```
#region Assembly devDept.Eyeshot.Control.Win.v2023, Version=2023.1.353.0, Culture=neutral,
PublicKeyToken=f3cd437f0d8061b5

// C:\Program Files\devDept Software\Eyeshot
2023\Bin\net472\devDept.Eyeshot.Control.Win.v2023.dll

#endregion


using devDept.Eyeshot.Entities;

using devDept.Geometry;

using devDept.Graphics;

using devDept.Serialization;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Windows.Forms;


namespace devDept.Eyeshot.Control

{

   //

   // Summary:

   //     Design control definition.

   [Description("Provides the ability to import or create geometry.")]

   [Designer("devDept.Eyeshot.Designer.DesignControlDesigner,
devDept.Eyeshot.Control.Win.v2023.Design")]

   [ToolboxBitmap(typeof(Design))]

   [ToolboxItem(true)]

   public class Design : Workspace, IDesign, IWorkspace

   {

      public Design();
```

```
//
// Summary:
//     Light 8 attribute.
[Category("Workspace - Lighting")]
[Description("Light 8 attributes.")]
public LightSettings Light8 { get; set; }
//
// Summary:
//     Light 7 attribute.
[Category("Workspace - Lighting")]
[Description("Light 7 attributes.")]
public LightSettings Light7 { get; set; }
//
// Summary:
//     Light 6 attribute.
[Category("Workspace - Lighting")]
[Description("Light 6 attributes.")]
public LightSettings Light6 { get; set; }
//
// Summary:
//     Light 5 attribute.
[Category("Workspace - Lighting")]
[Description("Light 5 attributes.")]
public LightSettings Light5 { get; set; }
//
// Summary:
//     Light 4 attribute.
[Category("Workspace - Lighting")]
[Description("Light 4 attributes.")]
public LightSettings Light4 { get; set; }
//
```

```
// Summary:
//     Light 3 attribute.
[Category("Workspace - Lighting")]
[Description("Light 3 attributes.")]
public LightSettings Light3 { get; set; }
//
// Summary:
//     Light 2 attribute.
[Category("Workspace - Lighting")]
[Description("Light 2 attributes.")]
public LightSettings Light2 { get; set; }
//
// Summary:
//     Light 1 attribute.
[Category("Workspace - Lighting")]
[Description("Light 1 attributes.")]
public LightSettings Light1 { get; set; }
//
// Summary:
//     Gets or sets the viewports configuration.
//
// Remarks:
//     devDept.Eyeshot.Control.Design.Viewports collection needs to be initialized with
//     the correct number of Viewport items before changing this property.
[Category("Workspace - Viewports")]
[Description("Viewports configuration.")]
public virtual viewportLayoutType LayoutMode { get; set; }
//
// Summary:
//     Gets or sets the ambient light that is always added to the scene (multiplied
//     by the material's ambient component).
```

```
[Category("Workspace - Lighting")]
[Description("Ambient light intensity.")]
public Color AmbientLight { get; set; }
//
// Summary:
//     The minimum acceptable framerate for dynamic movements.
[Category("Workspace - Performance")]
[Description("The minimum acceptable framerate for dynamic movements.")]
public int MinimumFramerate { get; set; }
//
// Summary:
//     Sets the Hidden Lines options.
[Category("Workspace - Display Settings")]
[Description("Hidden Lines settings.")]
public HiddenLinesSettings HiddenLines { get; set; }
//
// Summary:
//     Gets or sets the display settings for Rendered mode, shared by all viewports.
//
// Remarks:
//     Call Design.CompileUserInterfaceElements()
[Category("Workspace - Display Settings")]
[Description("Display Settings for Rendered mode, shared by all viewports.")]
public DisplayModeSettingsRendered Rendered { get; set; }
//
// Summary:
//     Gets or sets the display settings for Flat mode, shared by all viewports.
[Category("Workspace - Display Settings")]
[Description("Display Settings for Flat mode, shared by all viewports.")]
public DisplayModeSettingsFlat Flat { get; set; }
//
```

```
        // Summary:
        //     Gets or sets the display settings for Shaded mode, shared by all viewports.
        [Category("Workspace - Display Settings")]
        [Description("Display Settings for Shaded mode, shared by all viewports.")]
        public DisplayModeSettingsShaded Shaded { get; set; }
        //
        // Summary:
        //     Gets or sets the display settings for Wireframe mode, shared by all viewports.
        [Category("Workspace - Display Settings")]
        [Description("Display Settings for Wireframe mode, shared by all viewports.")]
        public DisplayModeSettings Wireframe { get; set; }
        //
        // Summary:
        //     Gets or sets the convexHull type used as optimization during entities zoom fit.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public zoomFitType ZoomFitMode { get; set; }
        //
        // Summary:
        //     Material collection. This collection contains material definitions.
        //
        // Exceptions:
        //   T:devDept.EyeshotException:
        //     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Design.Document.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public MaterialKeyedCollection Materials { get; set; }
        //
        // Summary:
        //     Gets or sets the gap between viewports.
```

```
[Category("Workspace - Viewports")]
[Description("Gap between Viewports in pixels.")]
public int ViewportsGap { get; set; }
//
// Summary:
//     Gets or sets the active units of measurement.
//
// Remarks:
//     Affects the unit system of exported models.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
[Obsolete("Use CurrentBlock.Units instead.")]
public linearUnitsType Units { get; set; }
//
// Summary:
//     Bounding box settings.
//
// Remarks:
//     You need to call devDept.Eyeshot.Control.Workspace.CompileUserInterfaceElements
//     after assigning a new BoundingBox.
[Category("Workspace")]
[Description("Bounding box settings.")]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
public BoundingBoxSettings BoundingBox { get; set; }
//
// Summary:
//     Gets or sets the default material attributes used by the entities without their
//     own material.
//
// Remarks:
//     Only devDept.Eyeshot.Material.Ambient, devDept.Eyeshot.Material.Specular and
```

```
        //     devDept.Eyeshot.Material.Shininess are used.
        [Category("Workspace")]
        [Description("Default material attributes used by the entities without their own material.")]
        public Material DefaultMaterial { get; set; }
        //
        // Summary:
        //     Gets or sets the distance between the ground plane and the design's bounding
        //     box expressed as a fraction of the design height.
        //
        // Remarks:
        //     This distance is use to compute the plane used by the planar reflections and
        //     the planar shadows
        [Category("Workspace")]
        [Description("Distance between the ground plane and the design's bounding box expressed as a
fraction of the design height (range 0-1).")]
        [TypeConverter(typeof(OpacityConverter))]
        public double GroundPlaneDistance { get; set; }
        //
        // Summary:
        //     Gets or sets the planar shadow's opacity.
        [Category("Workspace")]
        [Description("Planar shadow's opacity (range 0-1).")]
        [TypeConverter(typeof(OpacityConverter))]
        public double PlanarShadowOpacity { get; set; }
        //
        // Summary:
        //     Gets or sets the animation step that increments the
devDept.Eyeshot.Control.Design.AnimationFrameNumber
        //     at each tick of the animation timer. Can be a negative number.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public int AnimationStep { get; set; }
```

```
        //
        // Summary:
        //     Tells if the animation is currently running.
        //
        // Remarks:
        //     During animation the silhouettes, if enabled, are skipped.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool IsAnimationRunning { get; }
        //
        // Summary:
        //     The devDept.Eyeshot.SketchManager for devDept.Eyeshot.Entities.SketchEntity editing.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public SketchManager SketchManager { get; }
        //
        // Summary:
        //     When true, frozen entities are drawn in the Z buffer.
        //
        // Remarks:
        //     See
devDept.Eyeshot.Control.Workspace.SetCurrent(devDept.Eyeshot.Entities.BlockReference,System.Bo
olean)
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool WriteDepthForFrozen { get; set; }
        //
        // Summary:
        //     When true, adds an extra pass to write transparent entities in the Z buffer.
        //
        // Remarks:
        //     Refers to inaccurate transparency only.
```

```
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool WriteDepthForTransparents { get; set; }
//
// Summary:
//     Gets or sets the angle (in radians) used to discriminate adjacent triangles in
//     the selection of the faces of Mesh and Solids.
//
// Remarks:
//     The default is 0.5 rad.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public double FaceSelectionAngle { get; set; }
//
// Summary:
//     WinForms: When the viewport control is hosted inside a SplitContainer control
//     an annoying repaint issue is present during SplitContainer splitter moving.
//     WPF: When the viewport control is hosted inside a Grid control with GridSplitter
//     an annoying repaint issue is present during GridSplitter moving.
//     Setting properly this flag it can be avoided.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool SplitterMoving { get; set; }
//
// Summary:
//     Gets or sets the pick filter used by the pick action modes.
[Category("Workspace - Selection")]
public selectionFilterType SelectionFilterMode { get; set; }
//
// Summary:
//     Gets or sets a boolean that determines whether the selection goes inside the
```

```
        //     devDept.Eyeshot.Entities.BlockReference tree.
        [Category("Workspace - Selection")]
        public assemblySelectionType AssemblySelectionMode { get; set; }
        //
        // Summary:
        //     Gets or sets the coordinate system orientation mode.
        [Category("Workspace")]
        [Description("Coordinate system orientation mode.")]
        public orientationType OrientationMode { get; set; }
        //
        // Summary:
        //     Gets or sets a flag that keeps the scene upright when doing SetView operations
        //     or clicking on the devDept.Eyeshot.Control.ViewCubeIcon faces.
        //
        // Remarks:
        //     If false, the SetView operations and the mouse clicks on the ViewCubeIcon set
        //     the orientation with the up-vector closest to the current one.
        [Category("Workspace")]
        [Description("If false, the SetView operations set the orientation with the up-vector closest to
the current one.")]
        public bool KeepSceneUpright { get; set; }
        //
        // Summary:
        //     Gets or sets the backface settings, shared by all viewports. This settings don't
        //     apply to multicolor entities like devDept.Eyeshot.Entities.Mesh for example.
        [Category("Workspace")]
        [Description("Backface color, shared by all viewports.")]
        public BackfaceSettings Backface { get; set; }
        //
        // Summary:
        //     Gets or sets the default color used by top level entities with
devDept.Eyeshot.Entities.colorMethodType.byParent
```

```
//     color method.
[Category("Workspace - Display Settings")]
[Description("Default color for top entities with ByParent ColorMethod.")]
public Color DefaultColor { get; set; }
//
// Summary:
//     Gets or sets the border settings.
[Category("Workspace")]
[Description("Border settings.")]
public BorderSettings ViewportBorder { get; set; }
//
// Summary:
//     The manipulator used to graphically position the selected entities.
[Category("Workspace")]
[Description("Gets or sets the manipulator used to graphically position the selected entities.")]
public ObjectManipulator ObjectManipulator { get; set; }
//
// Summary:
//     Clipping plane 6 attributes.
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 6 attributes.")]
public ClippingPlane ClippingPlane6 { get; set; }
//
// Summary:
//     Clipping plane 5 attributes.
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 5 attributes.")]
public ClippingPlane ClippingPlane5 { get; set; }
//
// Summary:
//     Clipping plane 4 attributes.
```

```
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 4 attributes.")]
public ClippingPlane ClippingPlane4 { get; set; }
//
// Summary:
//     Clipping plane 3 attributes.
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 3 attributes.")]
public ClippingPlane ClippingPlane3 { get; set; }
//
// Summary:
//     Clipping plane 2 attributes.
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 2 attributes.")]
public ClippingPlane ClippingPlane2 { get; set; }
//
// Summary:
//     Clipping plane 1 attributes.
[Category("Workspace - Clipping planes")]
[Description("Clipping plane 1 attributes.")]
public ClippingPlane ClippingPlane1 { get; set; }
//
// Summary:
//     Gets the active viewport.
//
// Exceptions:
//   T:System.IndexOutOfRangeException:
//     Throw when devDept.Eyeshot.Control.Design.Viewports have not been initialized.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public Viewport ActiveViewport { get; }
```

```
        //
        // Summary:
        //     Gets or sets the active viewport index.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public int ActiveViewportIndex { get; set; }
        //
        // Summary:
        //     Gets or sets the list of viewports.
        [Category("Workspace - Viewports")]
