of Fault Rate Time Repair Hours in this section of the meter zone.

Parameter 6: Meters.SumBranchFltRates

This parameter returns the sum of the branch fault rates in this section of the meter's zone.

MetersS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr MetersS(int32_t Parameter, CStr Argument);

This interface returns a string according to the number sent in the variable "parameter". The parameter can be one of the following:

Parameter 0: Meters.Name read

This parameter returns the active Energy Meter's name.

Parameter 1: Meters.Name Write

This parameter sets the active Energy Meter's name.

Parameter 2: Meters.MeteredElement read

This parameter returns the name of the metered element (considering the active Energy Meter).

Parameter 3: Meters.MeteredElement Write

This parameter sets the name of the metered element (considering the active Energy Meter).

MetersV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  MetersV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a variant according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Meters.AllNames

This parameter returns an array of all Energy Meter names.

Parameter 1: Meters.RegisterNames

This parameter returns an array of strings containing the names of the registers.

Parameter 2: Meters.RegisterValues

This parameter returns an array of values contained in the Meter registers for the active Meter.

Parameter 3: Meters.Totals

This parameter returns the totals for all registers of all Meters.

Parameter 4: Meters.PeakCurrent read

This parameter returns an array of doubles with the Peak Current Property.

Parameter 5: Meters.PeakCurrent Write

This parameter receives an array of doubles to set values of Peak Current Property.

Parameter 6: Meters.CalCurrent read

This parameter returns the magnitude of the real part of the Calculated Current (normally determined by solution) for the meter to force some behavior on Load Allocation.

Parameter 7: Meters.CalcCurrent Write

This parameter sets the magnitude of the real part of the Calculated Current (normally determined by solution) for the meter to force some behavior on Load Allocation.

Parameter 8: Meters.AllocFactors read

This parameter returns an array of doubles: allocation factors for the active Meter.

Parameter 9: Meters. AllocFactors Write

This parameter receives an array of doubles to set the phase allocation factors for the active Meter.

Parameter 10: Meters.AllEndElements

This parameter returns a variant array of names of all zone end elements.

Parameter 11: Meters.AllBranchesInZone

This parameter returns a wide string list of all branches in zone of the active Energy Meter object.

Parameter 12: Meters.ZonePCE

This parameter returns a wide string list of all the PCE in zone of the active Energy Meter object.

Monitors Interface

This interface implements the Monitors (IMonitors) interface of OpenDSS by declaring 3 procedures for accessing the different properties included in this interface.

MonitorsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t MonitorsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer according to the number sent in the variable "parameter". The parameter can be one of the following:

Parameter 0: Monitors.First

This parameter sets the first monitor active. Returns 0 if no monitors.

Parameter 1: Monitors.Next

This parameter set the next monitor active. Returns 0 if no more.

Parameter 2: Monitors.Reset

This parameter resets the active Monitor object.

Parameter 3: Monitors.ResetAll

This parameter resets all Monitor object.

Parameter 4: Monitors.Sample

This parameter causes active monitor to take a sample.

Parameter 5: Monitors.Save

This parameter 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.

Parameter 6: Monitors.Show

This parameter converts monitor file into text and displays with text editor.

Parameter 7: Monitors.Mode read

This parameter returns the monitor mode (bitmask integer - see DSS Help).

Parameter 8: Monitors.Mode write

This parameter sets the monitor mode (bitmask integer - see DSS Help).

Parameter 9: Monitors.SampleCount

This parameter returns number of samples in Monitor at present.

Parameter 10: Monitors.SampleAll

This parameter causes all Monitors to take a sample of the present state. Returns 0.

Parameter 11: Monitors.SaveAll

This parameter save all Monitor buffers to their respective file streams. Returns 0.

Parameter 12: Monitors.Count
This parameter returns the number of monitors.
Parameter 13: Monitors.Process
This parameter post-process monitor samples taken so far, e.g., Pst for mode = 4.
Parameter 14: Monitors.ProcessAll
This parameter makes that all monitors post-process the data taken so far.
Parameter 15: Monitors.FileVersion
This parameter returns the Monitor File version (integer).
Parameter 16: Monitors.RecordSize
This parameter returns the size of each record in ByteStream.
Parameter 17: Monitors.NumChannels
This parameter returns the number of Channels on the active Monitor.
Parameter 18: Monitors.Terminal read
This parameter returns the terminal number of element being monitored.
Parameter 19: Monitors.Terminal Write
This parameter sets the terminal number of element being monitored.

MonitorsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr MonitorsS(int32_t Parameter, CStr Argument);

This interface returns a string according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Monitors.FileName
This parameter returns the name of the CSV file associated with active monitor.
Parameter 1: Monitors.Name read
This parameter returns the active Monitor object by name.
Parameter 2: Monitors.Name Write
This parameter sets the active Monitor object by name.
Parameter 3: Monitors.Element read
This parameter returns the full name of element being monitored by the active Monitor.
Parameter 4: Monitors.Element Write
This parameter sets the full name of element being monitored by the active Monitor.

MonitorsV (variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  MonitorsV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a variant according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Monitors.AllNames

This parameter returns an array of all Monitor names (array of strings).

Parameter 1: Monitors.ByteStream

This parameter returns a byte array containing monitor stream values. Make sure a "save" is done first (standard solution modes do this automatically).

Parameter 2: Monitors.Header

This parameter returns the header string; Variant array of strings containing Channel Names.

Parameter 3: Monitors.dblHour

This parameter returns a variant array of doubles containing time value in hours for the time-sampled monitor values; empty if frequency-sampled values for harmonics solution (see dblFreq).

Parameter 4: Monitors.dblFreq

This parameter returns a variant array of doubles containing time values for harmonics mode solutions; empty for time mode solutions (use dblHour).

Parameter 5: Monitors.Channel

This parameter returns a variant array of doubles for the specified channel (usage: MyArray = DSSmonitor.Channel(i)) A save or SaveAll should be executed first. Done automatically by most standard solution modes.

Parallel Interface

These interfaces allows users to use the parallel processing features included in OpenDSS-PM. With this interface it is possible to create multiple actors, specify the CPU where the actor will be executed, control the execution of the actors, check the actors status and progress among many other functionalities.

