Jed OLE Scripting

By Alexei Novikov

OLE Scripting (a.k.a ActiveX Scripting) is one of the glorified Microsoft technologies. It had enough time to develop to something really useful. Basically it's a way for applications to communicate with each other and allow simple scripting languages control applications.

You can use JED's OLE scripting from any language that supports it. Just about every language does, for instance VC++, Delphi and Visual Basic. But you don't even need to have any of those. You can use freely downloadable Windows Scripting Host (WSH).WSH comes with Win98 or you can download it Here's the link. It allows you to write scripts in VBScript or JavaScript.

To use JED scripting you need to have JED 0.8+ running. The name of the JED interface object is "JED.App". So, for instance using WSH with JavaScript the line to create the JED object would be:

```
var Jed = WScript.CreateObject("JED.App");
```

For Delphi it will be:

```
var jed:variant;
...
Jed:=CreateOLEObject('JED.App');
```

If JED is not running, it creating of the object will fail. To access functions and procedure use, for instance:

```
n = Jed.Nthings;
Jed.NewLevel(false);
```

To access properties use:

```
dir=Jed.ProjectDir;
Jed.IsMots = false;
```

Here's the list Objects, their methods and properties copied/pasted directly from JED code:

```
TJEDApp=class(TAutoObject)

   Procedure Release;


   Procedure NewLevel(mots:WordBool);
   Procedure OpenLevel(const name:string);
   Procedure SaveJED(const name:string);
   Procedure SaveJKL(const name:string);
   Procedure UpdateMap;
   Procedure GetCurSF(var sc,sf:integer);
   Procedure SetCurSF(sc,sf:integer);
   Procedure GetCurVX(var sc,vx:integer);
   Procedure SetCurVX(sc,vx:integer);
   Procedure GetCurED(var sc,sf,ed:integer);
   Procedure SetCurED(sc,sf,ed:integer);
   Procedure GetCurFR(var th,fr:integer);
   Procedure SetCurFR(th,fr:integer);


   {Picking}

    Function PickThing(const curThing:String):String;
    Function PickTemplate(const curtpl:string):string;
    Function PickCMP(const CurCMP:string):string;
    Function PickWav(const CurWav:string):string;
    Function PickMAT(const CurMAT:string):string;
    Function PickCOG(const CurCog:string):string;
    Function Pick3DO(const cur3do:String):String;
    Function PickAI(const CurAI:string):string;
    Function PickKEY(const CurKEY:string):string;
    Function PickSND(const CurSND:string):string;
    Function PickPUP(const CurPUP:string):string;


    {Stuff}

    Procedure GetCamXYZ(var x,y,z:double);
    Procedure GetGridXYZ(var x,y,z:double);
```

```
{Properties}

Property Version:double;
Property MapMode:integer;
Property CurSC:integer;
Property CurTH:integer;
Property CurLT:integer;


Property Level:Variant;
Property GameDir:String;
Property ProjectDir:String;
Property JEDDir:String;
Property IsMots:WordBool;
```

TJEDApp is the object returned by .CreateObject("JED.App"). It contains root functionality. Here's the quick explanation for all functions:

```
Procedure Release;
```

This function signals the object to free all resources taken. You should call it after you're done working with it. In case of WSH script it isn't necessary for the main object (created by CreateObject()).

```
Procedure NewLevel(mots:WordBool);
```

Creates a new level. The boolean parameter defines if the new level will be a MOTS (true) or JK (false) level.

```
Procedure OpenLevel(const name:string);
```

Opens a level. It can be either .JED or .JKL file.

```
Procedure SaveJED(const name:string);
Procedure SaveJKL(const name:string);
```

Saves a level to .JKL or .JED. Careful - it doesn't check for existence of files and overwrites them without warning

```
Procedure UpdateMap;
```

This one redraws the JED editor area. You'll need to do that after you've made changes to the level, etc.

```
Procedure GetCurSF(var sc,sf:integer);
Procedure SetCurSF(sc,sf:integer);
Procedure GetCurVX(var sc,vx:integer);
Procedure SetCurVX(sc,vx:integer);
Procedure GetCurED(var sc,sf,ed:integer);
Procedure SetCurED(sc,sf,ed:integer);
Procedure GetCurFR(var th,fr:integer);
Procedure SetCurFR(th,fr:integer);
```

This group of functions gets/sets the current item in the editor. I goofed up here. Later in programming I found that neither VBScript nor JavaScript seem to support "var" parameters. It works in Delphi though. I made the function equivalents for other objects, but I missed these. So you might not be able to use .GetXX functions.

{Picking}

```
Function PickThing(const curThing:String):String;
Function PickTemplate(const curtpl:string):string;
Function PickCMP(const CurCMP:string):string;
Function PickWav(const CurWav:string):string;
Function PickMAT(const CurMAT:string):string;
Function PickCOG(const CurCog:string):string;
Function Pick3DO(const cur3do:String):String;
Function PickAI(const CurAI:string):string;
Function PickKEY(const CurKEY:string):string;
Function PickSND(const CurSND:string):string;
Function PickPUP(const CurPUP:string):string;
```

These functions invoke standard JED resource pickers. If you press "Esc" or "Cancel" button in a picker, the string you passed as a CurXXX will be returned.