        [Description("Viewports.")]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Content)]
        [Editor("devDept.Eyeshot.Designer.Converters.ViewportCollectionEditor",
"System.Drawing.Design.UITypeEditor")]
        [NotifyParentProperty(true)]
        public ViewportList Viewports { get; set; }
        //
        // Summary:
        //     Gets the devDept.Eyeshot.DesignDocument.
        public DesignDocument Document { get; }
        //
        // Summary:
        //     Gets or sets the animation interval.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public int AnimationInterval { get; set; }
        //
        // Summary:
        //     Gets or sets the animation frame number.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
```

```
        public int AnimationFrameNumber { get; set; }


        //
        // Summary:
        //     Rendered mat changer.
        //
        // Parameters:
        //   nature:
        //     The nature of the entity just to be drawn
        //
        //   material:
        //     The entity material
        //
        //   selected:
        //     The selection status
        //
        //   data:
        //     Data for high quality rendering
        public static void SetColorRendered(RenderContextBase context, entityNatureType nature,
Material material, bool selected, RenderParams data);
        //
        // Summary:
        //     Shaded color changer.
        //
        // Parameters:
        //   context:
        //     The render context
        //
        //   nature:
        //     The nature of the entity just to be drawn
        //
```

```
// color:
//   The color
//
// selected:
//   The selection status
//
// backface:
//   The BackFace settings
public static void SetColorShaded(RenderContextBase context, entityNatureType nature, Color color, bool selected, BackfaceSettings backface);
public override void Clear();
//
// Summary:
//   Creates the data for the Bill Of Materials.
//
// Parameters:
// entities:
//   The entities collection.
//
// blocks:
//   The blocks collection.
//
// partsOnly:
//   If true, only the block containing geometry are taken in account to create the
//   table.
//
// maxLevel:
//   The maximum depth level of the Bill of Materials.
//
// Returns:
//   A DataTable containing the Bill of Materials data.
```

```
public DataTable CreateBillOfMaterials(EntityList entities, BlockKeyedCollection blocks, bool partsOnly, int maxLevel = int.MaxValue);

//
// Summary:
//     Creates the Bill Of Materials.
//
// Parameters:
//   entites:
//     The entities collection.
//
//   blocks:
//     The blocks collection.
//
//   itemNumberText:
//     The text on the header row of the item number column.
//
//   partNumberText:
//     The text on the header row of the part number column.
//
//   descriptionText:
//     The text on the header raw of the description column.
//
//   quantityText:
//     The text on the header raw of the quantity column.
//
//   partsOnly:
//     If true, only the block containing geometry are taken in account to create the
//     table.
//
//   flowDirection:
//     The table's flow direction.
```

```
        //
        //   maxLevel:
        //     The maximum depth level of the Bill of Materials.
        //
        // Returns:
        //     Returns a Table entity that represents the Bill of Materials.
        public Table CreateBOMTable(EntityList entites, BlockKeyedCollection blocks, string
itemNumberText, string partNumberText, string descriptionText, string quantityText, bool partsOnly =
true, Table.flowDirection flowDirection = Table.flowDirection.Down, int maxLevel = int.MaxValue);
        //
        // Summary:
        //     Sets the sketch entity for editing.
        //
        // Parameters:
        //   sketchEntity:
        //     The sketch.
        public void EditSketch(SketchEntity sketchEntity);
        //
        // Summary:
        //     Extrudes a text creating an array of new meshes.
        //
        // Parameters:
        //   text:
        //     Entity text
        //
        //   tolerance:
        //     Entity text needs to be converted to Mesh type, this is the conversion tolerance
        //     (chordal error).
        //
        //   amount:
        //     Extrusion amount
        //
```

```
//   endCaps:
//     Closing Caps on both ends
//
//   meshNature:
//     Nature of the resulting mesh entity
//
// Returns:
//     The created array of meshes.
public Mesh[] ExtrudeText(Text text, double tolerance, double amount, bool endCaps,
Mesh.natureType meshNature);
//
// Summary:
//     Extrudes a text creating an array of new solids.
//
// Parameters:
//   text:
//     Entity text
//
//   tolerance:
//     Entity text needs to be converted to Solid type, this is the conversion tolerance
//     (chordal error).
//
//   amount:
//     Extrusion amount
//
//   endCaps:
//     Closing Caps on both ends
//
// Returns:
//     The created array of solids.
public Solid[] ExtrudeText(Text text, double tolerance, Vector3D amount, bool endCaps);
```

```
//
// Summary:
//     Extrudes a text creating an array of new solids.
//
// Parameters:
//   text:
//     Entity text
//
//   tolerance:
//     Entity text needs to be converted to Solid type, this is the conversion tolerance
//     (chordal error).
//
//   amount:
//     Extrusion amount
//
//   endCaps:
//     Closing Caps on both ends
//
// Returns:
//     The created array of solids.
public Solid[] ExtrudeText(Text text, double tolerance, double amount, bool endCaps);
//
// Summary:
//     Extrudes a text creating an array of new meshes.
//
// Parameters:
//   text:
//     Entity text
//
//   tolerance:
//     Entity text needs to be converted to Mesh type, this is the conversion tolerance
```

```
//     (chordal error).
//
//   amount:
//     Extrusion amount
//
//   endCaps:
//     Closing Caps on both ends
//
//   meshNature:
//     Nature of the resulting mesh entity
//
// Returns:
//     The created array of meshes.
    public Mesh[] ExtrudeText(Text text, double tolerance, Vector3D amount, bool endCaps,
Mesh.natureType meshNature);
//
// Summary:
//     Finds the closest triangle to the viewer of the specified item (which must be
//     related to an devDept.Eyeshot.Entities.IFace entity and may include the parents
//     stack for nested entities).
//
// Parameters:
//   item:
//     The item
//
//   mousePos:
//     Mouse cursor position (zero on top)
//
// Returns:
//     A sorted (by distance from the viewer) list of devDept.Eyeshot.Entities.HitTriangle
//     found under the mouse cursor.
```

```
        //
        // Remarks:
        //     The triangles intersection points are in the devDept.Eyeshot.Control.Workspace.Parents
        //     coordinate system.
        public IList<HitTriangle> FindClosestTriangle(SelectedItem item, System.Drawing.Point
mousePos);
        //
        // Summary:
        //     Finds the closest triangle to the viewer of the specified entity.
        //
        // Parameters:
        //   entity:
        //     The entity
        //
        //   mousePos:
        //     Mouse cursor position (zero on top)
        //
        // Returns:
        //     A sorted (by distance from the viewer) list of devDept.Eyeshot.Entities.HitTriangle
        //     found under the mouse cursor.
        //
        // Remarks:
        //     If the entity is inside a devDept.Eyeshot.Entities.BlockReference, it must be
        //     set as current.
        public IList<HitTriangle> FindClosestTriangle(IFace entity, System.Drawing.Point mousePos);
        //
        // Summary:
        //     Finds the closest triangle to the viewer of the specified item (which must be
        //     related to an devDept.Eyeshot.Entities.IFace entity and may include the parents
        //     stack for nested entities).
        //
```

```
// Parameters:
//   item:
//     The item
//
//   mousePos:
//     Mouse cursor position
//
//   hitPoint:
//     Closest triangle intersection point in the devDept.Eyeshot.Control.Workspace.Parents
//     coordinate system.
//
//   hitTriangleIndex:
//     Closest triangle index
//
// Returns:
//     The numbers of triangles found under the mouse cursor.
public int FindClosestTriangle(SelectedItem item, System.Drawing.Point mousePos, out Point3D hitPoint, out int hitTriangleIndex);
//
// Summary:
//     Finds the closest triangle to the viewer of the specified entity.
//
// Parameters:
//   entity:
//     The entity
//
//   mousePos:
//     Mouse cursor position
//
//   hitPoint:
//     Closest triangle intersection point in the devDept.Eyeshot.Control.Workspace.Parents
```

```
//     coordinate system.
//
//   hitTriangleIndex:
//     Closest triangle index
//
// Returns:
//     The numbers of triangles found under the mouse cursor.
      public int FindClosestTriangle(IFace entity, System.Drawing.Point mousePos, out Point3D
hitPoint, out int hitTriangleIndex);
//
// Summary:
//     For internal use only.
      public viewportLayoutType GetDefaultLayoutMode();
//
// Summary:
//     Gets the plane used to draw the planar reflections.
//
// Returns:
//     The plane
      public Plane GetPlanarReflectionsPlane();
//
// Summary:
//     Loads a devDept.Eyeshot.DesignDocument to the control.
//
// Parameters:
//   designDocument:
//     The devDept.Eyeshot.DesignDocument.
//
// Exceptions:
//   T:devDept.EyeshotException:
//     Thrown when trying to set a devDept.Eyeshot.Document already linked to another
```

```
//     devDept.Eyeshot.Control.Workspace.
public void LoadDocument(DesignDocument designDocument);
//
// Summary:
//     Moves a list of entities from plane XY to the specified plane.
//
// Parameters:
//   entList:
//     The entity list
//
//   plane:
//     Destination plane
public void MoveToPlane(ICollection<Entity> entList, Plane plane);
//
// Summary:
//     Moves a list of points from plane XY to the specified plane.
//
// Parameters:
//   points:
//     The points list
//
//   plane:
//     Destination plane
public void MoveToPlane(IList<Point3D> points, Plane plane);
//
// Summary:
//     Moves a list of entities from plane XY to the specified plane.
//
// Parameters:
//   entList:
//     The entity list
```

```
        //
        //   origin:
        //     The destination plane origin
        //
        //   xAxis:
        //     The destination plane X axis
        //
        //   yAxis:
        //     The destination plane Y axis
        public void MoveToPlane(ICollection<Entity> entList, Point3D origin, Point3D xAxis, Point3D
yAxis);
        //
        // Summary:
        //     Sets the sketch entity for editing.
        //
        // Parameters:
        //   sketchPlane:
        //     The sketch plane.
        public void NewSketch(Plane sketchPlane);
        //
        // Summary:
        //     Creates a new sketch on the Brep face.
        //
        // Parameters:
        //   faceIndex:
        //     The brep face index.
        //
        //   brep:
        //     The brep.
        public bool NewSketch(int faceIndex, Brep brep);
        //
```

```
// Summary:
//     Restores the scene synchronously from disk, including entities and all the master
//     collections (layers, blocks, etc.)
//
// Parameters:
//   fileName:
//     File name.
//
//   drawing:
//     The drawing control. Can be null/Nothing.
//
//   fileSerializer:
//     The devDept.Serialization.FileSerializer. Can be null/Nothing.
//
// Remarks:
//     This command is based on Eyeshot proprietary file format. If the fileSerializer
//     param is null, it skips your custom data. If you want to run the operation asynchronously,
//     use the devDept.Eyeshot.Translators.ReadFile class in conjunction with
devDept.Eyeshot.Control.Workspace.StartWork(devDept.WorkUnit)
//     method instead.
public virtual void OpenFile(string fileName, Drawing drawing = null, FileSerializer fileSerializer =
null);
//
// Summary:
//     Restores the scene synchronously from disk, including entities and all the master
//     collections (layers, blocks, etc.)
//
// Parameters:
//   stream:
//     The file stream.
//
//   drawing:
```

```
//     The drawing control. Can be null/Nothing.
//
//   fileSerializer:
//     The devDept.Serialization.FileSerializer. Can be null/Nothing.
//
// Remarks:
//     This command is based on Eyeshot proprietary file format. If the fileSerializer
//     param is null, it skips your custom data. If you want to run the operation asynchronously,
//     use the devDept.Eyeshot.Translators.ReadFile class in conjunction with
devDept.Eyeshot.Control.Workspace.StartWork(devDept.WorkUnit)
//     method.
    public virtual void OpenFile(Stream stream, Drawing drawing = null, FileSerializer fileSerializer =
null);
//
// Summary:
//     Orients the Camera between the two specified points.
//
// Parameters:
//   location:
//     The camera location
//
//   target:
//     The new camera target
//
// Remarks:
//     The parameter is used just to determine the direction. The final
devDept.Eyeshot.Camera.Location
//     depends on the target and the devDept.Eyeshot.Camera.Distance
    public virtual void OrientCamera(Point3D location, Point3D target);
//
// Parameters:
//   extents:
```

```
    //     If true fits the entire model in the viewport area to captures the whole scene,
    //     else uses the current camera
    //
    // Remarks:
    //     For more options use the devDept.Eyeshot.Control.HiddenLinesViewOnPaper class
    //     directly.
    public void Print(bool extents = false);
    //
    // Parameters:
    //   extents:
    //     If true fits the entire model in the viewport area to captures the whole scene,
    //     else uses the current camera