ParallelI (Integer) Interface

This interface allows to control parameters of the parallel computing suite of OpenDSS-PM where its value can be specified as an integer number. The structure of the interface is as follows:

```
int32_t ParallelI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parallel.NumCPUs

This parameter returns the number of CPUs available in the local computer.

Parameter 1: Parallel.NumCores

This parameter returns the number of physical cores available in the local computer. If your computer has less than 64 Cores, this number should be the number of CPUs/2. For more

information, please check: https://www.howtogeek.com/194756/cpu-basics-multiple-cpus-cores-and-hyper-threading-explained/.

Parameter 2: Parallel.ActiveActor read

This parameter returns the ID of the active actor.

Parameter 3: Parallel.ActiveActor write

This parameter sets the ID of the active actor; this number cannot be higher than the number of existing actors.

Parameter 4: Parallel.CreateActor

This parameter creates a new actor and sets the active actor ID as the ID for the recently created actor. If there are no more CPUs available, the system will not allow the creation of the new actor

Parameter 5: Parallel.ActorCPU read

This parameter gets the ID of the CPU assigned for the execution of the active actor.

Parameter 6: Parallel.ActorCPU write

This parameter sets the CPU for the execution of the active actor.

Parameter 7: Parallel.NumActors

This parameter gets the number of actors created in the actual session.

Parameter 8: Parallel.Wait

This parameter waits until all the actors are free and ready to receive a new command.

Parameter 9: Parallel.ActiveParallel read

This parameter gets if the parallel features of OpenDSS-PM are active. If active, this parameter will return 1, otherwise, will return 0 and OpenDSS-PM will behave sequentially.

Parameter 10: Parallel.ActiveParallel write

This parameter enables/disables the parallel features of OpenDSS-PM. To enable set the argument in 1, otherwise, the argument should be 0 and OpenDSS-PM will behave sequentially.

Parameter 11: Parallel.ConcatenateReports Read

This parameter gets the state of the ConcatenateReports property of OpenDSS-PM. If 1, means that every time the user executes a Show/Export monitor operation, the data stored on the monitors with the same name for each actor will be concatenated one after the other. Otherwise (0), to get access of each monitor the user will have to activate the actor of interest and then perform the Show/Export command on the desired monitor.

Parameter 12: Parallel.ConcatenateReports Write

This parameter sets the state of the ConcatenateReports property of OpenDSS-PM. If 1, means that every time the user executes a Show/Export monitor operation, the data stored on the monitors with the same name for each actor will be concatenated one after the other. Otherwise (0), to get access of each monitor the user will have to activate the actor of interest and then perform the Show/Export command on the desired monitor.

ParallelV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void ParallelV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parallel.ActorProgress

This parameter returns an array of integers containing the progress in percentage for each active actor.

Parameter 1: Parallel.ActorStatus

This parameter returns an array of integers containing the status of each active actor. If 1, the actor is ready to receive new commands, if 0, the actor is busy performing a simulation and cannot take new "solve" commands at this time. However, the actor is capable to deliver values while the simulation is being performed.

Parser Interface

This interface implements the CmathLib (ICmathLib) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

ParserI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t ParserI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parser.IntValue

This parameter return next parameter as a long integer.

Parameter 1: Parser.ResetDelimeters

This parameter reset delimiters to their default values.

Parameter 2: Parser.AutoIncrement read

In this parameter the default is false (0). If true (1) parser automatically advances to next token after DblValue, IntValue, or StrValue. Simpler when you don't need to check for parameter names.

Parameter 3: Parser.AutoIncrement read

In this parameter the default is false (0). If true (1) parser automatically advances to next token after DblValue, IntValue, or StrValue. Simpler when you don't need to check for parameter names.

ParserF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double ParserF(int32_t Parameter, double Argument);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parser.DblValue

This parameter returns next parameter as a double.

ParserS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr ParserS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parser.CmdString read

This parameter gets a string to be parsed. Loading this string resets the parser to the beginning of the line. Then parse off the tokens in sequence.

Parameter 1: Parser.CmdString write

This parameter sets a string to be parsed. Loading this string resets the parser to the beginning of the line. Then parse off the tokens in sequence.

Parameter 2: Parser.NextParam

This parameter gets next token and return tag name (before = sign) if any. See Autoincrement.

Parameter 3: Parser.StrValue

This parameter returns next parameter as a string.

Parameter 4: Parser.WhiteSpace read

This parameter gets the characters used for White space in the command string. Default in blank and Tab.

Parameter 5: Parser.WhiteSpace write

This parameter sets the characters used for White space in the command string. Default in blank and Tab.

Parameter 6: Parser.BeginQuote read

This parameter gets the string containing the characters for quoting in OpenDSS scripts. Matching pairs defined in EndQuote. Default is "([{.

Parameter 7: Parser.BeginQuote write

This parameter sets the string containing the characters for quoting in OpenDSS scripts. Matching pairs defined in EndQuote. Default is "([{.

Parameter 8: Parser.EndQuote read

This parameter gets the string containing the characters, in order, that match the beginning quote characters in BeginQuote. Default is ")]}.

Parameter 9: Parser.EndQuote write

This parameter sets the string containing the characters, in order, that match the beginning quote characters in BeginQuote. Default is ")]}.

Parameter 10: Parser.Delimiters read

This parameter gets the string defining hard delimiters used to separate token on the command string. Default is , and =. The = separates token name from token value. These override whitespaces to separate tokens.

Parameter 11: Parser.Delimiters write

This parameter sets the 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.

ParserV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void ParserV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Parser.Vector

This parameter returns token as variant array of doubles. For parsing quoted array syntax.

Parameter 1: Parser.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.

Parameter 2: Parser.SymMatrix

Use this property to parse a Matrix token in lower triangular form. Symmetry is forced.

PDElements Interface

This interface implements the PDElements (IPDElements) interface of OpenDSS by declaring 3 procedures for accessing the different properties included in this interface.

PDElementsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t PDElementsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PDElements.Count

This parameter gets number of PDElements in active circuit.

Parameter 1: PDElements.First

This parameter sets the first enabled PD element to be the active element. Returns 0 if none found.

Parameter 2: PDElements.Next

This parameter sets the next enabled PD element to be the active element. Returns 0 if none found.

Parameter 3: PDElements.IsShunt

This parameter returns 1 if the PD element should be treated as a shunt element rather than a series element. Applies to capacitor and reactor elements in particular.

Parameter 4: PDElements.NumCustomers

This parameter gets the number of customers in this branch.

Parameter 5: PDElements.TotalCustomers

This parameter gets the total number of customers from this branch to the end of the zone.

Parameter 6: PDElements.ParentPDElement

This parameter gets the parent PD element to be the active circuit element. Returns 0 if no more elements upline.

Parameter 7: PDElements.FromTerminal

This parameter gets the number of the terminal of active PD element that is on the "from" side. This is set after the meter zone is determined.

Parameter 8: PDElements.SectionID

This parameter gets the integer ID of the feeder section that this PDElement branch is part of.

PDElementsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double PDElementsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PDElements.FaultRate read

This parameter gets the number of failures per year. For LINE elements: Number of failures per unit length per year.

Parameter 1: PDElements.FaultRate write

This parameter sets the number of failures per year. For LINE elements: Number of failures per unit length per year.

Parameter 2: PDElements.PctPermanent read

This parameter gets the percent of faults that are permanent (require repair). Otherwise, fault is assumed to be transient/temporary.

Parameter 3: PDElements.PctPermanent write

This parameter sets the percent of faults that are permanent (require repair). Otherwise, fault is assumed to be transient/temporary.

Parameter 4: PDElements.Lambda

This parameter gets the failure rate for this branch. Faults per year including length of line.

Parameter 5: PDElements.AccumulatedL

This parameter gets the accumulated failure rate for this branch on down line.

Parameter 6: PDElements.RepairTime

This parameter gets the average time to repair a permanent fault on this branch, hours.

Parameter 7: PDElements.TotalMiles

This parameter gets the total miles of line from this element to the end of the zone. For recloser siting algorithm.

PDElementsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr PDElementsF(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PDElements.Name read

This parameter gets the name of the active PDElement, returns null string if active element id not PDElement.

Parameter 1: PDElements.Name write

This parameter sets the name of the active PDElement, returns null string if active element id not PDElement.

PVsystems Interface

This interface implements the PVSystems (IPVSystems) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

PVSystemsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t PVSystemsI(int32_t Parameter, int32_t Argument);

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PVSystems.Count

This parameter returns the number of PVSystem objects currently defined in the active circuit.

Parameter 1: PVSystems.First

This parameter sets the first PVSystem to be active; returns 0 if none.

Parameter 2: PVSystems.Next

This parameter sets the next PVSystem to be active; returns 0 if none.

Parameter 3: PVSystems.Idx read

This parameter gets the active PVSystem by index; 1..Count.

Parameter 4: PVSystems.Idx write

This parameter sets the active PVSystem by index; 1..Count.

PVSystemsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double PVSystemsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PVSystems.Irradiance read

This parameter gets the present value of the Irradiance property in W/sq-m.

Parameter 1: PVSystems.Irradiance write

This parameter sets the present value of the Irradiance property in W/sq-m.

Parameter 2: PVSystems.kW

This parameter gets the kW output.

Parameter 3: PVSystems.kvar read

This parameter gets the kvar value.

Parameter 4: PVSystems.kvar write

This parameter sets the kvar value.

Parameter 5: PVSystems.pf read

This parameter gets the power factor value.

Parameter 6: PVSystems.pf write

This parameter sets the power factor value.

Parameter 7: PVSystems.KVARated read

This parameter gets the rated kVA of the PVSystem.

Parameter 8: PVSystems.KVARated write

This parameter sets the rated kVA of the PVSystem.

Parameter 9: PVSystems.pmpp read

This parameter gets the rated max power of the PV array for 1.0 kW/sq-m irradiance and a user-selected array temperature of the active PVSystem.

Parameter 10: PVSystems.pmpp write

This parameter sets the rated max power of the PV array for 1.0 kW/sq-m irradiance and a user-selected array temperature of the active PVSystem.

Parameter 11: PVSystems.IrradianceNow

This parameter returns the current irradiance value of the active PVSystem. The current irradiance value is the one provided by the irradiance shape linked to the PV, use it to get this information while running simulations.

PVSystemsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr PVSystemsS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PVSystems.Name read
This parameter gets the name of the active PVSystem.
Parameter 1: PVSystems.Name write
This parameter sets the name of the active PVSystem.
Parameter 2: PVSystems.Sensor
This parameter returns the name of the sensor monitoring the active PV System.

PVSystemsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void PVSystemsV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: PVSystems.AllNames
This parameter gets the variant array of string containing names of all PVSystems in the circuit.

Reclosers Interface

This interface implements the Reclosers (IReclosers) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

RecloserSI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t RecloserSI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Reclosers.Count

This parameter gets number of Reclosers in active circuit.

Parameter 1: Reclosers.First

This parameter sets first recloser to be active Circuit Element. Returns 0 if none.

Parameter 2: Reclosers.Next

This parameter sets next recloser to be active Circuit Element. Returns 0 if none.

Parameter 3: Reclosers.MonitoredTerm read

This parameter gets the terminal number of Monitored Object for the Recloser.

Parameter 4: Reclosers.MonitoredTerm write

This parameter sets the terminal number of Monitored Object for the Recloser.

Parameter 5: Reclosers.SwitchedTerm read

This parameter gets the terminal of the controlled device being switched by the Recloser.

Parameter 6: Reclosers.SwitchedTerm write

This parameter sets the terminal of the controlled device being switched by the Recloser.

Parameter 7: Reclosers.NumFast read

This parameter gets the number of fast shots.

Parameter 8: Reclosers.NumFast write

This parameter sets the number of fast shots.

Parameter 9: Reclosers.Shots read

This parameter gets the number of shots to lockout (fast + delayed).

Parameter 10: Reclosers.Shots write

This parameter sets the number of shots to lockout (fast + delayed).

Parameter 11: Reclosers.Open

This parameter opens the recloser's controlled element and lock out the recloser.

Parameter 12: Reclosers.Close

This parameter closes the switched object controlled by the recloser. Resets recloser to first operation.

Parameter 13: Reclosers.Idx read

This parameter gets the active recloser by index into the recloser list. 1..Count.

Parameter 14: Reclosers.Idx write

This parameter sets the active recloser by index into the recloser list. 1..Count.

Parameter 15: Reclosers.Reset

This parameter resets the recloser to its normal state. If open, lock out the recloser. If closed, resets recloser to first operation.

RecloserSF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double RecloserSF(int32_t Parameter, double Argument);
```