{Stuff}

```
Procedure GetCamXYZ(var x,y,z:double);
Procedure GetGridXYZ(var x,y,z:double);
```

These two is a part of what I started, but didn't have time to finish. One returns current camera position, the other - grid center. They use "var" parameters to you can't use them from VBScript or JavaScript...

{Properties}

Properties behave like variables. I.e. you can put them in expressions as variables and assign stuff to them. For instance to use the Version property you would use something like this:
If Jed.Version=0.8 then ...

Property Version:double;

Return Jed version. You should check this property to ensure the availability of the interface functions that will be added in later versions.

Property MapMode:integer;

Returns current map mode:
0 - sectors
1 - surfaces
2 - vertices
3 - things
4 - edges
5 - lights
6 - frames

Property CurSC:integer read GetCurSC write SetCurSC;
Property CurTH:integer;
Property CurLT:integer;

These properties return/set current Sector, Thing or Light. You should be in the appropriate mode

Property Level:Variant read GetLevel;

This property returns TOLELevel object that lets you access the current loaded level

Property GameDir:String;
Property ProjectDir:String;
Property JEDDir:String;

These properties return JED's directories. GameDir is JK or MOTS directory (depending on whether you're in JK or MOTS mode). ProjectDir is the current project directory. If the project was created or imported, but not saved, ProjectDir returns an empty string. JEDDir is JED directory. That is, where JED.EXE is.

Property IsMots:WordBool;

This property lets you read and set the MOTS mode. True - MOTS mode, false - JK mode.

Here's the TOLELevel object, which is returned by TJEDApp.Level property.

```
TOLELevel=class(TAutoObject)

   Function GetSector(n:integer):variant;
   Function GetSurface(sc,sf:integer):variant;
   Function GetThing(n:integer):variant;
   Function GetLight(n:integer):variant;


   Procedure DeleteSector(n:integer);
   Procedure DeleteThing(n:integer);
   Procedure DeleteLight(n:integer);

   Function AddSector:Integer;
   Function AddThing:Integer;
   Function AddLight:Integer;

   Function GetLayerName(n:integer):String;
   Function AddLayer(const name:string):integer;

   Property NSectors:integer read GetNSectors;
   Property NThings:integer read GetNThings;
   Property NLights:integer read GetNLights;
   Property MasterCMP:string read GetMasterCMP write SetMasterCMP;
```

Function GetSector(n:integer):variant;
Function GetSurface(sc,sf:integer):variant;
Function GetThing(n:integer):variant;
Function GetLight(n:integer):variant;

These functions return respectively, TOLESector, TOLESurface, TOLEThing and TOLELight objects corresponding to the specified item. that are used to interface with level's data.

Procedure DeleteSector(n:integer);
Procedure DeleteThing(n:integer);
Procedure DeleteLight(n:integer);

These functions delete the corresponding items. All references to this item are removed.

```
Function AddSector:Integer;
Function AddThing:Integer;
Function AddLight:Integer;
```

These add an item and return its number.

```
Function GetLayerName(n:integer):String;
Function AddLayer(const name:string):integer;
```

These are pretty self-explanatory

```
Property NSectors:integer;
Property NThings:integer;
Property NLights:integer;

Property MasterCMP:string;
```

So are these.

TOLESector is the object used to interface with sector.

```
TOLESector=class(TAutoObject)

   Function AddSurface:Integer;
   Function InsertSurface(n:integer):integer;
   Procedure DeleteSurface(n:integer);
   Function GetSurface(n:integer):Variant;

   Function AddVertex(x,y,z:double):Integer;
   Function FindVertex(x,y,z:double):Integer;
   Procedure DeleteVertex(n:integer);
   Function GetVertex(n:integer;x,y,z:double):Variant;
   Procedure SetVertex(n:integer;x,y,z:double);

   Procedure Update;
   Procedure Release;

   Procedure GetTint(var r,g,b:double);
   Procedure SetTint(r,g,b:double);

   Property NVertices:Integer;
   Property NSurfaces:Integer;

   Property Tinx_R:double;
   Property Tinx_G:double;
   Property Tinx_B:double;

   Property Flags:longint;
   Property Ambient:double;
   Property ExtraLight:double;
   Property ColorMap:String;
   Property Sound:string read;
   Property Volume:double;
   Property Layer:Integer;
```

```
Function AddSurface:Integer;
Function InsertSurface(n:integer):integer;
Procedure DeleteSurface(n:integer);
Function GetSurface(n:integer):Variant;
```

These functions respectively, add/insert an empty surface, delete or retrieve and interface object.

```
Function AddVertex(x,y,z:double):Integer;
Function FindVertex(x,y,z:double):Integer;
Procedure DeleteVertex(n:integer);

Function GetVertex(n:integer;x,y,z:double):Variant;
Procedure SetVertex(n:integer;x,y,z:double);
```

These functions add/delete/get/set the vertices to/from sector's vertices pool and FindVertex() lets you check if the vertex with these x,y,z is in the sector. The vertices that are "close" (less than 0.0001 units away) are considered equal. Never mind that "variant" return value of GetVertex(). It's simply a typo in my code, that function doesn't return anything.

```
Procedure Update;
```

Updates some JED variables. Call it after you're done changing the sector.