    //
    // Remarks:
    //     For more options use the devDept.Eyeshot.Control.HiddenLinesViewOnPaperPreview
    //     class directly.
    public void PrintPreview(Size printPreviewDlgClientSize, bool extents = false);
    //
    // Summary:
    //     Rotates the view simulating the movement between two mouse position.
    //
    // Parameters:
    //   mousePos1:
    //     The initial mouse position
    //
    //   mousePos2:
    //     The final mouse position
    public virtual void RotateCamera(System.Drawing.Point mousePos1, System.Drawing.Point
mousePos2);
    //
    // Summary:
```

```
//     Rotates the view of the specified amount.
//
// Parameters:
//   dx:
//     Horizontal rotation amount
//
//   dy:
//     Vertical rotation amount
//
//   animate:
//     If true performs an animation when changing the view
public virtual void RotateCamera(int dx, int dy, bool animate);
//
// Summary:
//     Rotates the view of the specified amount.
//
// Parameters:
//   dx:
//     Horizontal rotation amount
//
//   dy:
//     Vertical rotation amount
public virtual void RotateCamera(int dx, int dy);
//
// Summary:
//     Rotates the view of the specified amount.
//
// Parameters:
//   axis:
//     Rotation axis
//
```

```
        //   rotAngleInDegrees:
        //     Rotation amount
        //
        //   trackBall:
        //     if true, the rotation is applied after the current rotation, else it's applied
        //     before
        //
        //   animate:
        //     If true performs an animation when changing the view
        public virtual void RotateCamera(Vector3D axis, double rotAngleInDegrees, bool trackBall, bool
animate);
        //
        // Summary:
        //     Rotates the view of the specified amount.
        //
        // Parameters:
        //   last:
        //     Previous axis
        //
        //   current:
        //     Current axis
        public virtual void RotateCamera(Vector3D last, Vector3D current);
        //
        // Summary:
        //     Rotates the view of the specified amount.
        //
        // Parameters:
        //   last:
        //     Previous axis
        //
        //   current:
```

```
//     Current axis
//
//   animate:
//     If true performs an animation when changing the view
public virtual void RotateCamera(Vector3D last, Vector3D current, bool animate);
//
// Summary:
//     Rotates the view of the specified amount.
//
// Parameters:
//   axis:
//     Rotation axis
//
//   rotAngleInDegrees:
//     Rotation amount
//
//   trackBall:
//     if true, the rotation is applied after the current rotation, else it's applied
//     before
public virtual void RotateCamera(Vector3D axis, double rotAngleInDegrees, bool trackBall);
//
// Summary:
//     Rotates the view downwards.
//
// Parameters:
//   degrees:
//     Degrees of rotation
public void RotateDown(double degrees);
//
// Summary:
//     Rotates the view to the left.
```

```
//
// Parameters:
//   degrees:
//     Degrees of rotation
public void RotateLeft(double degrees);
//
// Summary:
//     Rotates the view to the right.
//
// Parameters:
//   degrees:
//     Degrees of rotation
public void RotateRight(double degrees);
//
// Summary:
//     Rotates the view upwards.
//
// Parameters:
//   degrees:
//     Degrees of rotation
public void RotateUp(double degrees);
//
// Summary:
//     Saves the current scene synchronously on disk, including entities and all the
//     master collections (layers, blocks, etc.)
//
// Parameters:
//   stream:
//     The file stream.
//
//   drawing:
```

```
//     The drawing control. Can be null/Nothing.
//
//   contentType:
//     The devDept.Serialization.contentType to store the file.
//
//   fileSerializer:
//     The custom devDept.Serialization.FileSerializer where are defined custom objects.
//     Can be null/Nothing.
//
// Remarks:
//     This command is based on Eyeshot proprietary file format. If the fileSerializer
//     param is null, it skips your custom data. If you want to run the operation asynchronously,
//     use the devDept.Eyeshot.Translators.WriteFile class in conjunction with
devDept.Eyeshot.Control.Workspace.StartWork(devDept.WorkUnit)
//     method instead.
    public virtual void SaveFile(Stream stream, Drawing drawing = null, contentType contentType =
contentType.GeometryAndTessellation, FileSerializer fileSerializer = null);
//
// Summary:
//     Saves the current scene synchronously on disk, including entities and all the
//     master collections (layers, blocks, etc.)
//
// Parameters:
//   fileName:
//     File name.
//
//   drawing:
//     The drawing control. Can be null/Nothing.
//
//   contentType:
//     The devDept.Serialization.contentType to store the file.
//
```

```
//   fileSerializer:
//     The custom devDept.Serialization.FileSerializer where are defined custom objects.
//     Can be null/Nothing.
//
// Remarks:
//     This command is based on Eyeshot proprietary file format. If the fileSerializer
//     param is null, it skips your custom data. If you want to run the operation asynchronously,
//     use the devDept.Eyeshot.Translators.WriteFile class in conjunction with
devDept.Eyeshot.Control.Workspace.StartWork(devDept.WorkUnit)
//     method.
    public virtual void SaveFile(string fileName, Drawing drawing = null, contentType contentType =
contentType.GeometryAndTessellation, FileSerializer fileSerializer = null);
//
// Summary:
//     Sets the parents stack that represents the starting point for the selection.
//
// Remarks:
//     The top most element of the stack is the inner most BlockReference (the last
//     parent of the hierarchy).
    public void SetSelectionScope(Stack<BlockReference> parents);
//
// Summary:
//     Starts the animation timer. After calling this method the Animate method of all
//     the entities is called periodically to allow the computation of the new entity
//     position.
//     To animate group of entities the approach is the following:
//     1. Create groups of moving objects (using devDept.Eyeshot.Block class and adding
//     entities to the devDept.Eyeshot.Block.Entities collection)
//     2. Add blocks created at point 1) to the Design.Blocks collection
//     3. Subclass the BlockReference class for each moving group of objects
//     4. Override the BlockReference.Animate() method of the class at point 3) and
//     add the code to compute the data for entities transformation at that specific
```

```
//     time frame
//     5. Override the BlockReference.MoveTo() method of the class at point 3) and add
//     the code to move the objects on the GPU
//     6. Override the Entity.IsInFrustum() method of the class at point 3) and return
//     true without calling the base to avoid undesired clipping. It might be necessary
//     to override also the bounding box scene extents
//     7. Add the block references created at point 3) to the Design.Entities collection
//     8. Call StartAnimation() providing the time interval between each frame
//     9. Call StopAnimation() to stop the animation
//
// Remarks:
//     During animation the silhouettes, if enabled, are skipped.
public void StartAnimation();
//
// Summary:
//     Starts the animation timer. After calling this method the Animate method of all
//     the entities is called periodically to allow the computation of the new entity
//     position.
//     To animate group of entities the approach is the following:
//     1. Create groups of moving objects (using devDept.Eyeshot.Block class and adding
//     entities to the devDept.Eyeshot.Block.Entities collection)
//     2. Add blocks created at point 1) to the Design.Blocks collection
//     3. Subclass the BlockReference class for each moving group of objects
//     4. Override the BlockReference.Animate() method of the class at point 3) and
//     add the code to compute the data for entities transformation at that specific
//     time frame
//     5. Override the BlockReference.MoveTo() method of the class at point 3) and add
//     the code to move the objects on the GPU
//     6. Override the BlockReference.IsInFrustum() method of the class at point 3)
//     and return true without calling the base to avoid undesired clipping. It might
//     be necessary to override also the bounding box scene extents
```

```
//     7. Add the block references created at point 3) to the Design.Entities collection
//     8. Call StartAnimation() providing the time interval between each frame
//     9. Call StopAnimation() to stop the animation
//
// Parameters:
//   interval:
//     Timer interval. See System.Threading.Timer object.
//
// Remarks:
//     During animation the silhouettes, if enabled, are skipped.
public void StartAnimation(int interval);
//
// Summary:
//     Starts the animation timer. After calling this method the Animate method of all
//     the entities is called periodically to allow the computation of the new entity
//     position.
//     To animate group of entities the approach is the following:
//     1. Create groups of moving objects (using devDept.Eyeshot.Block class and adding
//     entities to the devDept.Eyeshot.Block.Entities collection)
//     2. Add blocks created at point 1) to the Design.Blocks collection
//     3. Subclass the BlockReference class for each moving group of objects
//     4. Override the BlockReference.Animate() method of the class at point 3) and
//     add the code to compute the data for entities transformation at that specific
//     time frame
//     5. Override the BlockReference.MoveTo() method of the class at point 3) and add
//     the code to move the objects on the GPU
//     6. Override the Entity.IsInFrustum() method of the class at point 3) and return
//     true without calling the base to avoid undesired clipping. It might be necessary
//     to override also the bounding box scene extents
//     7. Add the block references created at point 3) to the Design.Entities collection
//     8. Call StartAnimation() providing the time interval between each frame
```

```
        //     9. Call StopAnimation() to stop the animation
        //
        // Parameters:
        //   stopAfter:
        //     The animation will stop after this number of frames
        //
        //   interval:
        //     Timer interval, see System.Threading.Timer object.
        //
        // Remarks:
        //     During animation the silhouettes, if enabled, are skipped.
        public void StartAnimation(int interval, int stopAfter);
        //
        // Summary:
        //     Stops the animation timer. Call base class method when overriding.
        public virtual void StopAnimation();
        //
        // Summary:
        //     Updates the Viewports dimensions and locations depending on the
devDept.Eyeshot.Control.Design.LayoutMode
        //     property.
        //
        // Remarks:
        //     If overridden, in WPF must call Workspace.ApplyOpacityMask
        public virtual void UpdateViewportsSizeAndLocation();
    }
}
```

```
#region Assembly devDept.Eyeshot.v2023, Version=2023.1.353.0, Culture=neutral,
PublicKeyToken=9440825e8b4733bc
// C:\Program Files\devDept Software\Eyeshot 2023\Bin\net472\devDept.Eyeshot.v2023.dll
#endregion

using devDept.Geometry.Converters;
using devDept.Serialization;
using System.ComponentModel;
using System.IO;
using System.Reflection;
using System.Xml.Linq;

namespace devDept.Geometry
{
   //
   // Summary:
   //     Defines a 3D point.
   [DefaultMember("Item")]
   [TypeConverter(typeof(Point3DConverter))]
   public class Point3D : Point2D
   {
      //
      // Summary:
      //     Z coordinate value.
      public double Z;

      //
      // Summary:
      //     Empty constructor.
      public Point3D();
      //
```

```
// Summary:
//     Double array constructor.
//
// Parameters:
//   coords:
//     The XYZ coordinates as a double array
public Point3D(double[] coords);
//
// Summary:
//     2D point constructor
//
// Parameters:
//   x:
//     X coordinate value
//
//   y:
//     Y coordinate value
public Point3D(double x, double y);
//
// Summary:
//     Standard constructor.
//
// Parameters:
//   x:
//     X coordinate value
//
//   y:
//     Y coordinate value
//
//   z:
//     Z coordinate value
```

```
public Point3D(double x, double y, double z);
protected Point3D(Point3D another);

public double this[int index] { get; }

//
// Summary:
//     Returns the (0,0,0) point.
public static Point3D Origin { get; }
//
// Summary:
//     Returns a 3D point with coordinates at double.MaxValue.
public static Point3D MaxValue { get; }
//
// Summary:
//     Returns a 3D point with coordinates at double.MinValue.
public static Point3D MinValue { get; }
public override double MaximumCoordinate { get; }
[Browsable(false)]
public Vector3D AsVector { get; }

//
// Summary:
//     Compares two 3D points in the given domain.
//
// Parameters:
//   p1:
//     First point
//
//   p2:
//     Second point
```

```
//
//   domainSize:
//     The 3D diagonal length of your model.
//
// Returns:
//     True if the two point are coincident, false otherwise.
public static bool AreEqual(Point3D p1, Point3D p2, double domainSize);
//
// Summary:
//     Computes the distance between two 3D points.
//
// Parameters:
//   a:
//     First 3D point
//
//   b:
//     Second 3D point
//
// Returns:
//     The distance between a and b.
public static double Distance(Point3D a, Point3D b);
//
// Summary:
//     Computes the squared distance between two 3D points.
//
// Parameters:
//   a:
//     First point
//
//   b:
//     Second point
```

```
//
// Returns:
//     The squared distance between a and b.
public static double DistanceSquared(Point3D a, Point3D b);