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Reclosers.PhaseTrip read
This parameter gets the phase trip curve multiplier or actual amps.
Parameter 1: Reclosers.PhaseTrip write
This parameter sets the phase trip curve multiplier or actual amps.
Parameter 2: Reclosers.PhaseInst read
This parameter gets the phase instantaneous curve multiplier or actual amps.
Parameter 3: Reclosers.PhaseInst write
This parameter sets the phase instantaneous curve multiplier or actual amps.
Parameter 4: Reclosers.GroundTrip read
This parameter gets the ground (3I0) trip multiplier or actual amps.
Parameter 5: Reclosers.GroundTrip write
This parameter sets the ground (3I0) trip multiplier or actual amps.
Parameter 6: Reclosers.GroundInst read
This parameter gets the ground (3I0) instantaneous trip setting - curve multiplier or actual amps.
Parameter 7: Reclosers.GroundInst write
This parameter sets the ground (3I0) instantaneous trip setting - curve multiplier or actual amps.

RecloserssS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr RecloserssS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Reclosers.Name read
This parameter gets the name of the active Recloser Object.
Parameter 1: Reclosers.Name write
This parameter sets the name of the active Recloser Object.
Parameter 2: Reclosers.MonitoredObj read
This parameter gets the full name of object this Recloser is monitoring.
Parameter 3: Reclosers.MonitoredObj write
This parameter sets the full name of object this Recloser is monitoring.
Parameter 4: Reclosers.SwitchedObj read
This parameter gets the full name of the circuit element that is being switched by this Recloser.
Parameter 5: Reclosers.SwitchedObj write
This parameter sets the full name of the circuit element that is being switched by this Recloser.
Parameter 6: Reclosers.State read
This property gets the present state of recloser.

Parameter 7: Reclosers.State write

This property sets the present state of recloser. If set to open, open recloser's controlled element and lock out the recloser. If set to close, close recloser's controlled and resets recloser to first operation.

Parameter 8: Reclosers.NormalState read

This property gets the normal state (the state for which the active recloser will be forced into at the beginning of the simulation) for the active recloser.

Parameter 9: Reclosers.NormalState write

This property sets the normal state (the state for which the active recloser will be forced into at the beginning of the simulation) for the active recloser.

RecloserSV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void RecloserSV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Reclosers.AllNames

This parameter gets a variant array of strings with names of all reclosers in active circuit.

Parameter 1: Reclosers.RecloseIntervals

This parameter gets a variant array of doubles: reclose intervals (s) between shots.

ReduceCkt Interface

This interface implements the ReduceCkt (IReduceCkt) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

ReduceCktI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t ReduceCktI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: ReduceCkt.Do1phLaterals

This method removes all 1-phase laterals in the active EnergyMeter's zone. Loads and other shunt elements are moved to the parent 3-phase bus.

Parameter 1: ReduceCkt.DoBranchRemove

This method removes (disable) all branches down-line from the active PDelement. Circuit must have an Energymeter on this branch. If KeepLoad=Y (default) a new Load element is defined and kW, kvar

set to present power flow solution for the first element eliminated. The EditString is applied to each new Load element defined.

Parameter 2: ReduceCkt.DoDangling

This method reduces Dangling Algorithm; branches with nothing connected.

Parameter 3: ReduceCkt.DoDefault

This method Executes the default circuit reduction, which eliminates all dangling end buses and buses without load.

Parameter 4: ReduceCkt.DoLoopBreak

This method breaks (disables) all the loops found in the active circuit. Disables one of the Line objects at the head of a loop to force the circuit to be radial.

Parameter 5: ReduceCkt.DoParallelLines

This method merges all parallel lines found in the circuit to facilitate its reduction.

Parameter 6: ReduceCkt.DoShortLines

This method executes the Do ShortLines algorithm: Set Zmag first if you don't want the default.

Parameter 7: ReduceCkt.DoSwitches

This method merges Line objects in which the IsSwitch property is true with the down-line Line object.

Parameter 8: ReduceCkt.KeepLoad - Read

This property gets a flag indicating to keep (1) the load for Reduction options that remove branches.

Parameter 9: ReduceCkt.KeepLoad - Write

This property sets the flag for indicating to keep (1) the load for Reduction options that remove branches. Set 1 in the argument if want to keep the load for the reduction, otherwise, set the argument 0.

