DNV Software Products : Parametric naming

This page last changed on okt 29, 2008 by peder.

```
//
//
//  GeniE Scripting Example
//
//  This example shows a proposal for a naming scheme and how this can be used
//  to create point objects on an ordered topological pattern
//  Further it shows how you can use these point objects to create beams
//
//  The last section shows how you can create a function to split a beam into segments and set beam
 offset.
//  This example is taken from a real case of modeling a frame with about 200 joints of rather
 irregular shape.
//  Being able to define offsets and end reinforcements in a ordered tabulated form was much
//  more efficient and safer that having to do it interactively through the GUI.
//
//  Starting by creating some sections and a material
//
Box2000 = BoxSection(2, 1, 0.02, 0.02);
Box1800 = BoxSection(1.8, 1, 0.02, 0.02);
Box1600 = BoxSection(1.6, 1, 0.02, 0.02);
Box1400 = BoxSection(1.4, 1, 0.02, 0.02);
Box1200 = BoxSection(1.2, 0.6, 0.02, 0.02);
Box1000 = BoxSection(1., 0.5, 0.02, 0.02);
Box800 = BoxSection(0.8, 0.4, 0.02, 0.02);
Box600 = BoxSection(0.8, 0.3, 0.02, 0.02);
//
S275 = Material(255E6, 7.85E3, 2.1E11, 0.3, 1.2E-5, 0.03);
//
// Introducing the naming pattern
// Object names are built by concatenating four strings; Nam1, Nam2, Nam3 and Nam4
//
// First part denotes object type, second part denotes topology in X-dir
// third part denotes Y-dir and fourth part denotes Z-dir
//
// By putting the actual name strings in arrays and referring to the array elements you may change
 the actual names
// without changing the routines using these names.
// You use the array elements to make names like:
// MyBeam.name = Nam1[0] +Nam2[i] +Nam3[j] +Nam4[k] ;
//
var Nam1 = new Array();
var Nam2 = new Array();
var Nam3 = new Array();
var Nam4 = new Array();
//
// First part denotes type
Nam1[0] = "P";   // Point
Nam1[1] = "Bx";  // Beam in Xdir
Nam1[2] = "By";  // Beam in Ydir
Nam1[3] = "Bz";  // Beam in Zdir
Nam1[4] = "Dx";  // Beam diagonal in XZdir
Nam1[5] = "Dy";  // Beam diagonal in YZdir
//
// Second part denotes topology in X-dir
Nam2[0] = "1";
Nam2[1] = "2";
Nam2[2] = "3";
Nam2[3] = "4";
Nam2[4] = "5";
Nam2[5] = "6";
Nam2[6] = "7";
```

```
Nam2[7] = "8";
Nam2[8] = "9";
Nam2[9] = "10";
// Third part denotes topology in Y-dir
Nam3[0] = "A";
Nam3[1] = "B";
Nam3[2] = "C";
Nam3[3] = "D";
Nam3[4] = "E";
Nam3[5] = "F";
Nam3[6] = "G";
Nam3[7] = "H";
Nam3[8] = "I";
Nam3[9] = "J";
// Forth part denotes topology in Z-dir
Nam4[0] = "1";
Nam4[1] = "2";
Nam4[2] = "3";
Nam4[3] = "4";
Nam4[4] = "5";
Nam4[5] = "6";
Nam4[6] = "7";
Nam4[7] = "8";
Nam4[8] = "9";
Nam4[9] = "10";
//
// By using these arrays of name strings you can easily name objects according to drawing axis etc.
// Just remember to use legal names like A1, B_2 ( not: A3+200mm etc.)
//
//  Make a function to get a name based on four selected name array elements
//  and another function to get a named object by the same four elements
//
 function GetName (N1,N2,N3,N4) { return(Nam1[N1] +Nam2[N2] +Nam3[N3] +Nam4[N4]); }
 function GetObjectByName (N1,N2,N3,N4) { return(GetNamedObject(GetName(N1,N2,N3,N4))); }
//
// An example:
// Print("The name is " + GetName(2,3,4,5));
//
// Now use this naming pattern to create some points and beams.