//
// Summary:
//     Computes the midpoint between two 3D points.
//
// Parameters:
//   a:
//     First 3D point
//
//   b:
//     Second 3D point
//
// Returns:
//     The midpoint between a and b.
public static Point3D MidPoint(Point3D a, Point3D b);
//
// Summary:
//     Creates a deep copy of this 3D point.
//
// Returns:
//     The new 3D point.
public override object Clone();
public override Point2DSurrogate ConvertToSurrogate();
//
// Summary:
//     Computes the point-line distance.
//
// Parameters:
```

```
//   seg:
//     The line
//
// Returns:
//     The distance between this 3D point and its projection on the line.
public double DistanceTo(Segment3D seg);
//
// Summary:
//     Computes the point to plane signed distance.
//
// Parameters:
//   plane:
//     The plane
//
// Returns:
//     The distance between this 3D point and its projection on the plane.
public double DistanceTo(Plane plane);
//
// Summary:
//     Computes the distance to 3D point b.
//
// Parameters:
//   b:
//     The other point
//
// Returns:
//     The distance between this 3D point and b.
public double DistanceTo(Point3D b);
public override bool Equals(object obj);
public bool Equals(Point3D other);
public override int GetHashCode();
```

```
//
// Summary:
//     For internal use only.
public override XElement GetXElement();
//
// Summary:
//     Checks if the point is inside the specified volume.
//
// Parameters:
//   boxMin:
//     Lower-left-front corner
//
//   boxMax:
//     Top-right-rear corner
//
// Returns:
//     True if the point is inside, false otherwise.
public bool IsInside(Point3D boxMin, Point3D boxMax);
public override bool IsValid();
//
// Summary:
//     Project this point onto a 3D segment.
//
// Parameters:
//   seg:
//     The 3D segment
//
// Returns:
//     The projected point.
public Point3D ProjectTo(Segment3D seg);
//
```

```
// Summary:
//     Returns an array of point's coordinates.
//
// Returns:
//     The double array.
public override double[] ToArray();
//
// Summary:
//     Converts this 3D point to a human readable string.
//
// Returns:
//     A string that represents this 3d point.
public override string ToString();
//
// Summary:
//     Converts this 3D point to a human readable string.
//
// Returns:
//     A string that represents this 2D point.
public override string ToStringXml();
//
// Summary:
//     Transforms the 3D point by the specified transformation.
//
// Parameters:
//   xform:
//     The transformation to be applied
public override void TransformBy(Transformation xform);
public virtual void WriteAsFloat(BinaryWriter bw);

//
```

```
// Summary:
//     Addition between two 3D points.
//
// Parameters:
//   a:
//     First 3D point
//
//   b:
//     Second 3D point
//
// Returns:
//     The resulting 3D point.
public static Point3D operator +(Point3D a, Point3D b);
//
// Summary:
//     Addition between a 3D vector and a 3D point.
//
// Parameters:
//   v:
//     The 3D vector
//
//   p:
//     The 3D point
//
// Returns:
//     The resulting 3D point.
public static Point3D operator +(Vector3D v, Point3D p);
//
// Summary:
//     Addition between a 3D point and a 3D vector.
//
```

```
// Parameters:
//   a:
//     The 3D point
//
//   b:
//     The 3D vector
//
// Returns:
//     The resulting 3D point.
public static Point3D operator +(Point3D a, Vector3D b);
//
// Summary:
//     Subtraction between a 3D vector and a 3D point.
//
// Parameters:
//   v:
//     The 3D vector
//
//   p:
//     The 3D point
//
// Returns:
//     The resulting 3D point.
public static Point3D operator -(Vector3D v, Point3D p);
//
// Summary:
//     Subtraction between two 3D points.
//
// Parameters:
//   a:
//     First 3D point
```

```
//
//   b:
//     Second 3D point
//
// Returns:
//     The resulting 3D point.
public static Point3D operator -(Point3D a, Point3D b);
//
// Summary:
//     Subtraction between a 3D point and a 3D vector.
//
// Parameters:
//   a:
//     The 3D point
//
//   b:
//     The 3D vector
//
// Returns:
//     The resulting 3D point.
public static Point3D operator -(Point3D a, Vector3D b);
//
// Summary:
//     Product between a 3D point and a scalar s.
//
// Parameters:
//   s:
//     Scalar value
//
//   p:
//     The point
```

```
//
// Returns:
//     The resulting 3D point.
public static Point3D operator *(Point3D p, double s);
//
// Summary:
//     Product between a 3D point and a scalar s.
//
// Parameters:
//   s:
//     Scalar value
//
//   p:
//     The point
//
// Returns:
//     The resulting 3D point.
public static Point3D operator *(double s, Point3D p);
//
// Summary:
//     Division between a 3D point and a scalar s.
//
// Parameters:
//   s:
//     Scalar value
//
//   p:
//     The point
//
// Returns:
//     The resulting 3D point.
```

```
        public static Point3D operator /(Point3D p, double s);
        public static bool operator ==(Point3D left, Point3D right);
        public static bool operator !=(Point3D left, Point3D right);
    }
}

#region Assembly devDept.Eyeshot.v2023, Version=2023.1.353.0, Culture=neutral,
PublicKeyToken=9440825e8b4733bc
// C:\Program Files\devDept Software\Eyeshot 2023\Bin\net472\devDept.Eyeshot.v2023.dll
#endregion

using devDept.Geometry;
using devDept.Graphics;
using devDept.Serialization;
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;

namespace devDept.Eyeshot.Entities
{
    //
    // Summary:
    //     Region entity definition. By convention the first contour in the list is the
    //     outer and has counterclockwise orientation. Inners are oriented clockwise.
    public class Region : PlanarEntity, IFace, ICloneable, ISelectableSubItems
    {
        protected IndexLine[] edges;
        protected EntityGraphicsData drawEdges;

        //
        // Summary:
```

```
//     Empty Constructor. For internal use only.
public Region();
//
// Summary:
//     List of contours constructor.
//
// Parameters:
//   contours:
//     The list of contours
public Region(params ICurve[] contours);
//
// Summary:
//     List of contours constructor.
//
// Parameters:
//   contours:
//     The list of contours
public Region(IList<ICurve> contours);
//
// Summary:
//     Single contour constructor.
//
// Parameters:
//   outer:
//     The outer contour
public Region(ICurve outer);
//
// Summary:
//     Single contour and plane constructor.
//
// Parameters:
```

```
//   outer:
//     The outer contour
//
//   pln:
//     The contour plane
public Region(ICurve outer, Plane pln);
//
// Summary:
//     List of contours and plane constructor.
//
// Parameters:
//   contours:
//     The list of planar contours
//
//   pln:
//     The contours plane
public Region(IList<ICurve> contours, Plane pln);
//
// Summary:
//     Constructor for deserializing objects.
//
// Parameters:
//   info:
//     A System.Runtime.Serialization.SerializationInfo instance that defines the serialized
//     data.
//
//   context:
//     A System.Runtime.Serialization.StreamingContext instance that contains the serialized
//     data.
public Region(SerializationInfo info, StreamingContext context);
//
```

```
// Summary:
//     Single contour, plane and sorting flag constructor.
//
// Parameters:
//   outer:
//     The outer contour
//
//   pln:
//     The contour plane
//
//   sortAndOrient:
//     When true, the contour is properly oriented.
public Region(ICurve outer, Plane pln, bool sortAndOrient = true);
//
// Summary:
//     List of contours, plane and sorting flag constructor.
//
// Parameters:
//   contours:
//     The list of planar contours
//
//   pln:
//     The contours plane
//
//   sortAndOrient:
//     When true, the contours are properly sorted and oriented.
public Region(IList<ICurve> contours, Plane pln, bool sortAndOrient);
//
// Summary:
//     Copy constructor.
//
```

```
        // Parameters:
        //   another:
        //     The other region
        protected Region(Region another);
        //
        // Summary:
        //     Proprietary file format constructor.
        //
        // Parameters:
        //   surrogate:
        //     The devDept.Serialization.RegionSurrogate.
        //
        // Remarks:
        //     This constructor is used by devDept.Serialization.RegionSurrogate.ConvertToObject
        //     method.
        protected internal Region(RegionSurrogate surrogate);


        //
        // Summary:
        //     Gets o sets the list of contours inside the region.
        public List<ICurve> ContourList { get; set; }
        //
        // Summary:
        //     Gets or sets the region edges.
        //
        // Remarks:
        //     Region edges are generated automatically by
devDept.Eyeshot.Entities.Entity.Regen(System.Double).
        public IndexLine[] Edges { get; set; }
        //
        // Summary:
```

```
        //     Returns true when the number of contours is bigger than one.
        public bool HasHoles { get; }
        //
        // Summary:
        //     Gets or sets the region triangles.
        //
        // Remarks:
        //     Region triangles are generated automatically by
devDept.Eyeshot.Entities.Entity.Regen(System.Double).
        public IndexTriangle[] Triangles { get; set; }
        public selectionFilterType SelectionMode { get; set; }


        //
        // Summary:
        //     Creates a circular region.
        //
        // Parameters:
        //   radius:
        //     Radius
        public static Region CreateCircle(double radius);
        //
        // Summary:
        //     Creates a circular region by position.
        //
        // Parameters:
        //   x:
        //     Center's X coordinate
        //
        //   y:
        //     Center's Y coordinate
        //
```

```
//   radius:
//     Radius
public static Region CreateCircle(double x, double y, double radius);
//
// Summary:
//     Creates a circular region by plane.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   radius:
//     Radius
public static Region CreateCircle(Plane sketchPlane, double radius);
//
// Summary:
//     Creates a circular region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   x:
//     Center's X coordinate
//
//   y:
//     Center's Y coordinate
//
//   radius:
//     Radius
public static Region CreateCircle(Plane sketchPlane, double x, double y, double radius);
```

```
//
// Summary:
//     Creates a circular region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   center:
//     2D center point
//
//   radius:
//     Radius
public static Region CreateCircle(Plane sketchPlane, Point2D center, double radius);
//
// Summary:
//     Creates a region with the shape of a circular slot by position.
//
// Parameters:
//   x:
//     Position along the plane X axis
//
//   y:
//     Position along the plane Y axis
//
//   angle:
//     Angle in radians
//
//   radius:
//     Circle radius
//
```

```
    //   slotRadius:
    //     Slot radius
    public static Region CreateCircularSlot(double x, double y, double angle, double radius, double
slotRadius);
    //
    // Summary:
    //     Creates a region with the shape of a circular slot by plane.
    //
    // Parameters:
    //   sketchPlane:
    //     Sketch plane
    //
    //   angle:
    //     Angle in radians
    //
    //   radius:
    //     Circle radius
    //
    //   slotRadius:
    //     Slot radius
    public static Region CreateCircularSlot(Plane sketchPlane, double angle, double radius, double
slotRadius);
    //
    // Summary:
    //     Creates a region with the shape of a circular slot by plane, position and start
    //     angle.
    //
    // Parameters:
    //   sketchPlane:
    //     Sketch plane
    //
    //   x:
```

```
//     Position along the plane X axis
//
//   y:
//     Position along the plane Y axis
//
//   startAngle:
//     Start angle in radians
//
//   deltaAngle:
//     Delta angle in radians
//
//   radius:
//     Circle radius
//
//   slotRadius:
//     Slot radius
    public static Region CreateCircularSlot(Plane sketchPlane, double x, double y, double startAngle, double deltaAngle, double radius, double slotRadius);
//
// Summary:
//     Creates a region with the shape of a circular slot by plane and start angle.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   startAngle:
//     Start angle in radians
//
//   deltaAngle:
//     Delta angle in radians
```

```
//
//   radius:
//     Circle radius
//
//   slotRadius:
//     Slot radius
public static Region CreateCircularSlot(Plane sketchPlane, double startAngle, double deltaAngle,
double radius, double slotRadius);
//
// Summary:
//     Creates a region with the shape of a circular slot.
//
// Parameters:
//   angle:
//     Angle in radians
//
//   radius:
//     Circle radius
//
//   slotRadius:
//     Slot radius
public static Region CreateCircularSlot(double angle, double radius, double slotRadius);
//
// Summary:
//     Creates a region with the shape of a circular slot by position and start angle.
//
// Parameters:
//   x:
//     Position along the plane X axis
//
//   y:
```

```
//     Position along the plane Y axis
//
//   startAngle:
//     Start angle in radians
//
//   deltaAngle:
//     Delta angle in radians
//
//   radius:
//     Circle radius
//
//   slotRadius:
//     Slot radius
    public static Region CreateCircularSlot(double x, double y, double startAngle, double deltaAngle,
double radius, double slotRadius);
//
// Summary:
//     Creates a region with the shape of a circular slot by start angle.
//
// Parameters:
//   startAngle:
//     Start angle in radians
//
//   deltaAngle:
//     Delta angle in radians
//
//   radius:
//     Circle radius
//
//   slotRadius:
//     Slot radius
```