ReduceCktF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double ReduceCktF(int32_t Parameter, double Argument);
```

This interface returns a floating-point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: ReduceCkt.Zmag - Read

This property gets the value of impedance value associated with the DoShortLines method. Lines with less impedance will be merged out of the circuit, eliminating one or more buses.

Parameter 1: ReduceCkt.Zmag - Write

This property sets the value of impedance value associated with the DoShortLines method. Lines with less impedance will be merged out of the circuit, eliminating one or more buses.

ReduceCktS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr ReduceCktS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: ReduceCkt.EditString - Read

This property gets the string containing the OpenDSS command appended to the BranchRemove option when a new load is defined to represent the load on the removed branch.

Parameter 1: ReduceCkt.EditString - Write

This property sets the string containing the OpenDSS command appended to the BranchRemove option when a new load is defined to represent the load on the removed branch. The EditString must be specified in the argument.

Parameter 2: ReduceCkt.EnergyMeter - Read

This property gets the name of EnergyMeter object for circuit reduction.

Parameter 3: ReduceCkt.EnergyMeter - Write

This property sets the name of EnergyMeter object for circuit reduction. The name of the EnergyMeter must be specified in the argument.

Parameter 4: ReduceCkt.SaveCircuit

This method executes the Save Circuit command from the COM interface. Saves the reduced circuit in a directory with the specified name. The full path name of the Master.DSS file is return in the DSSText.Result property. The name of the circuit must be specified in the argument.

Parameter 5: ReduceCkt.StartPDElement - Read

This property gets the full name of the first PDelement for the BranchRemove command (class.name -> e.g. Line.myLine).

Parameter 6: ReduceCkt.StartPDElement - Write

This property sets the full name of the first PDelement for the BranchRemove command (class.name -> e.g. Line.myLine). The PDElement name must be specified in the argument.

RegControls Interface

This interface implements the RegControls (IRegControls) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

RegControlsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t RegControlsI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: RegControls.First

This parameter sets the first RegControl active. Returns 0 if no more.

Parameter 1: RegControls.Next

This parameter sets the next RegControl active. Returns 0 if no more

Parameter 2: RegControls.TapWinding read

This parameter gets the tapped winding number.

Parameter 3: RegControls.TapWinding write

This parameter sets the tapped winding number.

Parameter 4: RegControls.Winding read

This parameter gets the winding number for PT and CT connections.

Parameter 5: RegControls.Winding write

This parameter sets the winding number for PT and CT connections.

Parameter 6: RegControls.IsReversible read

This parameter gets the setting in the reverse direction, usually not applicable to substation transformers.

Parameter 7: RegControls.IsReversible read

This parameter sets the different settings for the reverse direction (see Manual for details), usually not applicable to substation transformers.

Parameter 8: RegControls.IsInverseTime read

This parameter gets the inverse time feature. Time delay is inversely adjusted, proportional to the amount of voltage outside the regulator band.

Parameter 9: RegControls.IsInverseTime write

This parameter sets the inverse time feature. Time delay is inversely adjusted, proportional to the amount of voltage outside the regulator band.

Parameter 10: RegControls.MaxTapChange read

This parameter gets the maximum tap change per iteration in STATIC solution mode. 1 is more realistic, 16 is the default for faster solution.

Parameter 11: RegControls.MaxTapChange write

This parameter sets the maximum tap change per iteration in STATIC solution mode. 1 is more realistic, 16 is the default for faster solution.

Parameter 12: RegControls.Count

This parameter gets the number of RegControl objects in Active Circuit.

Parameter 13: RegControls.TapNumber read

This parameter gets the actual tap number of the active RegControl.

Parameter 14: RegControls.TapNumber write

This parameter sets the actual tap number of the active RegControl.

RegControlsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double RegControlsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: RegControls.CTPrimary read
This parameter gets the CT primary ampere rating (secondary is 0.2 amperes).
Parameter 1: RegControls.CTPrimary write
This parameter sets the CT primary ampere rating (secondary is 0.2 amperes).
Parameter 2: RegControls.PTRatio read
This parameter gets the PT ratio for voltage control settings.
Parameter 3: RegControls.PTRatio write
This parameter sets the PT ratio for voltage control settings.
Parameter 4: RegControls.ForwardR read
This parameter gets the LDC R settings in Volts.
Parameter 5: RegControls.ForwardR write
This parameter sets the LDC R settings in Volts.
Parameter 6: RegControls.ForwardX read
This parameter gets the LDC X settings in Volts.
Parameter 7: RegControls.ForwardX write
This parameter sets the LDC X settings in Volts.
Parameter 8: RegControls.ReverseR read
This parameter gets the reverse LDC R settings in Volts.
Parameter 9: RegControls.ReverseR write
This parameter sets the reverse LDC R settings in Volts.
Parameter 10: RegControls.ReverseX read
This parameter gets the reverse LDC X settings in Volts.
Parameter 11: RegControls.ReverseX write
This parameter sets the reverse LDC X settings in Volts.
Parameter 12: RegControls.Delay read
This parameter gets the time delay [s] after arming before the first tap change. Control may reset before actually changing taps.
Parameter 13: RegControls.Delay write
This parameter sets the time delay [s] after arming before the first tap change. Control may reset before actually changing taps.
Parameter 14: RegControls.TapDelay read
This parameter gets the time delay [s] for subsequent tap changes in a set. Control may reset before actually changing taps.
Parameter 15: RegControls.TapDelay write
This parameter sets the time delay [s] for subsequent tap changes in a set. Control may reset before actually changing taps.

Parameter 16: RegControls.VoltageLimit read

This parameter gets the first house voltage limit on PT secondary base. Setting to 0 disables this function.

Parameter 17: RegControls.VoltageLimit write

This parameter sets the first house voltage limit on PT secondary base. Setting to 0 disables this function.

Parameter 18: RegControls.ForwardBand read

This parameter gets the regulation bandwidth in forward direction, centered on Vreg.

Parameter 19: RegControls.ForwardBand write

This parameter sets the regulation bandwidth in forward direction, centered on Vreg.

Parameter 20: RegControls.ForwardVreg read

This parameter gets the target voltage in the forward direction, on PT secondary base.

Parameter 21: RegControls.ForwardVreg write

This parameter sets the target voltage in the forward direction, on PT secondary base.

Parameter 22: RegControls.ReverseBand read

This parameter gets the bandwidth in reverse direction, centered on reverse Vreg.

Parameter 23: RegControls.ReverseBand write

This parameter sets the bandwidth in reverse direction, centered on reverse Vreg.

Parameter 24: RegControls.ReverseVreg read

This parameter gets the target voltage in the reverse direction, on PT secondary base.

Parameter 25: RegControls.ReverseVreg write

This parameter sets the target voltage in the reverse direction, on PT secondary base.

RegControlsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr RegControlsS(int32_t Parameter, CStr Argument);
```

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: RegControls.Name read

This parameter gets the active RegControl name.

Parameter 1: RegControls.Name write

This parameter sets the active RegControl name.

Parameter 2: RegControls.MonitoredBus read

This parameter gets the name of the remote regulated bus, in lieu of LDC settings.

Parameter 3: RegControls. MonitoredBus write

This parameter sets the name of the remote regulated bus, in lieu of LDC settings.

Parameter 4: RegControls.Transformer read

This parameter gets the name of the transformer this regulator controls.

Parameter 5: RegControls.Transformer write
This parameter sets the name of the transformer this regulator controls.

RegControlsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void RegControlsV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: RegControls.AllNames
This parameter gets a variant array of strings containing all RegControl names.

Relays Interface

This interface implements the Relays (IRelays) interface of OpenDSS by declaring 3 procedures for accessing the different properties included in this interface.

RelaysI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t RelaysI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Relays.Count
This parameter gets number of Relays in active circuit.
Parameter 1: Relays.First
This parameter sets first relay active. If none, returns 0.
Parameter 2: Relays.Next
This parameter sets next relay active. If none, returns 0.
Parameter 3: Relays.MonitoredTerm read
This parameter gets the number of terminal of monitored element that this relay is monitoring.
Parameter 4: Relays.MonitoredTerm write
This parameter sets the number of terminal of monitored element that this relay is monitoring.
Parameter 5: Relays.SwitchedTerm read
This parameter gets the number of terminal of the switched object that will be opened when the relay trips.

Parameter 6: Relays.SwitchedTerm write

This parameter sets the number of terminal of the switched object that will be opened when the relay trips.

Parameter 7: Relays.Idx read

This parameter gets the active relay by index into the Relay list. 1..Count.

Parameter 8: Relays.Idx write

This parameter sets the active relay by index into the Relay list. 1..Count.

Parameter 9: Relays.Open

This parameter opens recloser's controlled element and lock out the relay.

Parameter 10: Relays.Close

This parameter closes the switched object controlled by the relay. Resets relay to first operation.

Parameter 11: Relays.Reset

This parameter resets the relay to its normal state. If open, lock out the relay. If closed, resets relay to first operation.

RelaysS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr RelaysS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Relays.Name read

This parameter gets the name of the active Relay.

Parameter 1: Relays.Name write

This parameter sets the name of the active Relay.

Parameter 2: Relays.MonitoredObj read

This parameter gets the full name of the object this relay is monitoring.

Parameter 3: Relays.MonitoredObj write

This parameter sets the full name of the object this relay is monitoring.

Parameter 4: Relays.SwitchedObj read

This parameter gets the full name of element that will switched when relay trips.

Parameter 5: Relays.SwitchedObj write

This parameter sets the full name of element that will switched when relay trips.

Parameter 6: Relays.State read

This property gets the present state of relay.

Parameter 7: Relays.State write

This property sets the present state of relay. If set to open, open relay's controlled element and lock out the relay. If set to close, close relay's controlled element and resets relay to first operation.

Parameter 8: Relays.NormalState read

This property gets the normal state (the state for which the active relay will be forced into at the beginning of the simulation) for the active relay.

Parameter 9: Relays.NormalState write

This property sets the normal state (the state for which the active relay will be forced into at the beginning of the simulation) for the active relay.

RelaysV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void RelaysV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Relays.AllNames

This parameter gets a variant array of strings containing names of all relay elements.

Sensors Interface

This interface implements the Sensors (ISensors) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

SensorsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t SensorsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Sensors.Count

This parameter gets number of sensors in active circuit.

Parameter 1: Sensors.First

This parameter sets the first sensor active. Returns 0 if none.

Parameter 2: Sensors.Next

This parameter sets the next sensor active. Returns 0 if none

Parameter 3: Sensors.IsDelta read

This parameter returns 1 if the sensor is connected in delta; otherwise, returns 0.

Parameter 4: Sensors.IsDelta write

This parameter allows to set 1 if the sensor is connected in delta; otherwise, set 0 (argument).

Parameter 5: Sensors.ReverseDelta read

This parameter returns 1 if voltage measurements are 1-3, 3-2, 2-1; otherwise 0.

Parameter 6: Sensors.ReverseDelta write

This parameter allows to set 1 if voltage measurements are 1-3, 3-2, 2-1; otherwise 0.

Parameter 7: Sensors.MeteredTerminal read

This parameter gets the number of the measured terminal in the measured element.

Parameter 8: Sensors.MeteredTerminal write

This parameter sets the number of the measured terminal in the measured element.

Parameter 9: Sensors.Reset

This parameter clears the active sensor.