Procedure Release;

Call this procedure when you're done with the sector. After you called the .Release method, the interface object is released and you can't use it anymore.

Property Tinx_R:double;
Property Tinx_G:double;
Property Tinx_B:double;
Procedure GetTint(var r,g,b:double);
Procedure SetTint(r,g,b:double);

These are the functions to get/set sector's tint.

Property NVertices:Integer;

Property NSurfaces:Integer;

As they says - returns number of vertices and surfaces in sector.

Property Flags:Integer;
Property Ambient:double;
Property ExtraLight:double;
Property ColorMap:String;
Property Sound:string;
Property Volume:double;
Property Layer:Integer;

Well, those ate self-explanatory - they get and set sector attributes.

TOLESurface is the object returned by TOLELevel.GetSurface() and TOLESector.GetSurface().

```
TOLESurface=class(TAutoObject)

   Procedure Release;
   Procedure Update;
   Procedure UpdateSector;

   Procedure GetAdjoin(var sc,sf:integer);
   Procedure SetAdjoin(sc,sf:integer);
   Procedure GetFlags(var AdjoinFlags, SurfFlags, FaceFlags:Longint);
   Procedure SetFlags(AdjoinFlags, SurfFlags, FaceFlags:Longint);
   Procedure GetGeoLightTex(var geo,light,tex:integer);
   Procedure SetGeoLightTex(geo,light,tex:integer);

   {Vertices}
   Procedure InsertVertex(n:integer;nvx:integer);
   Procedure DeleteVertex(n:integer);
   Function GetVertex(n:integer):Integer;
   Procedure SetVertex(n:integer;nvx:integer);
   Procedure GetVertexUV(n:integer;var u,v:double);
   Procedure SetVertexUV(n:integer;u,v:double);
   Function GetVertexLight(n:integer):double;
   Procedure SetVertexLight(n:integer;light:double);
   Procedure GetVertexRGB(n:integer;var r,g,b:double);
   Procedure SetVertexRGB(n:integer;r,g,b:double);

   Function GetAdjoinSC:Integer;
   Function GetAdjoinSF:Integer;
   Function GetVertexU(n:integer):Double;
   Function GetVertexV(n:integer):Double;
   Function GetVertexR(n:integer):Double;
   Function GetVertexG(n:integer):Double;
   Function GetVertexB(n:integer):Double;

   Property AdjoinFlags:Integer;
   Property SurfFlags:Integer;
   Property FaceFlags:Integer;
   Property Geo:integer;
   Property Light:integer;
   Property Tex:integer;

   Property Material:string;
   Property ExtraLight:double;
   Property TXScale:double;
```

Procedure Release;
Procedure Update;

Basically the same purpose as in TOLESector. For surface .Update() recalculates the surface's normal and texturing.

Procedure UpdateSector;

This one calls the .Update for a sector the surface belongs to. Just for a shortcut.
```
Procedure GetAdjoin(var sc,sf:integer);
Procedure SetAdjoin(sc,sf:integer);
Function GetAdjoinSC:Integer;
Function GetAdjoinSF:Integer;
```

These a re the functions to get/set the adjoin of a surface. The GetAdjoin() is also split in two because of that "var" parameter thing. Sc is the number of sector and sf is the number of surface in it. If there's no adjoin for this surface, -1 is returned. Likewise, to remove adjoin, set adjoin to -1,-1.

```
Procedure GetFlags(var AdjoinFlags, SurfFlags, FaceFlags:Longint);
Procedure SetFlags(AdjoinFlags, SurfFlags, FaceFlags:Longint);
Property AdjoinFlags:Integer;
Property SurfFlags:Integer;
Property FaceFlags:Integer;
```

These serves to get/set surface flags.

```
Procedure GetGeoLightTex(var geo,light,tex:integer);
Procedure SetGeoLightTex(geo,light,tex:integer);
Property Geo:integer;
Property Light:integer;
Property Tex:integer;
```

The same for Geo, light and tex fileds of the surface.Properties gor, light and tex are read-only. Use SetGeoLightTex() to set them.

```
Procedure InsertVertex(n:integer;nvx:integer);

Procedure DeleteVertex(n:integer);
Function GetVertex(n:integer):Integer;
Procedure SetVertex(n:integer;nvx:integer);
```

These functions let you insert/delete and change surface. The "n" parameter in these functions is the index of the vertex to perform operation at. The "nvx" parameter (or return value in case of GetVertex()) is the index of the vertex in sector's vertex list.

```
Procedure GetVertexUV(n:integer;var u,v:double);
Procedure SetVertexUV(n:integer;u,v:double);
Function GetVertexLight(n:integer):double;
Procedure SetVertexLight(n:integer;light:double);
Procedure GetVertexRGB(n:integer;var r,g,b:double);
Procedure SetVertexRGB(n:integer;r,g,b:double);
Function GetVertexU(n:integer):Double;
Function GetVertexV(n:integer):Double;
Function GetVertexR(n:integer):Double;
Function GetVertexG(n:integer):Double;
Function GetVertexB(n:integer):Double;
```