public static Region CreateCircularSlot(double startAngle, double deltaAngle, double radius, double slotRadius);

//

// Summary:

// Creates a region with the shape of a circular slot by plane and position.

//

// Parameters:

// sketchPlane:

// Sketch plane

//

// x:

// Position along the plane X axis

//

// y:

// Position along the plane Y axis

//

// angle:

// Angle in radians

//

// radius:

// Circle radius

//

// slotRadius:

// Slot radius

public static Region CreateCircularSlot(Plane sketchPlane, double x, double y, double angle, double radius, double slotRadius);

//

// Summary:

// Creates an elliptical region.

//

// Parameters:

// x:

```
//     Center's X coordinate
//
//   y:
//     Center's Y coordinate
//
//   rx:
//     Radius X
//
//   ry:
//     Radius Y
public static Region CreateEllipse(double rx, double ry);
//
// Summary:
//     Creates an elliptical region by plane.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   rx:
//     Radius X
//
//   ry:
//     Radius Y
public static Region CreateEllipse(Plane sketchPlane, double rx, double ry);
//
// Summary:
//     Creates an elliptical region by plane and position.
//
// Parameters:
//   sketchPlane:
```

```
//     Sketch plane
//
//   x:
//     Center's X coordinate
//
//   y:
//     Center's Y coordinate
//
//   rx:
//     Radius X
//
//   ry:
//     Radius Y
public static Region CreateEllipse(Plane sketchPlane, double x, double y, double rx, double ry);
//
// Summary:
//     Creates an elliptical region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   center:
//     2D center point
//
//   rx:
//     Radius X
//
//   ry:
//     Radius Y
public static Region CreateEllipse(Plane sketchPlane, Point2D center, double rx, double ry);
```

```
//
// Summary:
//     Creates an elliptical region by position.
//
// Parameters:
//   x:
//     Center's X coordinate
//
//   y:
//     Center's Y coordinate
//
//   rx:
//     Radius X
//
//   ry:
//     Radius Y
public static Region CreateEllipse(double x, double y, double rx, double ry);
//
// Summary:
//     Creates an hexagonal region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   x:
//     Position along the plane X axis
//
//   y:
//     Position along the plane Y axis
//
```

```
// radius:
//   The hexagon radius
//
// angle:
//   Rotation angle in radians
//
// inscribed:
//   When true, the radius is considered of the inscribed circle
public static Region CreateHexagon(Plane sketchPlane, double x, double y, double radius, double angle = 0, bool inscribed = false);
//
// Summary:
//   Creates an hexagonal region.
//
// Parameters:
// radius:
//   The hexagon radius
//
// inscribed:
//   When true, the radius is considered of the inscribed circle
public static Region CreateHexagon(double radius, bool inscribed = false);
//
// Summary:
//   Creates an hexagonal region by position.
//
// Parameters:
// x:
//   Position along the plane X axis
//
// y:
//   Position along the plane Y axis
```

```
        //
        //   radius:
        //     The hexagon radius
        //
        //   angle:
        //     Rotation angle in radians
        //
        //   inscribed:
        //     When true, the radius is considered of the inscribed circle
        public static Region CreateHexagon(double x, double y, double radius, double angle = 0, bool
inscribed = false);
        //
        // Summary:
        //     Creates an hexagonal region by plane.
        //
        // Parameters:
        //   sketchPlane:
        //     Sketch plane
        //
        //   radius:
        //     The hexagon radius
        //
        //   inscribed:
        //     When true, the radius is considered of the inscribed circle
        public static Region CreateHexagon(Plane sketchPlane, double radius, bool inscribed = false);
        //
        // Summary:
        //     Creates a polygonal region.
        //
        // Parameters:
        //   points:
```

```
//     A list of 2D points
public static Region CreatePolygon(params Point2D[] points);
//
// Summary:
//     Creates a polygonal region.
//
// Parameters:
//   points:
//     A list of 3D points laying on an arbitrary plane
public static Region CreatePolygon(params Point3D[] points);
//
// Summary:
//     Creates a polygonal region by plane.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   points:
//     A list of 2D points laying on the plane
public static Region CreatePolygon(Plane sketchPlane, IList<Point2D> points);
//
// Summary:
//     Creates a rectangular region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   x:
//     Position along the plane X axis
```

```
//
//  y:
//    Position along the plane Y axis
//
//  width:
//    Width
//
//  height:
//    Height
//
//  angle:
//    Rotation angle in radians
//
//  centered:
//    When true, the rectangle is built centered on the origin.
public static Region CreateRectangle(Plane sketchPlane, double x, double y, double width, double height, double angle = 0, bool centered = false);
//
// Summary:
//    Creates a rectangular region by plane.
//
// Parameters:
//  sketchPlane:
//    Sketch plane
//
//  width:
//    Width
//
//  height:
//    Height
//
```

```
    //   centered:
    //     When true, the rectangle is built centered on the origin.
    public static Region CreateRectangle(Plane sketchPlane, double width, double height, bool
centered = false);
    //
    // Summary:
    //     Creates a rectangular region.
    //
    // Parameters:
    //   width:
    //     Width
    //
    //   height:
    //     Height
    //
    //   centered:
    //     When true, the rectangle is built centered on the origin.
    public static Region CreateRectangle(double width, double height, bool centered = false);
    //
    // Summary:
    //     Creates a rectangular region by position.
    //
    // Parameters:
    //   x:
    //     Position along the plane X axis
    //
    //   y:
    //     Position along the plane Y axis
    //
    //   width:
    //     Width
```

```
        //
        //   height:
        //     Height
        //
        //   angle:
        //     Rotation angle in radians
        //
        //   centered:
        //     When true, the rectangle is built centered on the origin.
        public static Region CreateRectangle(double x, double y, double width, double height, double angle = 0, bool centered = false);
        //
        // Summary:
        //     Creates a rounded rectangular region.
        //
        // Parameters:
        //   width:
        //     Width
        //
        //   height:
        //     Height
        //
        //   radius:
        //     Corner radius
        //
        //   centered:
        //     When true, the rectangle is built centered on the origin.
        public static Region CreateRoundedRectangle(double width, double height, double radius, bool centered = false);
        //
        // Summary:
        //     Creates a rounded rectangular region by position.
```

```
//
// Parameters:
//   x:
//     Position along the plane X axis
//
//   y:
//     Position along the plane Y axis
//
//   width:
//     Width
//
//   height:
//     Height
//
//   radius:
//     Corner radius
//
//   angle:
//     Rotation angle in radians
//
//   centered:
//     When true, the rectangle is built centered on the origin.
public static Region CreateRoundedRectangle(double x, double y, double width, double height, double radius, double angle = 0, bool centered = false);
//
// Summary:
//     Creates a rounded rectangular region by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
```

```
//
//  x:
//    Position along the plane X axis
//
//  y:
//    Position along the plane Y axis
//
//  width:
//    Width
//
//  height:
//    Height
//
//  radius:
//    Corner radius
//
//  angle:
//    Rotation angle in radians
//
//  centered:
//    When true, the rectangle is built centered on the origin.
    public static Region CreateRoundedRectangle(Plane sketchPlane, double x, double y, double width, double height, double radius, double angle = 0, bool centered = false);
//
// Summary:
//    Creates a rounded rectangular region by plane.
//
// Parameters:
//  sketchPlane:
//    Sketch plane
//
```

```
// width:
//   Width
//
// height:
//   Height
//
// radius:
//   Corner radius
//
// centered:
//   When true, the rectangle is built centered on the origin.
public static Region CreateRoundedRectangle(Plane sketchPlane, double width, double height, double radius, bool centered = false);
//
// Summary:
//   Creates a region with the shape of a slot by position.
//
// Parameters:
// x:
//   Position along the plane X axis
//
// y:
//   Position along the plane Y axis
//
// length:
//   Center to center length
//
// radius:
//   Radius
//
// angle:
```

```
//     Rotation angle in radians
//
//   centered:
//     When true, the slot is built centered on the origin.
    public static Region CreateSlot(double x, double y, double length, double radius, double angle =
0, bool centered = false);
//
// Summary:
//     Creates a region with the shape of a slot by plane and position.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   x:
//     Position along the plane X axis
//
//   y:
//     Position along the plane Y axis
//
//   length:
//     Center to center length
//
//   radius:
//     Radius
//
//   angle:
//     Rotation angle in radians
//
//   centered:
//     When true, the slot is built centered on the origin.
```

```
public static Region CreateSlot(Plane sketchPlane, double x, double y, double length, double radius, double angle = 0, bool centered = false);
//
// Summary:
//     Creates a region with the shape of a slot by plane.
//
// Parameters:
//   sketchPlane:
//     Sketch plane
//
//   length:
//     Center to center length
//
//   radius:
//     Radius
//
//   centered:
//     When true, the slot is built centered on the origin.
public static Region CreateSlot(Plane sketchPlane, double length, double radius, bool centered = false);
//
// Summary:
//     Creates a region with the shape of a slot.
//
// Parameters:
//   length:
//     Center to center length
//
//   radius:
//     Radius
//
//   centered:
```

```
//     When true, the slot is built centered on the origin.
public static Region CreateSlot(double length, double radius, bool centered = false);
//
// Summary:
//     Boolean difference between one region and a list of regions.
//
// Parameters:
//   a:
//     First operand
//
//   b:
//     An array of second operands
//
// Returns:
//     The resulting region if the operation is successful, null/Nothing otherwise.
public static T Difference<T>(T a, params T[] b) where T : Region, new();
//
// Summary:
//     Boolean difference between two regions.
//
// Parameters:
//   a:
//     First operand
//
//   b:
//     Second operand
//
// Returns:
//     An array of regions if the operation is successful, null/Nothing otherwise.
public static T[] Difference<T>(T a, T b) where T : Region, new();
//
```

```
// Summary:
//     For internal use only.
public static Plane EstimatePlane(IList<ICurve> contourList, double tol);
//
// Summary:
//     Boolean intersection between two regions.
//
// Parameters:
//   a:
//     First operand
//
//   b:
//     Second operand
//
// Returns:
//     An array of regions if the operation is successful, null/Nothing otherwise.
public static T[] Intersection<T>(T a, T b) where T : Region, new();
//
// Summary:
//     Trims the region using the given - open - curves.
//
// Parameters:
//   original:
//     The region to be trim
//
//   curves:
//     The curves intersecting the region profiles
//
//   result:
//     The resulting list of regions
//
```

```
// Returns:
//     True if the operation was successful, false otherwise.
public static bool Trim(Region original, IList<ICurve> curves, out Region[] result);
//
// Summary:
//     Boolean union between two regions.
//
// Parameters:
//   a:
//     First operand
//
//   b:
//     Second operand
//
// Returns:
//     An array of regions if the operation is successful, null/Nothing otherwise.
public static T[] Union<T>(T a, T b) where T : Region, new();
//
// Summary:
//     Clears the contours selection status for all instances.
public void ClearSubContoursSelectionForAllInstances();
//
// Summary:
//     Creates a deep copy of this region.
//
// Returns:
//     The new region object.
public override object Clone();
public override void Compile(CompileParams data);
//
// Summary:
```

```
//     Converts this region to a devDept.Eyeshot.Entities.Mesh object.
//
// Parameters:
//   deviation:
//     The maximum deviation, zero for current tessellation.
//
//   meshNature:
//     The desired Mesh nature.
//
// Returns:
//     The resulting Mesh object.
    public Mesh ConvertToMesh(double deviation = 0, Mesh.natureType meshNature =
Mesh.natureType.Plain);
    public Mesh ConvertToMesh(double deviation, double angleInRadians, Mesh.natureType nature,
bool weld);
//
// Summary:
//     Converts this region to a devDept.Eyeshot.Entities.Mesh object.
//
// Parameters:
//   deviation:
//     The maximum deviation, zero for current tessellation.
//
//   meshNature:
//     The desired Mesh nature.
//
// Returns:
//     The resulting Mesh object.
    public T ConvertToMesh<T>(double deviation = 0, Mesh.natureType meshNature =
Mesh.natureType.Plain) where T : Mesh, new();
//
// Summary:
```

```
//     Converts this region to a devDept.Eyeshot.Entities.Mesh object.
//
// Parameters:
//   data:
//     The regeneration params
//
//   meshNature:
//     The desired Mesh nature.
//
// Returns:
//     The resulting Mesh object.
    public T ConvertToMesh<T>(RegenParams data = null, Mesh.natureType meshNature =
Mesh.natureType.Plain) where T : Mesh, new();
//
// Summary:
//     Converts this region to a devDept.Eyeshot.Entities.Solid object.
//
// Parameters:
//   tolerance:
//     Tessellation tolerance
    public Solid ConvertToSolid(double tolerance);
//
// Summary:
//     Converts this region to a devDept.Eyeshot.Entities.Solid object.
//
// Parameters:
//   tolerance:
//     Tessellation tolerance
    public T ConvertToSolid<T>(double tolerance) where T : Solid, new();
//
// Summary:
```

```
    //     Converts this region to a devDept.Eyeshot.Entities.Surface object.
    public PlanarSurface ConvertToSurface();
    public override EntitySurrogate ConvertToSurrogate();
    public override void Dispose();
    public override string Dump(linearUnitsType linearUnits = linearUnitsType.Unitless,
massUnitsType massUnits = massUnitsType.Unitless, LayerKeyedCollection layers = null,
MaterialKeyedCollection materials = null, BlockKeyedCollection blocks = null);
    public override Point3D[] EstimateBoundingBox(BlockKeyedCollection blocks,
LayerKeyedCollection layers);
    //
    // Summary:
    //     Extrudes the region to create a new Brep.
    //
    // Parameters:
    //   amount:
    //     The extrusion amount
    //
    //   angleInRadians:
    //     The draft angle in radians
    //
    //   tolerance:
    //     The regeneration tolerance. 0 for default value
    //
    // Returns:
    //     >The resulting Brep object.
    public Brep ExtrudeAsBrep(Interval amount, double angleInRadians = 0, double tolerance = 0);
    //
    // Summary:
    //     Extrudes a region creating a new devDept.Eyeshot.Entities.Brep.
    //
    // Parameters:
    //   amount:
```

```
//     The extrusion amount
//
//   angleInRadians:
//     The draft angle in radians
//
//   tolerance:
//     The regeneration tolerance. 0 for default value
//
// Returns:
//     >The resulting Brep object.
public Brep ExtrudeAsBrep(double amount, double angleInRadians = 0, double tolerance = 0);
//
// Summary:
//     Extrudes the region to create a new Brep.
//
// Parameters:
//   amount:
//     The extrusion amount
//
//   angleInRadians:
//     The draft angle in radians
//
//   tolerance:
//     The regeneration tolerance. 0 for default value
//
// Returns:
//     >The resulting Brep object.
public Brep ExtrudeAsBrep(Vector3D amount, double angleInRadians = 0, double tolerance = 0);
//
// Summary:
//     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh.
```

```
        //
        // Parameters:
        //   amount:
        //     Extrusion amount
        //
        //   tolerance:
        //     The regeneration tolerance
        //
        //   meshNature:
        //     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
        //     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
        //     object
        //
        // Returns:
        //     The resulting Mesh object.
        public Mesh ExtrudeAsMesh(Vector3D amount, double tolerance, Mesh.natureType
meshNature);
        //
        // Summary:
        //     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh along plane's
        //     Z-axis.
        //
        // Parameters:
        //   amount:
        //     Extrusion amount
        //
        //   tolerance:
        //     The regeneration tolerance
        //
        //   meshNature:
        //     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
```

```
    //     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
    //     object
    //
    // Returns:
    //     The resulting Mesh object.
    public override Mesh ExtrudeAsMesh(double amount, double tolerance, Mesh.natureType
meshNature);
    //
    // Summary:
    //     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh along plane's
    //     Z-axis.
    //
    // Parameters:
    //   amount:
    //     Extrusion amount
    //
    //   tolerance:
    //     The regeneration tolerance
    //
    //   meshNature:
    //     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
    //     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
    //     object
    //
    // Returns:
    //     The resulting Mesh object.
    public T ExtrudeAsMesh<T>(double amount, double tolerance, Mesh.natureType meshNature)
where T : Mesh, new();
    //
    // Summary:
    //     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh.
    //
```

```
// Parameters:
//   amount:
//     Extrusion amount
//
//   data:
//     The regeneration params
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
public Mesh ExtrudeAsMesh(Vector3D amount, RegenParams data, Mesh.natureType
meshNature);
//
// Summary:
//     Extrudes the region to create a new Mesh.
//
// Parameters:
//   amount:
//     The extrusion interval.
//
//   tolerance:
//     The regeneration tolerance. 0 for default value.
//
//   meshNature:
//     This is the devDept.Eyeshot.Entities.Mesh.natureType of the new
devDept.Eyeshot.Entities.Mesh
//     object.
//
```

```
        // Returns:
        //     >The resulting Mesh object.
        public Mesh ExtrudeAsMesh<T>(Interval amount, double tolerance, Mesh.natureType
meshNature) where T : Mesh, new();
        //
        // Summary:
        //     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh.
        //
        // Parameters:
        //   amount:
        //     Extrusion amount
        //
        //   tolerance:
        //     The regeneration tolerance
        //
        //   meshNature:
        //     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
        //     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
        //     object
        //
        // Returns:
        //     The resulting Mesh object.
        public T ExtrudeAsMesh<T>(Vector3D amount, double tolerance, Mesh.natureType
meshNature) where T : Mesh, new();
        //
        // Summary:
        //     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh.
        //
        // Parameters:
        //   amount:
        //     Extrusion amount
        //
```

```
//   data:
//     The regeneration params
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
    public T ExtrudeAsMesh<T>(Vector3D amount, RegenParams data, Mesh.natureType
meshNature) where T : Mesh, new();
//
// Summary:
//     Extrudes a region creating a new devDept.Eyeshot.Entities.Mesh.
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   tolerance:
//     The regeneration tolerance
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
    public Mesh ExtrudeAsMesh(Interval amount, double tolerance, Mesh.natureType meshNature);
```

```
//
// Summary:
//     Extrudes this region.
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   data:
//     The regeneration params
//
// Returns:
//     The resulting GSolid object.
public T ExtrudeAsSolid<T>(Vector3D amount, RegenParams data) where T : Solid, new();
//
// Summary:
//     Extrudes this region.
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   data:
//     The regeneration params
//
// Returns:
//     The resulting GSolid object.
public Solid ExtrudeAsSolid(Vector3D amount, RegenParams data);
//
// Summary:
//     Extrudes this region.
```

```
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public Solid ExtrudeAsSolid(Vector3D amount, double tolerance);
//
// Summary:
//     Extrudes this region.
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public T ExtrudeAsSolid<T>(double amount, double tolerance) where T : Solid, new();
//
// Summary:
//     Extrudes this region.
//
// Parameters:
//   amount:
```

```
    //     Extrusion amount
    //
    //   tolerance:
    //     The regeneration tolerance
    //
    // Returns:
    //     The resulting GSolid object.
    public Solid ExtrudeAsSolid(double amount, double tolerance);
    //
    // Summary:
    //     Extrudes this region.
    //
    // Parameters:
    //   x:
    //     Extrusion amount along the axis X
    //
    //   y:
    //     Extrusion amount along the axis Y
    //
    //   z:
    //     Extrusion amount along the axis Z
    //
    //   tolerance:
    //     The regeneration tolerance
    //
    // Returns:
    //     The resulting GSolid object.
    public T ExtrudeAsSolid<T>(double x, double y, double z, double tolerance) where T : Solid,
new();
    //
    // Summary:
```

```
//     Extrudes this region.
//
// Parameters:
//   x:
//     Extrusion amount along the axis X
//
//   y:
//     Extrusion amount along the axis Y
//
//   z:
//     Extrusion amount along the axis Z
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public Solid ExtrudeAsSolid(double x, double y, double z, double tolerance);
//
// Summary:
//     Extrudes this region.
//
// Parameters:
//   amount:
//     Extrusion amount
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
```

```
public T ExtrudeAsSolid<T>(Vector3D amount, double tolerance) where T : Solid, new();

public Surface[] ExtrudeAsSurface(double amount, double draftAngleInRadians, double
tolerance);

//

// Summary:

//     Extrudes a region creating a new devDept.Eyeshot.Entities.Surface.

//

// Parameters:

//   amount:

//     Extrusion amount

//

// Returns:

//     >The resulting Surface array.

public Surface[] ExtrudeAsSurface(Vector3D amount);

//

// Summary:

//     Extrudes a region creating a new devDept.Eyeshot.Entities.Surface.

//

// Parameters:

//   amount:

//     Extrusion amount

//

// Returns:

//     >The resulting Surface array.

public Surface[] ExtrudeAsSurface(double amount);

public Surface[] ExtrudeAsSurface(Vector3D amount, double draftAngleInRadians, double
tolerance);

//

// Summary:

//     Returns a list of triangles hit by the provided segment.

//

// Parameters:
```

```
    //   transf:
    //     The transformation applied to the entity (necessary if the entity is inside a
    //     Block to propagate the BlockReference transformation)
    //
    //   seg:
    //     The 3D segment representing the viewing direction
    //
    // Returns:
    //     The sorted list of triangles intersecting the provided segment.
    public IList<HitTriangle> FindClosestTriangle(Transformation transf, Segment3D seg);
    public void FlipNormal();
    public double GetArea(out Point3D centroid);
    public double GetMass(Material material, linearUnitsType linearUnits, massUnitsType
massUnits, out double convertedDensity);
    //
    // Summary:
    //     Populates a System.Runtime.Serialization.SerializationInfo instance with the
    //     data needed to serialize the target object.
    //
    // Parameters:
    //   info:
    //     A System.Runtime.Serialization.SerializationInfo instance that defines the serialized
    //     data.
    //
    //   context:
    //     A System.Runtime.Serialization.StreamingContext instance that contains the serialized
    //     data.
    public override void GetObjectData(SerializationInfo info, StreamingContext context);
    //
    // Summary:
    //     Computes the perimeter of the region.
```

```
//
// Returns:
//     The perimeter amount.
//
// Remarks:
//     The perimeter amount includes also the inner loops length.
public double GetPerimeter();
public Mesh[] GetTessellation();
public double GetVolume(out Point3D centroid);
//
// Summary:
//     Tells if there is a selected contour.
//
// Returns:
//     True if there is a selected contours
public bool IsAnySubContourSelected();
//
// Summary:
//     Tests if a 3D point is inside the region.
//
// Parameters:
//   testPoint:
//     The test 3D point
//
// Returns:
//     True if the 3D point is inside, false otherwise.
public bool IsPointInside(Point3D testPoint);
//
// Summary:
//     Tests if a 2D point is inside the region.
//
```

```
// Parameters:
//   testPoint:
//     The test 2D point, given in the plane's coordinate system
//
// Returns:
//     True if the 2D point is inside, false otherwise.
public bool IsPointInside(Point2D testPoint);
//
// Summary:
//     Tests if a 3D point is on one of the region's contours.
//
// Parameters:
//   testPoint:
//     The test 3D point
//
//   tol:
//     Maximum distance under which the point is considered on a contour.
//
// Returns:
//     True if the 3D point is on a contour, false otherwise.
public bool IsPointOnContour(Point3D testPoint, double tol);
public override bool IsValid();
//
// Summary:
//     Offsets the region of the specified amount.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   tolerance:
```

```
//     Tolerance
//
//   sharp:
//     If false, offset curves are connected with an arc.
//
// Returns:
//     The offset curves of the region contours.
public ICurve[] Offset(double amount, double tolerance, bool sharp);
//
// Summary:
//     Offsets the region of the specified amount.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   tolerance:
//     Tolereance
//
// Returns:
//     The offset curves of the region contours.
public ICurve[] Offset(double amount, double tolerance);
//
// Summary:
//     Pocket function for NC toolpaths.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   ct:
```

```
//     Corner type
//
//   miterLimit:
//     The higher the miter limit setting, the sharper the corner can be while retaining
//     its miter.
//
//   tol:
//     Regeneration tolerance
//
// Returns:
//     An array of curves.
[Obsolete("This method is deprecated.")]
public ICurve[] Pocket(double amount, cornerType ct, double miterLimit, double tol);
//
// Summary:
//     Pocket function for NC toolpaths.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   tol:
//     Regeneration tolerance
//
// Returns:
//     An array of curves, pocket of the region.
//
// Remarks:
//     Despite the type of the ICurves creating the contours of the region, it will
//     return only Arc and Lines for every path, occasionally organized into CompositeCurves.
//     Sharp corners are not preserved. It doesn't work if the contours of the region
```

```
//     intersect.
public ICurve[] Pocket(double amount, double tol);
//
// Summary:
//     Pocket function for NC toolpaths.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   ct:
//     Corner type
//
//   tol:
//     Regeneration tolerance
//
// Returns:
//     An array of curves.
[Obsolete("This method is deprecated.")]
public ICurve[] Pocket(double amount, cornerType ct, double tol);
//
// Summary:
//     Quick offset function for NC toolpaths.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   tol:
//     Regeneration tolerance
//
```

```
// Returns:
//     An array of curves, offset of the region contour list.
//
// Remarks:
//     Despite the type of the ICurves creating the contours of the region, it will
//     return only Arc and Lines, occasionally organized into CompositeCurves. Sharp
//     corners are not preserved. It doesn't work if the contours of the region intersect.
public ICurve[] QuickOffset(double amount, double tol);
//
// Summary:
//     Quick offset function for NC toolpaths.
//
// Parameters:
//   amount:
//     Signed offset amount
//
//   ct:
//     Corner type
//
//   tol:
//     Regeneration tolerance
//
// Returns:
//     An array of curves.
[Obsolete("This method is deprecated.")]
public ICurve[] QuickOffset(double amount, cornerType ct, double tol);
//
// Summary:
//     Quick offset function for NC toolpaths.
//
// Parameters:
```

```
//   amount:
//     Signed offset amount
//
//   ct:
//     Corner type
//
//   miterLimit:
//     The higher the miter limit setting, the sharper the corner can be while retaining
//     its miter.
//
//   tol:
//     Regeneration tolerance
//
// Returns:
//     An array of curves.
[Obsolete("This method is deprecated.")]
public ICurve[] QuickOffset(double amount, cornerType ct, double miterLimit, double tol);
public override void Regen(RegenParams data);
public void ResetSelectionMode();
//
// Summary:
//     Revolves this region around an axis.
//
// Parameters:
//   startAngle:
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axisStart:
```

```
//     The axis start point
//
//   axisEnd:
//     The axis end point
//
//   tolerance:
//     the regeneration tolerance
//
// Returns:
//     The resulting Brep object.
    public Brep RevolveAsBrep(double startAngle, double deltaAngle, Point3D axisStart, Point3D
axisEnd, double tolerance = 0);
//
// Summary:
//     Revolves this region around an axis.
//
// Parameters:
//   deltaAngle:
//     Revolution angle in radians
//
//   axis:
//     Axis direction
//
//   center:
//     Axis start point
//
//   tolerance:
//     the regeneration tolerance. 0 for default value
//
// Returns:
//     The resulting Brep object.
```

```
public Brep RevolveAsBrep(double deltaAngle, Vector3D axis, Point3D center, double tolerance =
0);

//

// Summary:

//     Revolves this region around an axis.

//

// Parameters:

//   startAngle:

//     Revolution start angle in radians

//

//   deltaAngle:

//     Revolution delta angle in radians

//

//   axis:

//     Axis direction

//

//   center:

//     Axis start point

//

//   tolerance:

//     the regeneration tolerance

//

// Returns:

//     The resulting Brep object.

public Brep RevolveAsBrep(double startAngle, double deltaAngle, Vector3D axis, Point3D center,
double tolerance = 0);

//

// Summary:

//     Revolves this region around an axis.

//

// Parameters:

//   startAngle:
```

```
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axisStart:
//     The axis start point
//
//   axisEnd:
//     The axis end point
//
//   slices:
//     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
//
//   tolerance:
//     The regeneration tolerance
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
    public T RevolveAsMesh<T>(double startAngle, double deltaAngle, Point3D axisStart, Point3D
axisEnd, int slices, double tolerance, Mesh.natureType meshNature) where T : Mesh, new();
//
// Summary:
//     Revolves this region around an axis.
//
```

```
// Parameters:
//   startAngle:
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axisStart:
//     The axis start point
//
//   axisEnd:
//     The axis end point
//
//   slices:
//     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
//
//   tolerance:
//     The regeneration tolerance
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
public Mesh RevolveAsMesh(double startAngle, double deltaAngle, Point3D axisStart, Point3D
axisEnd, int slices, double tolerance, Mesh.natureType meshNature);
//
// Summary:
```

```
    //     Revolves this region around an axis.
    //
    // Parameters:
    //   startAngle:
    //     Revolution start angle in radians
    //
    //   deltaAngle:
    //     Revolution delta angle in radians
    //
    //   axis:
    //     Axis direction
    //
    //   center:
    //     Axis start point
    //
    //   slices:
    //     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
    //
    //   tolerance:
    //     The regeneration tolerance
    //
    //   meshNature:
    //     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
    //     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
    //     object
    //
    // Returns:
    //     The resulting Mesh object.
    public Mesh RevolveAsMesh(double startAngle, double deltaAngle, Vector3D axis, Point3D
center, int slices, double tolerance, Mesh.natureType meshNature);
```

```
//
// Summary:
//     Revolves this region around an axis.
//
// Parameters:
//   startAngle:
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axis:
//     Axis direction
//
//   center:
//     Axis start point
//
//   slices:
//     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
//
//   tolerance:
//     The regeneration tolerance
//
//   meshNature:
//     Entity region needs to be converted to devDept.Eyeshot.Entities.Mesh, this is
//     the devDept.Eyeshot.Entities.Mesh.natureType of the new devDept.Eyeshot.Entities.Mesh
//     object
//
// Returns:
//     The resulting Mesh object.
```