Parameter 10: Sensors.ResetAll

This parameter clears all sensors in the active circuit.

SensorsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double SensorsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Sensors.PctError read

This parameter gets the assumed percent error in the Sensor measurement. Default is 1.

Parameter 1: Sensors.PctError write

This parameter sets the assumed percent error in the Sensor measurement. Default is 1.

Parameter 2: Sensors.Weight read

This parameter gets the weighting factor for this sensor measurement with respect to the other sensors. Default is 1.

Parameter 3: Sensors.Weight write

This parameter sets the weighting factor for this sensor measurement with respect to the other sensors. Default is 1.

Parameter 4: Sensors.kVbase read

This parameter gets the voltage base for the sensor measurements. LL for 2 and 3 - phase sensors, LN for 1-phase sensors.

Parameter 5: Sensors.kVbase write

This parameter sets the voltage base for the sensor measurements. LL for 2 and 3 - phase sensors, LN for 1-phase sensors.

SensorsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr SensorsS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Sensors.Name read

This parameter gets the name of the active sensor object.

Parameter 1: Sensors.Name write

This parameter sets the name of the active sensor object.

Parameter 2: Sensors.MeteredElement read

This parameter gets the full name of the measured element.

Parameter 3: Sensors.MeteredElement write

This parameter sets the full name of the measured element.

SensorsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void SensorsV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Sensors.AllNames

This parameter returns a variant array of sensor names.

Parameter 1: Sensors.Currents read

This parameter gets an array of doubles for the line current measurements; don't use with KWS and KVARS.

Parameter 2: Sensors.Currents write

This parameter sets an array of doubles for the line current measurements; don't use with KWS and KVARS.

Parameter 3: Sensors.KVARS read

This parameter gets an array of doubles for Q measurements; overwrites currents with a new estimate using KWS.

Parameter 4: Sensors.KVARS write

This parameter sets an array of doubles for Q measurements; overwrites currents with a new estimate using KWS.

Parameter 5: Sensors.KWS read

This parameter gets an array of doubles for P measurements; overwrites currents with a new estimate using KVARS.

Parameter 6: Sensors.KWS write

This parameter sets an array of doubles for P measurements; overwrites currents with a new estimate using KVARS.

Parameter 7: Sensors.AllocationFactor

This parameter returns an array of doubles representing the allocation factors per phase for the active sensor.

Settings Interface

This interface implements the Settings (ISettings) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

SettingsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t SettingsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Settings.AllowDuplicates read

This parameter gets if OpenDSS allows duplicate names of objects: {1 allow, 0 not allow}.

Parameter 1: Settings.AllowDuplicates Write

This parameter sets if OpenDSS allows duplicate names of objects: {1 allow, 0 not allow}.

Parameter 2: Settings.ZoneLock read

This parameter gets the status of Lock zones on energy meters to prevent rebuilding if a circuit change occurs: {1= true, 0= False}.

Parameter 3: Settings.ZoneLock Write

This parameter sets the status of Lock zones on energy meters to prevent rebuilding if a circuit change occurs: {1= true, 0= False}.

Parameter 4: Settings.CktModel read

This parameter gets {dssMultiphase* | dssPositiveSeq} Indicate if the circuit model is positive sequence.

Parameter 5: Settings.CktModel Write

This parameter sets {dssMultiphase* | dssPositiveSeq} Indicate if the circuit model is positive sequence.

Parameter 6: Settings.Trapezoidal read

This parameter gets {True (1) | False (0)} value of trapezoidal integration flag in Energy Meters.

Parameter 7: Settings.Trapezoidal Write

This parameter sets {True (1) | False (0)} value of trapezoidal integration flag in Energy Meters.

SettingsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double SettingsF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Settings.AllocationFactors
This parameter sets all load allocation factors for all loads defined by XFKVA property to this value.
Parameter 1: Settings.NormVminpu read
This parameter gets the per unit minimum voltage for Normal conditions.
Parameter 2: Settings.NormVminpu write
This parameter sets the per unit minimum voltage for Normal conditions.
Parameter 3: Settings.NormVmaxpu read
This parameter gets the per unit maximum voltage for Normal conditions.
Parameter 4: Settings.NormVmaxpu write
This parameter sets the per unit maximum voltage for Normal conditions.
Parameter 5: Settings.EmergVminpu read
This parameter gets the per unit minimum voltage for Emergency conditions.
Parameter 6: Settings.EmergVminpu write
This parameter set the per unit minimum voltage for Emergency conditions.
Parameter 7: Settings.EmergVmaxpu read
This parameter gets the per unit maximum voltage for Emergency conditions.
Parameter 8: Settings.EmergVmaxpu write
This parameter sets the per unit maximum voltage for Emergency conditions.
Parameter 9: Settings.UEWeight read
This parameter gets the weighting factor applied to UE register values.
Parameter 10: Settings.UEWeight write
This parameter sets the weighting factor applied to UE register values.
Parameter 11: Settings.LossWeight read
This parameter gets the weighting factor applied to Loss register values.
Parameter 12: Settings.LossWeight write
This parameter sets the weighting factor applied to Loss register values.
Parameter 13: Settings.PriceSignal read
This parameter gets the price signal for the circuit.
Parameter 14: Settings.PriceSignal write
This parameter sets the price signal for the circuit.

SettingsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr SettingsS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Settings.AutoBusList read

This parameter gets the list of Buses or (File=xxxxx) syntax for the AutoAdd solution mode.

Parameter 1: Settings.AutoBusList write

This parameter sets the list of Buses or (File=xxxxx) syntax for the AutoAdd solution mode.

Parameter 2: Settings.PriceCurve read

This parameter gets the name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.

Parameter 3: Settings.PriceCurve write

This parameter sets the name of LoadShape object that serves as the source of price signal data for yearly simulations, etc.

SettingsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  SettingsV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Settings.UERegs read

This parameter gets the array of Integers defining Energy Meter registers to use for computing UE.

Parameter 1: Settings.UERegs write

This parameter sets the array of Integers defining Energy Meter registers to use for computing UE.

Parameter 2: Settings.LossRegs read

This parameter gets the array of Integers defining Energy Meter registers to use for computing Losses.

Parameter 3: Settings.LossRegs write

This parameter sets the array of Integers defining Energy Meter registers to use for computing Losses.

Parameter 4: Settings.VoltageBases read

This parameter gets the array of doubles defining the legal voltage bases in kV L-L.

Parameter 5: Settings.VoltageBases write

This parameter sets the array of doubles defining the legal voltage bases in kV L-L.

Solution Interface

This interface implements the Solution (ISolution) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

SolutionI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t SolutionI(int32_t Parameter, int32_t Argument);

This interface returns an integer according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Solution.Solve

Executes the solution for the present solution mode. Returns 0.

Parameter 1: Solution.Mode - Read

This parameter returns the present solution mode (See DSS help).

Parameter 2: Solution.Mode - Write

This parameter modifies the present solution mode (See DSS help).

Parameter 3: Solution.hour - Read

This parameter returns the present hour (See DSS help).

Parameter 4: Solution.hour - Write

This parameter modifies the present hour (See DSS help).

Parameter 5: Solution.Year - Read

This parameter returns the present Year (See DSS help).

Parameter 6: Solution.Year - Write

This parameter modifies the present Year (See DSS help).

Parameter 7: Solution.Iterations

This parameter returns the number of iterations taken for the last solution.

Parameter 8: Solution.MaxIterations - Read

This parameter returns the Maximum number of iterations used to solve the circuit.

Parameter 9: Solution.MaxIterations - Write

This parameter modifies the Maximum number of iterations used to solve the circuit.

Parameter 10: Solution.Number - Read

This parameter returns the number of solutions to perform for MonteCarlo and time series simulations.

Parameter 11: Solution.Number - Write

This parameter modifies the number of solutions to perform for MonteCarlo and time series simulations.

Parameter 12: Solution.Random - Read

This parameter returns the randomization mode for random variables "Gaussian" o "Uniform".

Parameter 13: Solution.Random - Write

This parameter modifies the randomization mode for random variables "Gaussian" o "Uniform".

Parameter 14: Solution.LoadModel - Read

This parameter returns the Load Model: {dssPowerFlow (default)|dssAdmittance}.

Parameter 15: Solution.LoadModel - Write

This parameter modifies the Load Model: {dssPowerFlow (default)|dssAdmittance}.

Parameter 16: Solution.AddType - Read

This parameter returns the type of device to add in AutoAdd Mode: {dssGen (default)|dssCap}.

Parameter 17: Solution.AddType - Write

This parameter modifies the type of device to add in AutoAdd Mode: {dssGen (default)|dssCap}.