These functions let you retrieve and set the vertex attributes. U and V are texture coordinates, Light is B/W intensity at the vertex and RGB - are Red, Green, Blue components, respectively. You can access RGB components of the vertex in any level, but they only play any role in MOTS levels. In Jk levels they don't even make it to .JKL file.

```
Property Material:string;
Property ExtraLight:double;
Property TXScale:double;
```

These properties simply get/set surface attributes.

```
TOLEThing:

   Procedure Release;
   Procedure Update;

   Procedure GetXYZ(var x,y,z:double);
   Procedure SetXYZ(x,y,z:double);
   Procedure GetOrient(var pch,yaw,rol:double);
   Procedure SetOrient(pch,yaw,rol:double);

   Procedure GetValue(n:Integer;var name,val:string);
   Procedure SetValue(n:Integer;const name,val:string);
   Function GetValueName(n:Integer):String;
   Function GetValueData(n:integer):string;

   Function AddValue(const name,val:string):integer;
   Procedure InsertValue(n:Integer;const name,val:string);
   Procedure DeleteValue(n:integer);

   Property X:double;
   Property Y:double;
   Property Z:double;
   Property PCH:double;
   Property YAW:double;
   Property ROL:double;
```

```
   Property NValues:Integer;
   Property Sector:Integer;
   Property Template:string;
   Property layer:integer;


Procedure Release;
Procedure Update;
```

The same as for sector and surface

```
Procedure GetXYZ(var x,y,z:double);
Procedure SetXYZ(x,y,z:double);
Procedure GetOrient(var pch,yaw,rol:double);
Procedure SetOrient(pch,yaw,rol:double);
Property X:double;
Property Y:double;
Property Z:double;
Property PCH:double;
Property YAW:double;
Property ROL:double;
```

These serve to get/set position and orientation of the thing. The X,Y,Z and PCH, Yaw, ROL properties are read-only. To set them use SetXYZ and SetOrient(). Such a mix is due to that "var" parameter problem.

```
Procedure GetValue(n:Integer; var name,val:string);
Procedure SetValue(n:Integer; const name,val:string);
Function GetValueName(n:Integer):String;
Function GetValueData(n:integer):string;
Function AddValue(const name,val:string):integer;
Procedure InsertValue(n:Integer;
const name,val:string);
Procedure DeleteValue(n:integer);
Property NValues:Integer;
```

These properties and functions are for operating thing's values. Things like "numframes=2" after a thing in JKL file.

```
Property Sector:Integer;
Property Template:string;
Property layer:integer;
```

Get/set the respective thing attribute.

```
TOLELight:

   Procedure GetXYZ(var x,y,z:double);
   Procedure SetXYZ(x,y,z:double);
   Procedure GetRGB(var r,g,b:double);
   Procedure SetRGB(r,g,b:double);
   Procedure GetIntensity(var white,rgb:double);
   Procedure SetIntensity(white,rgb:double);

   Property R:double read GetR;
   Property G:double read GetG;
   Property B:double read GetB;
   Property X:double read GetX;
   Property Y:double read GetY;
   Property Z:double read GetZ;

   Property Intensity:double;
   Property RGBIntensity:double;

   Property Range:double;
   Property Layer:integer;
   Property Flags:integer;
```

Well, here no additional explanation is due. The properties and functions simple get/set the light attributes.

New in JED 0.81:

There are several bug fixed in OLE scripting routines since 0.8. Here's the list (you can also fin it in History.txt):

```
- Fixed bug in CurLT property
- Here's the list of added methods.
- Picking functions (.PickXXX) now pop up the resource picker in
  front and return the focus to the calling application when done.
- Fixed a bug in TOLESector.GetVertex()
```

