# XYCurve Object

The XYCurve object is a general OpenDSS library object used by several circuit elements. For example, the *PVSystem* device uses this class of object for both efficiency curves and power-temperature curves. An XYCurve object may be used for other somewhat arbitrary purposes as well. A user can define an XYCurve object then interpolate (or extrapolate) values from the curve, which might be useful for any of several purposes.

An XYCurve object may be defined and accessed through both the text scripting interface and the COM interface. The basic idea is to first define a curve. Then pass either an X or Y value and get the other value from the curve. This is handled internally via public properties in the object that other objects can access, but the user can also do this from external means on any XYcurve currently defined. One obvious application is that users writing external code to drive the COM interface do not have to write code to manage any curves they might wish to include in their custom model. Simply use an XYCurve object.

The XYCurve object uses linear interpolation. Points past the end of the defined curve are extrapolated using the last two points in the curve.

Recently, shifting and scaling properties were added to the objects so that one may define a base curve and then shift it around to represent other sizes of elements, for examples.

## TEXT INTERFACE DESCRIPTION

| Property | Description |
|---|---|
| (1) npts | Max number of points to expect in curve. This could get reset to the actual number of points defined if less than specified. |
| (2) Points | One way to enter the points in a curve. Enter x and y values as one array in the order [x1, y1, x2, y2, ...]. For example:<br><br>Points=[1,100 2,200 3, 300]<br><br>Values separated by commas or white space. Zero fills arrays if insufficient number of values. |
| (3) Yarray | Alternate way to enter Y values. Enter an array of Y values corresponding to the X values. You can also use the syntax:<br><br>Yarray = (file=filename) !for text file one value per line<br><br>Yarray = (dblfile=filename) !for packed file of doubles<br><br>Yarray = (sngfile=filename) !for packed file of |

| | |
|---|---|
| | singles<br><br>Note: this property will reset Npts to a smaller value if the number of values in the files are fewer |
| (4) Xarray | Alternate way to enter X values. Enter an array of X values corresponding to the Y values. You can also use the syntax:<br><br>Xarray = (file=filename) !for text file one value per line<br><br>Xarray = (dblfile=filename) !for packed file of doubles<br><br>Xarray = (sngfile=filename) !for packed file of singles<br><br>Note: this property will reset Npts to a smaller value if the number of values in the files are fewer. |
| (5) csvfile | Switch input of X-Y curve data to a CSV file containing X, Y points one per line. NOTE: This action may reset the number of points to a lower value. |
| (6) sngfile | Switch input of X-Y curve data to a binary file of SINGLES containing X, Y points packed one after another. NOTE: This action may reset the number of points to a lower value. |
| (7) dblfile | Switch input of X-Y curve data to a binary file of DOUBLES containing X, Y points packed one after another. NOTE: This action may reset the number of points to a lower value. |
| (8) x | Enter a value and then retrieve the interpolated Y value from the Y property. On input, shifted then scaled to original curve. Scaled then shifted on output. |
| (9) y | Enter a value and then retrieve the interpolated X value from the X property. On input, shifted then scaled to original curve. Scaled then shifted on output. |
| (10) Xshift | Shift X property values (in/out) by this amount of offset. Default = 0. Does not change original |

| | |
|---|---|
| | definition of arrays. |
| (11) Yshift | Shift Y property values (in/out) by this amount of offset. Default = 0. Does not change original definition of arrays. |
| (12) Xscale | Scale X property values (in/out) by this factor. Default = 1.0. Does not change original definition of arrays. |
| (13) Yscale | Scale Y property values (in/out) by this factor. Default = 1.0. Does not change original definition of arrays. |
| (14) like | Make like another object, e.g.:<br><br>New XYCurve.Curve2 Like=Curve1 |

## EXAMPLE SCRIPT

You can use the script below to exercise an XYCurve object. A circuit must be defined, but you can exercise the object with no elements other than the initial Vsource.

```
clear
New Circuit.XYCurve  basekv=12.47  Isc3=1000 Isc1=900

// P-T curve is per unit of rated Pmpp vs temperature
// This one is for a Pmpp stated at 25 deg
New XYCurve.MyPvsT npts=4  xarray=[0  25  75  100]  yarray=[1.2 1.0 0.8  0.6]

// efficiency curve is per unit eff vs per unit power
New XYCurve.MyEff npts=4  xarray=[.1  .2  .4  1.0]  yarray=[.86  .9  .93  .97]


XYCurve.MyEff.x=0.6    ! set x value to 0.6
? XYCurve.MyEff.y      ! return corresponding Y in Result window

/*
   Returns 0.94333333 in Result window

   i.e., for 60% power, the inverter efficiency is 94.3333%
*/

XYCurve.MyPvsT.Yscale=0.5  X=75
?  XYCurve.MyPvsT.Y

/*
   Returns 0.4 in Result window (half of 0.8)
*/
```

## COM INTERFACE

### *Concise Pascal Definition of XYCurves Interface*

This is one of the more concise listings of the available interface properties. If you are using VBA or MATLAB to drive the COM interface, these are the names of the properties you will use. Refer to this list if your development environment is not kind enough to prompt you for them.