Parameter 18: Solution.Algorithm - Read

This parameter returns the base solution algorithm: {dssNormalSolve | dssNewtonSolve}.

Parameter 19: Solution.Algorithm - Write

This parameter modifies the base solution algorithm: {dssNormalSolve | dssNewtonSolve}.

Parameter 20: Solution.ControlMode - Read

This parameter returns the mode for control devices: {dssStatic (default) | dssEvent | dssTime}.

Parameter 21: Solution.ControlMode - Write

This parameter modifies the mode for control devices: {dssStatic (default) | dssEvent | dssTime}.

Parameter 22: Solution.ControlIterations - Read

This parameter returns the current value of the control iteration counter.

Parameter 23: Solution.ControlIterations - Write

This parameter modifies the current value of the control iteration counter.

Parameter 24: Solution.MaxControlIterations - Read

This parameter returns the maximum allowable control iterations.

Parameter 25: Solution.MaxControlIterations - Write

This parameter modifies the maximum allowable control iterations.

Parameter 26: Solution.Sample_DoControlActions

This parameter sample controls and then process the control queue for present control mode and dispatch control actions. Returns 0.

Parameter 27: Solution.CheckFaultStatus

This parameter executes status check on all fault objects defined in the circuit. Returns 0.

Parameter 28: Solution.SolveDirect

This parameter executes a direct solution from the system Y matrix, ignoring compensation currents of loads, generators (includes Yprim only).

Parameter 29: Solution.SolvePflow

This parameter solves using present power flow method. Iterative solution rather than direct solution.

Parameter 30: Solution.SolveNoControl

This parameter is similar to SolveSnap except no control actions are checked or executed.

Parameter 31: Solution.SolvePlusControl

This parameter executes a power flow solution (SolveNoControl) plus executes a CheckControlActions that executes any pending control actions.

Parameter 32: Solution.InitSnap

This parameter initializes some variables for snap shot power flow. SolveSnap does this automatically.

Parameter 33: Solution.CheckControls

This parameter performs the normal process for sampling and executing Control Actions and Fault Status and rebuilds Y if necessary.

Parameter 34: Solution.SampleControlDevices

This parameter executes a sampling of all intrinsic control devices, which push control actions into the control queue.

Parameter 35: Solution.DoControlActions

This parameter pops control actions off the control queue and dispatches to the proper control element.

Parameter 36: Solution.BuildYMatrix

This parameter forces building of the System Y matrix according to the argument: {1= series elements only | 2= Whole Y matrix}.

Parameter 37: Solution.SystemYChanged

This parameter indicates if elements of the System Y have been changed by recent activity. If changed returns 1; otherwise 0.

Parameter 38: Solution.Converged - Read

This parameter indicates whether the circuit solution converged (1 converged | 0 not converged).

Parameter 39: Solution.Converged - Write

This parameter modifies the converged flag (1 converged | 0 not converged).

Parameter 40: Solution.TotalIterations

This parameter returns the total iterations including control iterations for most recent solution.

Parameter 41: Solution.MostIterationsDone

This parameter returns the max number of iterations required to converge at any control iteration of the most recent solution.

Parameter 42: Solution.ControlActionsDone - Read

This parameter indicates that the control actions are done: {1 done, 0 not done}.

Parameter 43: Solution.ControlActionsDone - Write

This parameter modifies the flag to indicate that the control actions are done: {1 done, 0 not done}.

Parameter 44: Solution.FinishTimeStep

This parameter calls cleanup, sample monitors, and increment time at end of time step.

Parameter 45: Solution.Cleanup

This parameter update storage, invcontrol, etc., at end of time step.

Parameter 46: Solution.SolveAll

This parameter starts the solution process for all the actors created in memory. Please be sure that the circuits of each actor have been compiled and ready to be solved before using this command.

Parameter 47: Solution.CalcIncMatrix

This parameter starts the calculation of the incidence matrix for the active actor. Please be sure that the circuits of each actor have been compiled and ready to be solved before using this command.

Parameter 48: Solution.CalcIncMatrix_O

This parameter starts the calculation of the Branch to Node incidence matrix for the active actor. Please be sure that the circuits of each actor have been compiled and ready to be solved before using this command. The difference between this command and the *CalcIncMatrix* is that the calculated matrix will be ordered hierarchically from the substation to the feeder end, which can be helpful for many operations. Additionally, the Bus Levels vector is calculated and the rows (PDElements) and columns (Buses) are permuted so it is easy to identify their position in the circuit.

SolutionF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double SolutionF(int32_t Parameter, double Argument);

This interface returns a floating point number according to the number sent in the variable "parameter". The parameter can be one of the following:

Parameter 0: Solution.Frequency read
This parameter returns the frequency for the next solution.
Parameter 1: Solution.Frequency Write
This parameter sets the frequency for the next solution.
Parameter 2: Solution.Seconds read
This parameter returns the seconds from top of the hour.
Parameter 3: Solution.Seconds Write
This parameter sets the seconds from top of the hour.
Parameter 4: Solution.StepSize read
This parameter returns the step size for the next solution.
Parameter 5: Solution.StepSize Write
This parameter sets the step size for the next solution.
Parameter 6: Solution.LoadMult read
This parameter returns the default load multiplier applied to all non-fixed loads.
Parameter 7: Solution.LoadMult Write
This parameter sets the default load multiplier applied to all non-fixed loads.
Parameter 8: Solution.Tolerance read
This parameter returns the solution convergence tolerance.
Parameter 9: Solution.Tolerance Write
This parameter sets the solution convergence tolerance.
Parameter 10: Solution.pctgrowth read
This parameter returns the percent default annual load growth rate.
Parameter 11: Solution.pctgrowth Write
This parameter sets the percent default annual load growth rate.
Parameter 12: Solution.GenkW read
This parameter returns the generator kW for AutoAdd mode.
Parameter 13: Solution.GenkW Write
This parameter sets the generator kW for AutoAdd mode.
Parameter 14: Solution.GenPF read
This parameter returns the pf for generators in AutoAdd mode.
Parameter 15: Solution.GenPF Write
This parameter sets the pf for generators in AutoAdd mode.

Parameter 16: Solution.Capkvar read

This parameter returns the capacitor kvar for adding in AutoAdd mode.

Parameter 17: Solution.Capkvar Write

This parameter sets the capacitor kvar for adding in AutoAdd mode.

Parameter 18: Solution.GenMult read

This parameter returns the default multiplier applied to generators (like LoadMult).

Parameter 19: Solution.GenMult Write

This parameter sets the default multiplier applied to generators (like LoadMult).

Parameter 20: Solution.dblHour read

This parameter returns the hour as a double, including fractional part.

Parameter 21: Solution.dblHour Write

This parameter sets the hour as a double, including fractional part.

Parameter 22: Solution.StepSizeMin

This parameter sets the step size in minutes.

Parameter 23: Solution.StepSizeHr

This parameter sets the step size in Hours.

Parameter 24: Solution.Process_Time

This parameter retrieves the time required (microseconds) to perform the latest solution time step, this time does not include the time required for sampling meters/monitors.

Parameter 25: Solution.Total_Time read

This parameter retrieves the accumulated time required (microseconds) to perform the simulation.

Parameter 26: Solution.Total_Time Write

This parameter sets the accumulated time (microseconds) register. The new value for this register must be specified in the argument.

Parameter 27: Solution.Time_TimeStep

This parameter retrieves the time required (microseconds) to perform the latest solution time step including the time required for sampling meters/monitors.

SolutionS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr SolutionS(int32_t Parameter, CStr Argument);

This interface returns a string according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Solution.ModeID

This parameter returns the ID (text) of the present solution mode.

Parameter 1: Solution.LDCurve read

This parameter returns the Load-Duration Curve name for LD modes.

Parameter 2: Solution.LDCurve write

This parameter sets the Load-Duration Curve name for LD modes.

Parameter 3: Solution.DefaultDaily read

This parameter returns the default daily load shape (defaults to "Default").

Parameter 4: Solution.DefaultDaily write

This parameter sets the default daily load shape (defaults to "Default").

Parameter 5: Solution.DefaultYearly read

This parameter returns the default yearly load shape (defaults to "Default").

Parameter 6: Solution.DefaultYearly write

This parameter sets the default yearly load shape (defaults to "Default").