// Make a function: MakeBeams (NumX, NumY, NumZ, Step) that creates a space of
// NumX x NumY x NumZ points spaced at a distance being a multiple of "Step"
// Then the function creates some beams between these points
//
//
function MakeBeams (NumX, NumY, NumZ, Step)
{
        var i, j, k;
        var Pts1 = new Array();
        var Bms1 = new Array();
    //
    // Stepping in X, Y and Z direction
         for (i = 0;i< NumX ; i++)
         {
             for (j = 0;j< NumY; j++)
             {
                   for (k = 0; k < NumZ; k++)
                   {
                //
                         X1 = i * Step;
                         Y1 = j * Step;
                         Z1 = k * 0.6 * Step;
                         Pts1[k] = Point(X1, Y1, Z1);
                         Pts1[k].name = GetName(0,i,j,k);
                    }
              }
         }
```

```
//
//   Now adding some beams
//
    // Stepping in X, Y and Z direction
         for (i = 0;i< NumX-1 ; i++)
         {
             for (j = 0;j< NumY-1 ; j++)
                 {
                      for (k = 0; k < NumZ-1 ; k++)
                      {
                        var Bms1 = new Array();
                        // Beams in X-direction
                        Bms1[i] = Beam(GetObjectByName(0,i,j,k), GetObjectByName(0,i+1,j,k));
                        Bms1[i].name = GetName(1,i,j,k);
                        Bms1[i].section = Box1400;
                        Bms1[i].material = S275;
                        //
                        var Bms1 = new Array();
                        // Beams in Y-direction
                        Bms1[i] = Beam(GetObjectByName(0,i,j,k), GetObjectByName(0,i,j+1,k));
                        Bms1[i].name = GetName(2,i,j,k);
                        Bms1[i].section = Box800;
                        Bms1[i].material = S275;
                        //
                        var Bms1 = new Array();
                        // Beams in Z-direction
                        Bms1[i] = Beam(GetObjectByName(0,i,j,k), GetObjectByName(0,i,j,k+1));
                        Bms1[i].name = GetName(3,i,j,k);
                        Bms1[i].section = Box1000;
                        Bms1[i].material = S275;
                        //
                        var Bms1 = new Array();
                        // Beams in diagonal XZ-direction
                        Bms1[i] = Beam(GetObjectByName(0,i,j,k), GetObjectByName(0,i+1,j,k+1));
                        Bms1[i].name = GetName(4,i,j,k);
                        Bms1[i].section = Box800;
                        Bms1[i].material = S275;
                        //
                       }
                  }
              }
        }
// End of function MakeBeams
//
// Calling MakeBeams
//
MakeBeams(8, 2, 3, 6.5);
//
//
// Function to update beam properties
// This example is taken from modeling of a module support frame
// In this case all joints were reinforced with gusset plates that were modeled as beam end
 segments with an equivalent cross section.
// Further there were alignment offsets of all beams relative to the system lines. Dimensions of
 the gusset plates and offsets were given on the drawings.
// Typically there were groups of 3-4 similar joints across the frame but there was no straight
 forward pattern to copy.
//
// To speed up the modeling and ease the quality assurance a function BeamProp was written to
 update all beams
//
// The function takes the following input parameters:
// Bea          = Name of beam
// End1_Segment = segment representing gusset plate end 1
// End2_Segment = segment representing gusset plate end 2
// End1_Section = equivalent cross section of gusset plate end 1
```

```
// End2_Section = equivalent cross section of gusset plate end 2
// Z_Offset     = alignment offset (local Z-axis)
// Main_Section = cross section of main beam
//
//
function BeamProp (Bea,End1_Segment,End2_Segment,Z_Offset,Main_Section,End1_Section,End2_Section)
  {
    // the beams are stored in an array Bea. NumBeam is the length of that array.
    var NumBeam = Bea.length;
    for (i = 1; i < NumBeam ; i++)  //

       {

        Bea[i].section = Main_Section;

        if ( Z_Offset != 0. m )

           {
           Bea[i].setBeamOffset(Bea[i].localSystem.zVector.normalise()*Z_Offset);

           }

    //

    // End1 Offset and/or End2 offset

    //

        if ( End1_Segment != 0. m)

           { Bea[i].divideSegmentAt(1,(End1_Segment/Bea[i].length()));  // finding the relative
 length and dividing the beam
             Bea[i].SetSegmentSection(1, End1_Section);

           if ( End2_Segment != 0. m)

              { Bea[i].divideSegmentAt(2,((Bea[i].getSegmentLength(2)-End2_Segment)/
Bea[i].getSegmentLength(2)));
                Bea[i].SetSegmentSection(3, End2_Section);
              }

            }

    // Only end 2 offset

            if ( End1_Segment == 0. m )

            {  if ( End2_Segment != 0. m)

                  { Bea[i].divideSegmentAt(1,((Bea[i].length()- End2_Segment)/Bea[i].length()));

                    Bea[i].SetSegmentSection(2, End2_Section);

                  }

            }

        }
   }
//   End function  BeamProp
//

// You may also extract e.g. a beams cross section like Bx3A1.section.height() and use this to
 calculate an offset or eccentricity
//
//  Now you can use this function to update the beams.
```

```
//  List a group of beams that have the same gussets and offsets and call BeamProp to set the
 offsets end reinforcements
//
//=============================================================================================================
//
//   Beams
Bea = new Array();
Bea[1] = Bx1A1;
Bea[2] = Bx5A1;
Bea[3] = Bx2A2;
Bea[4] = Bx3A2;
Bea[5] = Dx1A1;
//
//  Properties
//
End1_Segment = 0.750 m;
End2_Segment = 1.118 m;
Z_Offset     = -0.314 m;
Main_Section = Box1000;
End1_Section = Box2000;
End2_Section = Box1800;
//
BeamProp(Bea,End1_Segment,End2_Segment,Z_Offset,Main_Section,End1_Section,End2_Section);
//
//  Then next group
//
//=============================================================================================================
//
//   Beams
Bea = new Array();
Bea[1] = Bx2A1;
Bea[2] = Bx4A1;
Bea[3] = Bx1A2;
Bea[4] = Bx5A2;
Bea[5] = Dx4A1;
//
//  Properties
//
End1_Segment = 0.880 m;
End2_Segment = 0.600 m;
Z_Offset     = 0.214 m;
Main_Section = Box1200;
End1_Section = Box1800;
End2_Section = Box1600;
//
BeamProp(Bea,End1_Segment,End2_Segment,Z_Offset,Main_Section,End1_Section,End2_Section);
//
```