Here's the list of added methods and objects:

```
TJEDApp:
```

```
Function PickSPR(const CurSPR:string):string;
Function PickPAR(const CurPAR:string):string;
Function PickPER(const CurPER:string):string;

Procedure ReloadTemplates;
Procedure PanMessage(mtype:integer;const msg:string);
Procedure SendKey(shift:integer;key:integer);
Function ExecuteMenu(const itemref:string):WordBool;

Procedure SetCamXYZ(x,y,z:double);
Procedure SetGridXYZ(x,y,z:double);
Procedure GetGridAxes(var xx,xy,xz,yx,yy,yz,zx,zy,zz:double);
Function GetGridAxis(naxis,ncoord:integer):double;
Procedure SetGridAxes(xx,xy,xz,yx,yy,yz:double);

Procedure GetCamAxes(var xx,xy,xz,yx,yy,yz,zx,zy,zz:double);
Function GetCamAxis(naxis,ncoord:integer):double;
Procedure SetCamAxes(xx,xy,xz,yx,yy,yz:double);

Procedure SetPreviewCamXYZ(x,y,z:double);
Procedure GetPreviewCamXYZ(var x,y,z:double);
Procedure SetPreviewCamPY(pch,yaw:double);
Procedure GetPreviewCamPY(var pch,yaw:double);
Function GetPreviewCamParam(what:integer):double;

Function NMultiSelected(what:integer):integer;
Procedure ClearMultiselection(what:integer);
Procedure RemoveFromMultiselection(what,n:integer);

Function GetSelectedSC(n:integer):integer;
Function GetSelectedTH(n:integer):integer;
Function GetSelectedLT(n:integer):integer;

procedure GetSelectedSF(n:integer;var sc,sf:integer);
procedure GetSelectedED(n:integer;var sc,sf,ed:integer);
procedure GetSelectedVX(n:integer;var sc,vx:integer);
procedure GetSelectedFR(n:integer;var th,fr:integer);

Function GetSelectedSFop(n:integer;what:integer):integer;
Function GetSelectedEDop(n:integer;what:integer):integer;
Function GetSelectedVXop(n:integer;what:integer):integer;
Function GetSelectedFRop(n:integer;what:integer):integer;

Function SelectSC(sc:integer):integer;
Function SelectSF(sc,sf:integer):integer;
Function SelectED(sc,sf,ed:integer):integer;
Function SelectVX(sc,vx:integer):integer;
Function SelectTH(th:integer):integer;
Function SelectFR(th,fr:integer):integer;
Function SelectLT(lt:integer):integer;

Function FindSelectedSC(sc:integer):integer;
Function FindSelectedSF(sc,sf:integer):integer;
Function FindSelectedED(sc,sf,ed:integer):integer;
Function FindSelectedVX(sc,vx:integer):integer;
Function FindSelectedTH(th:integer):integer;
Function FindSelectedFR(th,fr:integer):integer;
Function FindSelectedLT(lt:integer):integer;


{Properties}
Property IsPreviewActive:WordBool;
Property CurSF:integer;
Property CurED:integer;
Property CurVX:integer;
Property CurFR:integer;
Property GridX:double;
Property GridY:double;
Property GridZ:double;
Property CamX:double;
Property CamY:double;
Property CamZ:double;
```

Function PickSPR(const CurSPR:string):string;
Function PickPAR(const CurPAR:string):string;
Function PickPER(const CurPER:string):string;

These are simply the additional pickers for .SPR, .PAR and .PER files. PER picker is only applicable to MOTS.

Procedure ReloadTemplates;

Has the same effect as menu "Commands\Reload Templates".

Procedure PanMessage(mtype:integer;const msg:string);

Adds a message to JED's message pan. "mtype" parameter is the type of message: 0 - informational, 1 - warning, 2 - error.

```
Procedure SendKey(shift:integer;key:integer);
Function ExecuteMenu(const itemref:string):WordBool;
```

These two methods let you simulate keypresses and picking menus in JED's main window. "Shift" parameter of SendKey is a bit flag set. 1 - Ctrl, 2 - Shift, 4 - Alt. "Key" is the Windows code of the key. Look up VK_XXX constants in Windows reference. "Itemref" parameter of the reference to a menu item. The items are references just like files. For instance, the "Calculate lights" menu would be referenced as "Tools\Calculate Lighting". Don't forget that in, say, JavaScript "\" character needs to be doubled. So in JavaScript this reference will be "Tools\\Calculate Lighting".

```
Procedure SetCamXYZ(x,y,z:double);
Procedure SetGridXYZ(x,y,z:double);
Procedure GetGridAxes(var xx,xy,xz,yx,yy,yz,zx,zy,zz:double);
Function GetGridAxis(naxis,ncoord:integer):double;
Procedure SetGridAxes(xx,xy,xz,yx,yy,yz:double);

Procedure GetCamAxes(var xx,xy,xz,yx,yy,yz,zx,zy,zz:double);
Function GetCamAxis(naxis,ncoord:integer):double;
Procedure SetCamAxes(xx,xy,xz,yx,yy,yz:double);
Property GridX:double;
Property GridY:double;
Property GridZ:double;
Property CamX:double;
Property CamY:double;
Property CamZ:double;
```

This set of methods and properties lets you get/set position and orientation of the wireframe view (main window) camera and the grid. The "Grid*" and "Cam*" properties are read-only. To set them you should use .SetCamXYZ() and .SetGridXYZ(). The orientation of the camera and the grid is specified by 3 axes - X,Y and Z - which represent the rotated coordinate system. .GetGridAxes() and .GetCamAxes() reads all of them. .GetCamAxis() and .GetGridAxis() lets you read individual components (for use in JavaScript that can't handle VAR parameters). "naxis" parameter specifies the axis to read - 1 -x, 2 - y, 3 -z. "ncoord" - the component of the axis, the same way 1 -x, 2 -y, 3 - z. So, .GetGridAxis(1,2) will return Y component of X axis. If you're wondering how PCH,YAW,ROL orientation can be translated to that 3 axis orientation, the answer is - it can be. But it involves matrices.

```
Procedure SetPreviewCamXYZ(x,y,z:double);
Procedure GetPreviewCamXYZ(var x,y,z:double);
Procedure SetPreviewCamPY(pch,yaw:double);
Procedure GetPreviewCamPY(var pch,yaw:double);
Function GetPreviewCamParam(what:integer):double;
Property IsPreviewActive:WordBool;
```