SolutionV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void  SolutionV(int32_t Parameter, VARIANT  *Argument);
```

This interface returns a variant according to the number sent in the variable “parameter”. The parameter can be one of the following:

Parameter 0: Solution.EventLog

This parameter returns an array of strings containing the Event Log.

Parameter 1: Solution.IncMatrix

This parameter returns an array of integers containing the incidence matrix (1-D). Each cell of the incidence matrix is delivered using 3 elements of the array delivered, the first is the row, the second is the column and the third is the value (1/-1). This procedure will only deliver the non-zero elements.

Parameter 2: Solution.BusLevels

This parameter returns an array of integers containing BusLevels array. This array gives a numeric value to each bus to specify how far it is from the circuit’s backbone (a continuous path from the feeder head to the feeder end). It is very handy to understand the circuit’s topology.

Parameter 3: Solution.IncMatrixRows

This parameter returns an array of strings specifying the way the rows of the incidence matrix (PDElements) are organized, depending on the way the Branch to node incidence matrix was calculated (*CalcIncMatrix*/*CalcIncMatrix_O*) the result could be very different.

Parameter 4: Solution.IncMatrixCols

This parameter returns an array of strings specifying the way the cols of the incidence matrix (buses) are organized, depending on the way the Branch to node incidence matrix was calculated (*CalcIncMatrix*/*CalcIncMatrix_O*) the result could be very different.

Parameter 5: Solution.Laplacian

This parameter returns an array of integers containing the Laplacian matrix using the incidence matrix previously calculated, this means that before calling this command the incidence matrix

needs to be calculated using calcincmatrix/calcincmatrix_o. This command will return only the non-zero values in compressed coordinate format (row, col, value).

SwtControls Interface

This interface implements the SwtControls (ISwtControls) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

SwtControlsI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t SwtControlsI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: SwtControls.First

This parameter sets the first SwtControl active. Returns 0 if no more.

Parameter 1: SwtControls.Next

This parameter sets the next SwtControl active. Returns 0 if no more.

Parameter 2: SwtControls.Action read

This parameter gets the open (1) or close (2) action of the switch. No effect if switch is locked. However, reset removes any lock and then closes the switch (shelf state). 0 = none action.

Parameter 3: SwtControls.Action write

This parameter sets open (1) or close (2) the switch. No effect if switch is locked. However, reset removes any lock and then closes the switch (shelf state). 0 = none action (see manual for details).

Parameter 4: SwtControls.IsLocked read

This parameter gets the lock state: {1 locked | 0 not locked}.

Parameter 5: SwtControls.IsLocked write

This parameter sets the lock to prevent both manual and automatic switch operation.

Parameter 6: SwtControls.SwitchedTerm read

This parameter gets the terminal number where the switch is located on the SwitchedObj.

Parameter 7: SwtControls.SwitchedTerm write

This parameter sets the terminal number where the switch is located on the SwitchedObj.

Parameter 8: SwtControls.Count

This parameter gets the total number of SwtControls in the active circuit.

SwtControlsF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
double SwtControlsF(int32_t Parameter, double Argument);
```

This interface returns a floating point number (64 bits) with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: SwtControls.Delay read

This parameter gets the time delay [s] between arming and opening or closing the switch. Control may reset before actually operating the switch.

Parameter 1: SwtControls.Delay write

This parameter sets the time delay [s] between arming and opening or closing the switch. Control may reset before actually operating the switch.

SwtControlsS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr SwtControlsS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: SwtControls.Name read

This parameter gets the name of the active SwtControl.

Parameter 1: SwtControls.Name write

This parameter sets a SwtControl active by name.

Parameter 2: SwtControls.SwitchedObj read

This parameter gets the name of the switched object by the active SwtControl.

Parameter 3: SwtControls.SwitchedObj write

This parameter sets the switched object by name.

SwtControlsV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void SwtControlsV(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: SwtControls.AllNames read

This parameter gets a variant array of strings with all SwtControl names in the active circuit.

Text Interface

This interface can be used to send commands to the text interface of OpenDSS (DSS.Text). The structure of the interface is as follows:

CStr DSSPut_Command(CStr Command);

This interface returns a string pointer (ANSI) with the result of the command sent, the variable "Command" is used as input to send an OpenDSS command to the Text interface of OpenDSS.

Topology Interface

This interface implements the Topology (ITopology) interface of OpenDSS by declaring 3 procedures for accessing the different properties included in this interface.

Tolopolgyl (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t TopologyI(int32_t Parameter, int32_t Argument);

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Topology.NumLoops
This parameter gets the number of loops.
Parameter 1: Topology.NumIsolatedBranches
This parameter gets the number of isolated branches (PD elements and capacitors).
Parameter 2: Topology.NumIsolatedLoads
This parameter gets the number of isolated loads.
Parameter 3: Topology.First
This parameter sets the first branch active, returns 0 if none.
Parameter 4: Topology.Next
This parameter sets the next branch active, returns 0 if none.
Parameter 5: Topology.ActiveBranch
This parameter returns the index of the active Branch.
Parameter 6: Topology.ForwardBranch
This parameter moves forward in the tree, return index of new active branch or 0 if no more.
Parameter 7: Topology.BackwardBranch
This parameter moves back toward the source, return index of new active branch or 0 if no more.
Parameter 8: Topology.LoopedBranch
This parameter moves to looped branch, return index or 0 if none.
Parameter 9: Topology.ParallelBranch
This parameter mode to directly parallel branch, return index or 0 if none.
Parameter 10: Topology.FirstLoad
This parameter sets as active load the first load at the active branch, return index or 0 if none.

Parameter 11: Topology.NextLoad

This parameter sets as active load the next load at the active branch, return index or 0 if none.

Parameter 12: Topology.ActiveLevel

This parameter gets the topological depth of the active branch.