This interface uses a set of functions similar to what are used in most other OpenDSS interfaces. The idea is to find an object of this class, set it active, and operate on it. If you know the name of the XYCurve object, you can set it active directly by the Name property. Or you can search through the list by the First..Next iterators until you find the one you want or operate on all as you go through the list.

You can get/set the values of the X and Y arrays used to define the curves. Then you can get/set shifting and scaling factors and interpolate X and Y values just like in the text interface. This should be somewhat faster because the values don't have to be converted to and from text.

```
// ************************************************************************//
// DispIntf:  IXYCurvesDisp
// Flags:     (4416) Dual OleAutomation Dispatchable
// GUID:      {97AA7680-E994-4A0C-BAC3-9B67BA49825C}
// ************************************************************************//
  IXYCurvesDisp = dispinterface
    ['{97AA7680-E994-4A0C-BAC3-9B67BA49825C}']
    property Count: Integer readonly dispid 201;
    property First: Integer readonly dispid 202;
    property Next: Integer readonly dispid 203;
    property Name: WideString dispid 204;
    property Npts: Integer dispid 205;
    property Xarray: OleVariant dispid 206;
    function Yarray: OleVariant; dispid 207;
    property x: Double dispid 208;
    property y: Double dispid 209;
    property Xshift: Double dispid 210;
    property Yshift: Double dispid 211;
    property Xscale: Double dispid 212;
    property Yscale: Double dispid 213;
  end;
```

A listing in IDL format is also included below for those programmers more familiar with that format. The files are available in the Source code repository.

## EXAMPLE IN VBA

This example subroutine assumes the existence of a public variable *DSSCircuit* and sets a local variable to the XYCurves interface that is created when OpenDSSEngine is started. Thus, we only have to use the *DSSCurves* variable to access any or all of the XYCurve object presently defined.

This sub first prints out the names of all XYCurve object presently defined. The typical First .. Next iterators are used for this. Then it sets the last one active using the Name property. Like other interfaces in OpenDSS, once this object is set active, you can operate on it with other appropriate interfaces such as DSSElement. Then, the code exercises several properties in the interface and prints out their values to confirm that their actions.

You can put this VBA code in MS Excel and execute it there in the usual manner. (See

SampleDSSDriver.XLS if you are not familiar with this.) You will obviously have to load in a circuit definition that has XYCurve objects defined.

```
Public Sub TestXYCurve()

   Dim DSSCurves As XYCurves
   Dim i As Long
   Dim V As Variant
   Dim LastName As String


   Set DSSCurves = DSSCircuit.XYCurves

   Debug.Print "Version: "; DSSobj.Version

   Debug.Print "Count="; DSSCurves.Count

   i = DSSCurves.First
   Do While i > 0
        LastName = DSSCurves.Name
        Debug.Print i; ", "; LastName
        i = DSSCurves.Next
   Loop

   DSSCurves.Name = LastName   ' set active by name
   Debug.Print "Active curve="; DSSCurves.Name  ' check to make sure it worked
   Debug.Print "Num Points = "; DSSCurves.Npts

   V = DSSCurves.Xarray
   Debug.Print "Xarray=';"
   For i = LBound(V) To UBound(V)
        Debug.Print V(i); ", ";
   Next i
   Debug.Print " "

   V = DSSCurves.Yarray
   Debug.Print "Yarray=';"
   For i = LBound(V) To UBound(V)
        Debug.Print V(i); ", ";
   Next i
   Debug.Print " "


   ' test x and y actions
   With DSSCurves
            .x = 0.4
            Debug.Print "For X="; .x; " Y="; .y

            ' Scale curve down
            .Yscale = 0.5 ' derated by half
            .x = 0.4
            Debug.Print "For X="; .x; " and Yscale="; .Yscale; " Y="; .y

            .Yscale = 1#
            .Yshift = 1#  ' shift curve up by 1
            .x = 0.4
            Debug.Print "For X="; .x; " and Yshift="; .Yshift; " Y="; .y

            ' now go the other way ... set Y and get x
            .y = 1.93  ' shift is still in effect
            Debug.Print "For Y="; .y; " and Yshift="; .Yshift; " X="; .x
```

```
            ' back to original
            .Yshift = 0#
            .y = 0.93  ' shift is still in effect
            Debug.Print "For Y="; .y; " and Yshift="; .Yshift; " X="; .x

   End With

End Sub
```