This set of methods lets you get/set position and orientation of the camera in 3D Preview. IsPreviewActive property returns true if 3D Preview is displayed and false otherwise. If 3D Preview is not active, settings its camera will have no effect. .GetPreviewCamParam() lets you get parameters one by one (that VAR parameter thing again). "what" parameter specified which part to get. 1 - x, 2 - y, 3 - z,4 - PCH, 5 - YAW. Notice that 3D Preview never tilts your view sideways, so ROL in 3D Preview is always 0.

```
Function NMultiSelected(what:integer):integer;
Procedure ClearMultiselection(what:integer);
Procedure RemoveFromMultiselection(what,n:integer);
```

These methods let you get number of multiselected items, clear multiselection and remove specific item from multiselection, accordingly. "what" parameter specifies the kind of items to operate on. It accepts the same constants that are used by .MapMode property:
0 - sectors
1 - surfaces
2 - vertices
3 - things
4 - edges
5 - lights
6 - frames
"n" parameter of .RemoveFromMultiselection() is the index of item to remove from multiselection. Note that it isn't the number of item in the level, rather its index in multiselection.

```
Function GetSelectedSC(n:integer):integer;
Function GetSelectedTH(n:integer):integer;
Function GetSelectedLT(n:integer):integer;

procedure GetSelectedSF(n:integer;var sc,sf:integer);
procedure GetSelectedED(n:integer;var sc,sf,ed:integer);
procedure GetSelectedVX(n:integer;var sc,vx:integer);
procedure GetSelectedFR(n:integer;var th,fr:integer);

Function GetSelectedSFop(n:integer;what:integer):integer;
Function GetSelectedEDop(n:integer;what:integer):integer;
Function GetSelectedVXop(n:integer;what:integer):integer;
Function GetSelectedFRop(n:integer;what:integer):integer;

Function SelectSC(sc:integer):integer;
```

```
Function SelectSF(sc,sf:integer):integer;
Function SelectED(sc,sf,ed:integer):integer;
Function SelectVX(sc,vx:integer):integer;
Function SelectTH(th:integer):integer;
Function SelectFR(th,fr:integer):integer;
Function SelectLT(lt:integer):integer;

Function FindSelectedSC(sc:integer):integer;
Function FindSelectedSF(sc,sf:integer):integer;
Function FindSelectedED(sc,sf,ed:integer):integer;
Function FindSelectedVX(sc,vx:integer):integer;
Function FindSelectedTH(th:integer):integer;
Function FindSelectedFR(th,fr:integer):integer;
Function FindSelectedLT(lt:integer):integer;
```

These methods let you get selected items (GetSelected* group), add items to multiselection (select* group) and check if item is selected/get its index (FindSelected* group). SC is the acronym for Sectors, SF - surfaces, ED - edhes, VX -vertices, TH - things, FR - frames, LT - lights. Notice that although you can access all of those independently, only the multiselection of items of the current map mode is guaranteed to be valid. I.e. in sector mode the sector multiselection will be valid, but you can't rely on validity of surface multiselection. To get relyably valid surface multiselection you need to switch to surface mode. However, you can rely on Thing and Light multiselection to be always valid.
"n" parameter is the index of item in multiselection. It ranges from 0 to NSelected() for this type of items. For "GetSelectedXXop" method group "what" parameter specifies which part to get. For surfaces: 1 - sector, 2 - surface, for edges: 1 - sector, 2 - surface, 3 - edge, for vertices: 1 - sector, 2 - vertex, for frames: 1 - thing, 2 - frame.
The "Select" group of methods return the index of the item in multiselection. They check if the item is already present in multiselection.
"FindSelected" group of methods lets you check if an item is in multiselection already. if it is, its index in multiselection is returned. Otherwise it returns -1. Note that you can pass the index (if it isn't -1) returned by "FindSelected" to .removeFromMultiselection() to unselect this item.

```
{Properties}
Property CurSF:integer;
Property CurED:integer;
Property CurVX:integer;
Property CurFR:integer;
```

These properties fill the gap in 0.8's functionality letting JavaScript (and VBScript) retrieve currently selected surface, edge, vertex and frame. Note that they are formed by 2 or more values. For instance the currently selected edge will be referenced by 3 values: CurSC,CurSF and CurED.

```
TOLELevel=class(TAutoObject)
  Function GetCOG(n:integer):variant;
  Procedure DeleteCOG(n:integer);
  Function AddCOG(const name:string):Integer;
```

```
Function GetCOG(n:integer):variant;
Procedure DeleteCOG(n:integer);
Function AddCOG(const name:string):Integer;
```

These three methods let you add/delete and retrieve interface to placed COGs. .getCOG() method returns the TOLECOG iterface described below. .AddCOG() methods also reads the requested COG and loads its values. Returns the index of a new COG COG list.

```
TOLESurface=class(TAutoObject)
   Procedure GetNormal(var x,y,z:double);
   Property UScale:double;
   Property VScale:double;
   Property NormalX:double;
   Property NormalY:double;
   Property NormalZ:double;
```

```
Procedure GetNormal(var x,y,z:double);
Property NormalX:double;
Property NormalY:double;
Property NormalZ:double;
```

These let you retrieve surface's normal. Note that you can't set surface's normal, it's calculated from surface's data by .Update() method.

```
Property UScale:double;
Property VScale:double;
```

These properties replace old .TXScale property. .TXScale is still supported for compatibility.

```
TOLELight:
 procedure Release;
 procedure Update;
```

Nothing much here. Just something that was missed in 0.8.