TolopolgyS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
CStr TopologyS(int32_t Parameter, CStr Argument);
```

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Topology.BranchName read

This parameter gets the name of the active branch.

Parameter 1: Topology.BranchName write

This parameter sets the name of the active branch.

Parameter 2: Topology.BusName read

This parameter gets the name of the active Bus.

Parameter 3: Topology.BusName write

This parameter sets the Bus active by name.

TolopolgyV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void TopologyV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: Topology.AllLoopedPairs

This parameter gets a variant array of all looped element names, by pairs.

Parameter 1: Topology.AllIsolatedBranches

This parameter gets a variant array of all isolated branch names.

Parameter 2: Topology.AllIsolatedLoads

This parameter gets a variant array of all isolated load names.

Transformers Interface

This interface implements the Transformers (ITransformer) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

TransformersI (Int) Interface

This interface can be used to read/modify the properties of the Transformers Class where the values are integers. The structure of the interface is as follows:

int32_t TransformersI(int32_t Parameter, int32_t argument) ;

This interface returns an integer (signed 32 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Transformers.NumWindings read
This parameter gets the number of windings on this transformer. Allocates memory; set or change this property first.
Parameter 1: Transformers.NumWindings write
This parameter sets the number of windings on this transformer. Allocates memory; set or change this property first.
Parameter 2: Transformers.Wdg read
This parameter gets the active winding number from 1..NumWindings. Update this before reading or setting a sequence of winding properties (R, Tap, kV, kVA, etc.).
Parameter 3: Transformers.Wdg write
This parameter sets the active winding number from 1..NumWindings. Update this before reading or setting a sequence of winding properties (R, Tap, kV, kVA, etc.).
Parameter 4: Transformers.NumTaps read
This parameter gets the active winding number of tap steps between MinTap and MaxTap.
Parameter 5: Transformers.NumTaps write
This parameter sets the active winding number of tap steps between MinTap and MaxTap
Parameter 6: Transformers.IsDelta read
This parameter gets the information about if the active winding is delta (1) or wye (0) connection.
Parameter 7: Transformers.IsDelta write
This parameter sets the information about if the active winding is delta (1) or wye (0) connection.
Parameter 8: Transformers.First
This parameter sets the first Transformer active. Return 0 if no more.
Parameter 9: Transformers.Next
This parameter sets the next Transformer active. Return 0 if no more.
Parameter 10: Transformers.Count
This parameter gets the number of transformers within the active circuit.
Parameter 11: Transformers.CoreType read
This parameter gets the code for the core type of the active transformer.
Parameter 12: Transformers.CoreType write
This parameter sets the code for the core type of the active transformer using the value given by the user in the argument.

TransformersF (Float) Interface

This interface can be used to read/modify the properties of the Transformers Class where the values are doubles. The structure of the interface is as follows:

double TransformersF(int32_t Parameter, double argument) ;

This interface returns a floating point number (64 bits), the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Transformers.R read
This parameter gets the active winding resistance in %.
Parameter 1: Transformers.R write
This parameter sets the active winding resistance in %.
Parameter 2: Transformers.Tap read
This parameter gets the active winding tap in per-unit.
Parameter 3: Transformers.Tap write
This parameter sets the active winding tap in per-unit.
Parameter 4: Transformers.MinTap read
This parameter gets the active winding minimum tap in per-unit.
Parameter 5: Transformers.MinTap write
This parameter sets the active winding minimum tap in per-unit.
Parameter 6: Transformers.MaxTap read
This parameter gets the active winding maximum tap in per-unit.
Parameter 7: Transformers.MaxTap write
This parameter sets the active winding maximum tap in per-unit.
Parameter 8: Transformers.kV read
This parameter gets the active winding kV rating. Phase-phase for 2 or 3 phases, actual winding kV 1 phase transformer.
Parameter 9: Transformers.kV write
This parameter sets the active winding kV rating. Phase-phase for 2 or 3 phases, actual winding kV 1 phase transformer.
Parameter 10: Transformers.kVA read
This parameter gets the active winding kVA rating. On winding 1, this also determines normal and emergency current ratings for all windings.
Parameter 11: Transformers.kVA write
This parameter sets the active winding kVA rating. On winding 1, this also determines normal and emergency current ratings for all windings.
Parameter 12: Transformers.Xneut read
This parameter gets the active winding neutral reactance [ohms] for wye connections.
Parameter 13: Transformers.Xneut write
This parameter sets the active winding neutral reactance [ohms] for wye connections.

Parameter 14: Transformers.Rneut read
This parameter gets the active winding neutral resistance [ohms] for wye connections. Set less than zero ungrounded wye.
Parameter 15: Transformers.Rneut write
This parameter sets the active winding neutral resistance [ohms] for wye connections. Set less than zero ungrounded wye.
Parameter 16: Transformers.Xhl read
This parameter gets the percent reactance between windings 1 and 2, on winding 1 kVA base. Use for 2 winding or 3 winding transformers.
Parameter 17: Transformers.Xhl write
This parameter sets the percent reactance between windings 1 and 2, on winding 1 kVA base. Use for 2 winding or 3 winding transformers.
Parameter 18: Transformers.Xht read
This parameter gets the percent reactance between windings 1 and 3, on winding 1 kVA base. Use for 3 winding transformers only.
Parameter 19: Transformers.Xht write
This parameter sets the percent reactance between windings 1 and 3, on winding 1 kVA base. Use for 3 winding transformers only.
Parameter 20: Transformers.Xlt read
This parameter gets the percent reactance between windings 2 and 3, on winding 1 kVA base. Use for 3 winding transformers only.
Parameter 21: Transformers.Xlt write
This parameter sets the percent reactance between windings 2 and 3, on winding 1 kVA base. Use for 3 winding transformers only.

TransformersS (String) Interface

This interface can be used to read/modify the properties of the Transformers Class where the values are Strings. The structure of the interface is as follows:

CStr TransformersS(int32_t Parameter, CStr argument) ;

This interface returns a string, the variable “parameter” is used to specify the property of the class to be used and the variable “argument” can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Transformers.XfmrCode read
This parameter gets the name of an XfrmCode that supplies electrical parameters for this transformer.
Parameter 1: Transformers.XfmrCode write
This parameter sets the name of an XfrmCode that supplies electrical parameters for this transformer.
Parameter 2: Transformers.Name read
This parameter gets the active transformer name.

Parameter 3: Transformers.Name write
This parameter sets the active transformer by name
Parameter 4: Transformers.StrWdgVoltages
This parameter gets the voltages at the active winding of the active transformer in string format.

TransformersV (Variant) Interface

This interface can be used to read/modify the properties of the Transformers Class where the values are Variants. The structure of the interface is as follows:

void TransformersV(int32_t Parameter, VARIANT *Argument) ;

This interface returns a Variant, the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

Parameter 0: Transformers.AllNames
This parameter gets a variant array of strings with all Transformer names in the active circuit.
Parameter 1: Transformers.WdgVoltages
This parameter gets a variant array of doubles containing the voltages at the active winding on the active transformer. These voltages come as complex pairs.
Parameter 2: Transformers.WdgCurrents
This parameter gets a variant array of doubles containing the currents at the active winding on the active transformer. These currents come as complex pairs.

VSources Interface

This interface implements the Vsources (IVSources) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

VSourcesI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

int32_t VSourcesI(int32_t Parameter, int32_t Argument);

This interface returns an integer number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: VSources.Count
This parameter returns the number of VSource objects currently defined in the active circuit.
Parameter 1: VSources.First
This parameter sets the first VSource to be active; returns 0 if none.

Parameter 2: VSources.Next
This parameter sets the next VSource to be active; returns 0 if none.
Parameter 3: VSources.Phases read
This parameter gets the number of phases of the active VSource.
Parameter 4: VSources.Phases write
This parameter sets the number of phases of the active VSource.

VSourcesF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double VSourcesF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: VSources.BasekV read
This parameter gets the source voltage in kV.
Parameter 1: VSources.BasekV write
This parameter sets the source voltage in kV.
Parameter 2: VSources.pu read
This parameter gets the source voltage in pu.
Parameter 3: VSources.pu write
This parameter sets the source voltage in pu.
Parameter 4: VSources.Angledeg read
This parameter gets the source phase angle of first phase in degrees.
Parameter 5: VSources.Angledeg write
This parameter sets the source phase angle of first phase in degrees.
Parameter 6: VSources.Frequency read
This parameter gets the source frequency in Hz.
Parameter 7: VSources.Frequency write
This parameter sets the source frequency in Hz.

VSourcesS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr VSourcesS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: VSources.Name read
This parameter gets the name of the active VSource.
Parameter 1: VSources.Name write
This parameter sets the name of the active VSource.

VSourcesV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
void VSourcesV(int32_t Parameter, VARIANT *Argument);
```

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: VSources.AllNames
This parameter gets the name of the active VSource.

XYCurves Interface

This interface implements the XYCurves (IXYCurves) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

XYCurvesI (Int) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

```
int32_t XYCurvesI(int32_t Parameter, int32_t Argument);
```

This interface returns an integer with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: XYCurves.Count
This parameter gets number of XYCurves in active circuit.
Parameter 1: XYCurves.First
This parameter sets first XYCurves object active; returns 0 if none.
Parameter 2: XYCurves.Next
This parameter sets next XYCurves object active; returns 0 if none.
Parameter 3: XYCurves.Npts read
This parameter gets the number of points in X-Y curve.
Parameter 4: XYCurves.Npts read
This parameter sets the number of points in X-Y curve.

XYCurvesF (Float) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

double XYCurvesF(int32_t Parameter, double Argument);

This interface returns a floating point number with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: XYCurves.X read
This parameter gets the interpolated value after setting Y.
Parameter 1: XYCurves.X write
This parameter sets the X value.
Parameter 2: XYCurves.Y read
This parameter gets the interpolated value after setting X.
Parameter 3: XYCurves.Y write
This parameter sets the Y value.
Parameter 4: XYCurves.XShift read
This parameter gets the amount to shift X value from original curve.
Parameter 5: XYCurves.XShift write
This parameter sets the amount to shift X value from original curve.
Parameter 6: XYCurves.YShift read
This parameter gets the amount to shift Y value from original curve.
Parameter 7: XYCurves.YShift write
This parameter sets the amount to shift Y value from original curve.
Parameter 8: XYCurves.XScale read
This parameter gets the factor to scale X values from original curve.
Parameter 9: XYCurves.XScale write
This parameter sets the factor to scale X values from original curve.
Parameter 10: XYCurves.YScale read
This parameter gets the factor to scale Y values from original curve.
Parameter 11: XYCurves.YScale write
This parameter sets the factor to scale Y values from original curve.

XYCurvesS (String) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

CStr XYCurvesS(int32_t Parameter, CStr Argument);

This interface returns a string with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: XYCurves.Name read

This parameter gets the name of the active XYCurve Object.

Parameter 1: XYCurves.Name write

This parameter sets the name of the active XYCurve Object.

XYCurvesV (Variant) Interface

This interface can be used to read/write certain properties of the active DSS object. The structure of the interface is as follows:

void XYCurvesS(int32_t Parameter, VARIANT *Argument);

This interface returns a Variant with the result of the query according to the value of the variable Parameter, which can be one of the following:

Parameter 0: XYCurves.XArray read

This parameter gets the X values as a variant array of doubles. Set Npts to max number expected if setting.

Parameter 1: XYCurves.XArray write

This parameter sets the X values as a variant array of doubles specified in *Argument*. Set Npts to max number expected if setting.

Parameter 2: XYCurves.YArray read

This parameter gets the Y values as a variant array of doubles. Set Npts to max number expected if setting.

Parameter 3: XYCurves.YArray write

This parameter sets the Y values as a variant array of doubles specified in *Argument*. Set Npts to max number expected if setting