## COM INTERFACE DEFINITION IN IDL FORMAT

```
[
    uuid(97AA7680-E994-4A0C-BAC3-9B67BA49825C),
    helpstring("Dispatch interface for XYCurves Object"),
    dual,
    oleautomation
  ]
  interface IXYCurves: IDispatch
  {
    [propget, id(0x000000C9), helpstring("Number of XYCurve Objects")]
    HRESULT _stdcall Count([out, retval] long* Value);
    [propget, id(0x000000CA), helpstring("Sets first XYcurve object active; returns 0
if none.")]
    HRESULT _stdcall First([out, retval] long* Value);
    [propget, id(0x000000CB), helpstring("Advances to next XYCurve object; returns 0
if no more objects of this class")]
    HRESULT _stdcall Next([out, retval] long* Value);
    [propget, id(0x000000CC), helpstring("Name of active XYCurve Object")]
    HRESULT _stdcall Name([out, retval] BSTR* Value);
    [propput, id(0x000000CC), helpstring("Get Name of active XYCurve Object")]
    HRESULT _stdcall Name([in] BSTR Value);
    [propget, id(0x000000CD), helpstring("Get/Set Number of points in X-Y curve")]
    HRESULT _stdcall Npts([out, retval] long* Value);
    [propput, id(0x000000CD), helpstring("Get/Set Number of Points in X-Y curve")]
    HRESULT _stdcall Npts([in] long Value);
    [propget, id(0x000000CE), helpstring("Get/Set X values as a Variant array of
doubles. Set Npts to max number expected if setting")]
    HRESULT _stdcall Xarray([out, retval] VARIANT* Value);
    [propput, id(0x000000CE), helpstring("Get/Set X values as a Variant array of
doubles. Set Npts to max number expected if setting")]
    HRESULT _stdcall Xarray([in] VARIANT Value);
    [propget, id(0x000000CF), helpstring("Get/Set Y values in curve; Set Npts to max
number expected if setting")]
    HRESULT _stdcall Yarray([out, retval] VARIANT* Value);
    [propput, id(0x000000CF), helpstring("Get/Set Y values in curve; Set Npts to max
number expected if setting")]
    void _stdcall Yarray([in] VARIANT Value);
    [propget, id(0x000000D0), helpstring("Set X value or get interpolated value after
setting Y")]
    HRESULT _stdcall x([out, retval] double* Value);
    [propput, id(0x000000D0)]
    HRESULT _stdcall x([in] double Value);
    [propget, id(0x000000D1), helpstring("Y value for present X or set this value then
get corresponding X")]
    HRESULT _stdcall y([out, retval] double* Value);
    [propput, id(0x000000D1), helpstring("Set Y value or get interpolated Y value
after setting X")]
    HRESULT _stdcall y([in] double Value);
    [propget, id(0x000000D2), helpstring("Amount to shift X value from original
curve")]
    HRESULT _stdcall Xshift([out, retval] double* Value);
    [propput, id(0x000000D2)]
    HRESULT _stdcall Xshift([in] double Value);
    [propget, id(0x000000D3), helpstring("amount to shift Y valiue from original
curve")]
    HRESULT _stdcall Yshift([out, retval] double* Value);
    [propput, id(0x000000D3)]
    HRESULT _stdcall Yshift([in] double Value);
    [propget, id(0x000000D4), helpstring("Factor to scale X values from original
curve")]
    HRESULT _stdcall Xscale([out, retval] double* Value);
    [propput, id(0x000000D4), helpstring("Factor to scale X values from original
```

```
curve")]
    HRESULT _stdcall Xscale([in] double Value);
    [propget, id(0x000000D5), helpstring("Factor to scale Y values from original
curve")]
    HRESULT _stdcall Yscale([out, retval] double* Value);
    [propput, id(0x000000D5), helpstring("Amount to scale Y values from original
curve. Represents a curve shift.")]
    HRESULT _stdcall Yscale([in] double Value);
  };
```