The new class was added in 0.81 - TOLECOG. It lets you interface with placed COGs of the level.

```
TOLECOG=class(TAutoObject)
   Procedure Release;
   Procedure Update;

   Function GetValue(n:Integer):string;
   Function SetValue(n:Integer;const val:string):WordBool;

   Function GetValueName(n:Integer):String;
   Function GetValueType(n:Integer):String;

   Procedure SetValueType(n:Integer;vtype:string);
   Procedure SetValueName(n:Integer;name:string);

   Function AddValue(const name,val,vtype:string):integer;
   Procedure InsertValue(n:Integer;const name,val,vtype:string);
   Procedure DeleteValue(n:integer);

   Property NValues:Integer;
   Property FileName:string;
  end;
```

```
Procedure Release;
Procedure Update;
```

These do the same as for all other objects.

```
Function GetValue(n:Integer):string;
Function SetValue(n:Integer;const val:string):WordBool;

Function GetValueName(n:Integer):String;
Function GetValueType(n:Integer):String;

Procedure SetValueType(n:Integer;vtype:string);
Procedure SetValueName(n:Integer;name:string);

Function AddValue(const name,val,vtype:string):integer;
Procedure InsertValue(n:Integer;const name,val,vtype:string);
Procedure DeleteValue(n:integer);

Property NValues:Integer;
```

This set of methods lets you alter COG's values. The most used methods would apparently be GetValue() and SetValue(). They accept values in the same form as you type them in Placed COG editor. Although you shouldn't normally alter placed COG's value list, their names and types, the correspodning methods are provided just in case.

```
Property FileName:string;
```

This is the name of the COG file. You shouldn't alter it either, but you can.

New in JED 0.85:

```
TOLESurface=class(TAutoObject)
   Property Nvertices:Integer;
```

```
Property Nvertices:Integer;
```

This is simply the number of vertices in a surface. i can't believe I forgot to add this at the very beginning.

```
TJEDApp=class(TAutoObject)
    Function EditSectorFlags(flags:LongInt):LongInt;
    Function EditSurfaceFlags(flags:LongInt):LongInt;
    Function EditAdjoinFlags(flags:LongInt):LongInt;

    Function EditThingFlags(flags:LongInt):LongInt;
```

```
    Function EditFaceFlags(flags:LongInt):LongInt;
    Function EditLightFlags(flags:Longint):LongInt;

    Function PickGeo(geo:integer):integer;
    Function PickLightMode(lmode:integer):integer;
    Function PickTEX(tex:integer):integer;

    Procedure StartUndo(const name:string);
    Procedure SaveUndoForThing(n:integer;change:integer);
    Procedure SaveUndoForLight(n:integer;change:integer);
    Procedure SaveUndoForSector(n:integer;change:integer;whatpart:integer);
    Procedure ClearUndoBuffer;
    Procedure ApplyUndo;
    Function GetJEDSetting(const name:string):variant;
    Function IsLayerVisible(n:integer):WordBool;
    Property LevelFile:String;
    Property Templates:variant;
```

```
Function EditSectorFlags(flags:LongInt):LongInt;
Function EditSurfaceFlags(flags:LongInt):LongInt;
Function EditAdjoinFlags(flags:LongInt):LongInt;

Function EditThingFlags(flags:LongInt):LongInt;
Function EditFaceFlags(flags:LongInt):LongInt;
Function EditLightFlags(flags:Longint):LongInt;

Function PickGeo(geo:integer):integer;
Function PickLightMode(lmode:integer):integer;
Function PickTEX(tex:integer):integer;
```

These are the fucntions to invoke the corresponding pickers for values JED uses.

```
Procedure StartUndo(const name:string);
Procedure SaveUndoForThing(n:integer;change:integer);
Procedure SaveUndoForLight(n:integer;change:integer);
Procedure SaveUndoForSector(n:integer;change:integer;whatpart:integer);
Procedure ClearUndoBuffer;
Procedure ApplyUndo;
```

These are the functions to control the Undo. StartUndo() begins an undo record. The name parameter is what will appear in Undo XXXX menu. SaveUndoForThing() saves the undo information for thing n into the current undo record (you can save undo information for several things/sectors/lights into one undo record). Parameter "change" is 0 if the thing is about to be changed, 1 if the thing was just added and 2 if it's about to be deleted. SaveUndoForLight() does the same for light. SaveUndoForSector has one additional parameter - whatpart. It's 1 if it's sector's vertices/surface composition/etc. is about to change, 2 if it's merely sector/surface attributes, 3 if both. Note that lighting values on vertices are not considered to be sector/surface attributes, so you need to specify 1 for "whatpart" if you're changing them. Also note that if you're changing a surface you need to save undo information for the entire sector it belongs to. ClearUndoBuffer procedure clears ALL undo records. ApplyUndo applies the last undo record.

```
Function GetJEDSetting(const name:string):variant;
```

This functions retrieves the JED setting by its name. The settings are those you can find in JED's registry key. I.e. - "JKDir", "CDDir" and such. This is in case you need it.

```
Function IsLayerVisible(n:integer):WordBool;
```

This function lets you check if a specific layer in currently visble in JED.

```
Property LevelFile:String;
```

This property returns a name of the current loaded file name.

```
Property Templates:variant;
```

This one return TOLETemplates object that lets you access templates.

```
TOLELevel=class(TAutoObject)
   Property NLayers:Integer;
```

This property returns a number of layers in the level.

```
  TOLETemplates=class(TAutoObject)
   Procedure ClearTemplates;
   Function NTemplates:integer;
   Function GetTemplateName(n:integer):String;
```

```
   Function GetTemplateParent(n:integer):string;
   Function GetTemplateValues(n:integer):String;

   Function GetTemplateDescription(n:integer):String;
   Procedure SetTemplateDescription(n:integer;const desc:string);
   Procedure GetTemplateBBox(n:integer;var x1,y1,z1,x2,y2,z2:double);
   Function GetTemplateBBoxEx(n:integer;what:integer):double;
   Procedure SetTemplateBBox(n:integer;x1,y1,z1,x2,y2,z2:double);

   Procedure DeleteTemplate(n:integer);
   Function AddTemplate(const name,parent,values:string):integer;
   Function FindTemplate(const name:string):integer;
   Procedure LoadTemplates(const filename:string);
   Procedure SaveTemplates(const filename:string);
  end;
```

This is the object returned by TJEDApp.Templates property. Most of its methods are pretty straighforward. A few details - "values" string is a space delimited list of template's values. I.e. "size=0.1 model3d=a.3do", etc. FindTemplate() returns -1 if the template isn't found and an index of the target template otherwise. GetTemplateBBoxEx() lets you retrieve template's bounding box by one value at a time. "What" parameter is 1 to 6 for x1 to z1 correspondingly. You can't change a template directly (there's no SetTemplateXXX() methods for template's internal values). if you want to change a template, you should delete the old one and add a new one. SaveTemplate() and LoadTemplates() methods let you save and load template to/from file. Note that you should save templates after you've modified them or the changes will be lost next time a new level is loaded in JED. Important - DO NOT save templates to the .TPL files in JEDDATA directory. Instead, save them to project directory.

New in JED 0.9:

```
TOLESector=class(TAutoObject)
   Function IsConvex:WordBool;
```

Function IsConvex:WordBool;

Returns true if sector is convex and false otherwise.

```
TOLESurface=class(TAutoObject)
   Function IsConvex:Wordbool;
   Function IsPlanar:Wordbool;
```

Function IsConvex:Wordbool;

Returns true if surface is convex and false otherwise.

Function IsPlanar:Wordbool;

Returns true if surface is planar and false otherwise.

```
TJEDApp=class(TAutoObject)
    procedure CheckConsistencyErrors;
    procedure CheckResources;
    Function NConsistencyErrors:integer;
    Function GetConsErrorString(n:integer):String;
    Function GetConsErrorType(n:integer):integer;
    Function GetConsErrorSector(n:integer):integer;
    Function GetConsErrorSurface(n:integer):integer;
    Function GetConsErrorThing(n:integer):integer;
    Function GetConsErrorCog(n:integer):integer;
```

procedure CheckConsistencyErrors;
procedure CheckResources;

Perform a consistency check of missing resource check. You should call one of those before getting consistency errors with GetConsErrorXXX functions.

Function NConsistencyErrors:integer;
Function GetConsErrorString(n:integer):String;
Function GetConsErrorType(n:integer):integer;
Function GetConsErrorSector(n:integer):integer;
Function GetConsErrorSurface(n:integer):integer;
Function GetConsErrorThing(n:integer):integer;
Function GetConsErrorCog(n:integer):integer;

These functions let you get the information about consistency errors. NConsistencyErrors returns the number of errors. GetConsErrorString() returns the text shown in listbox for this error. GetConsErrorType() returns the type of item that exhibited the error:
0 - Error is not tied to any particular item.
1 - error in a sector
2 - error in a surface
3 - error in a thing
4 - error in a COG
Teh rest GetConsErrorXXX() let you retrieve the reference to the errorneous object associate with the consistency error. You shoud check the error type before getting the reference. For surfaces the complete reference is formed by a pair of values returnedf by GetConsErrorSector() and GetConsErrorSurface().

That's it for now. If I remember anything else, I'll be sure to add it. And, of course, I'll update this document when new version of JED with new functions added is released.
Alex.