public T RevolveAsMesh<T>(double startAngle, double deltaAngle, Vector3D axis, Point3D center, int slices, double tolerance, Mesh.natureType meshNature) where T : Mesh, new();

//

// Summary:

// Revolves this region around an axis.

//

// Parameters:

// startAngle:

// Revolution start angle in radians

//

// deltaAngle:

// Revolution delta angle in radians

//

// axis:

// Axis direction

//

// center:

// Axis start point

//

// slices:

// Number of slices generated. This value can be also found using devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System.Double)

//

// tolerance:

// The regeneration tolerance

//

// Returns:

// The resulting GSolid object.

public Solid RevolveAsSolid(double startAngle, double deltaAngle, Vector3D axis, Point3D center, int slices, double tolerance);

//

```
// Summary:
//     Revolves this region around an axis.
//
// Parameters:
//   startAngle:
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axisStart:
//     The axis start point
//
//   axisEnd:
//     The axis end point
//
//   slices:
//     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public Solid RevolveAsSolid(double startAngle, double deltaAngle, Point3D axisStart, Point3D
axisEnd, int slices, double tolerance);
//
// Summary:
//     Revolves this region around an axis.
//
```

```
// Parameters:
//   startAngle:
//     Revolution start angle in radians
//
//   deltaAngle:
//     Revolution delta angle in radians
//
//   axisStart:
//     The axis start point
//
//   axisEnd:
//     The axis end point
//
//   slices:
//     Number of slices generated. This value can be also found using devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System.Double)
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public T RevolveAsSolid<T>(double startAngle, double deltaAngle, Point3D axisStart, Point3D axisEnd, int slices, double tolerance) where T : Solid, new();
//
// Summary:
//     Revolves this region around an axis.
//
// Parameters:
//   startAngle:
//     Revolution start angle in radians
```

```
        //
        //   deltaAngle:
        //     Revolution delta angle in radians
        //
        //   axis:
        //     Axis direction
        //
        //   center:
        //     Axis start point
        //
        //   slices:
        //     Number of slices generated. This value can be also found using
devDept.Geometry.Utility.NumberOfSegments(System.Double,System.Double,System.Double,System
.Double)
        //
        //   tolerance:
        //     The regeneration tolerance
        //
        // Returns:
        //     The resulting GSolid object.
        public T RevolveAsSolid<T>(double startAngle, double deltaAngle, Vector3D axis, Point3D center,
int slices, double tolerance) where T : Solid, new();
        //
        // Summary:
        //     Revolves this region around an axis.
        //
        // Parameters:
        //   startAngle:
        //     Revolution start angle in radians
        //
        //   deltaAngle:
        //     Revolution delta angle in radians
```

```
        //
        //   axisStart:
        //     The axis start point
        //
        //   axisEnd:
        //     The axis end point
        //
        // Returns:
        //     The resulting Surface array object.
        public Surface[] RevolveAsSurface(double startAngle, double deltaAngle, Point3D axisStart,
Point3D axisEnd);
        //
        // Summary:
        //     Revolves this region around an axis.
        //
        // Parameters:
        //   startAngle:
        //     Revolution start angle in radians
        //
        //   deltaAngle:
        //     Revolution delta angle in radians
        //
        //   axis:
        //     Axis direction
        //
        //   center:
        //     Axis start point
        //
        // Returns:
        //     The resulting Surface array object.
        public Surface[] RevolveAsSurface(double startAngle, double deltaAngle, Vector3D axis, Point3D
center);
```

```
public ICurve[] Section(Plane pln, double tol);
//
// Summary:
//     Sorts and orients internal contours.
public void SortAndOrient();
//
// Summary:
//     Creates a Brep by sweeping the region along a rail.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tol:
//     The trim tolerance
//
//   methodType:
//     The Sweep method
//
// Returns:
//     The resulting Brep object.
public Brep SweepAsBrep(ICurve rail, double tol, sweepMethodType methodType =
sweepMethodType.RotationMinimizingFrames);
//
// Summary:
//     Creates a Brep by sweeping the region along a rail.
//
// Parameters:
//   rail:
//     The rail curve
//
```

```
        //   tol:
        //     The trim tolerance
        //
        //   methodType:
        //     The Sweep method
        //
        //   merge:
        //     When true, it merges al the trimmed pieces. It Keeps them as separated objects
        //     otherwise.
        //
        // Returns:
        //     The resulting Brep object.
        public Brep[] SweepAsBrep(ICurve rail, double tol, bool merge, sweepMethodType methodType
= sweepMethodType.RotationMinimizingFrames);
        //
        // Summary:
        //     Creates a mesh by sweeping the region along a rail.
        //
        // Parameters:
        //   rail:
        //     The rail curve
        //
        //   tol:
        //     The trim tolerance
        //
        //   merge:
        //     When true, it merges al the trimmed pieces. It Keeps them as separated objects
        //     otherwise.
        //
        //   methodType:
        //     The Sweep method
```

```
        //
        //   natureType:
        //     The desired mesh nature
        //
        // Returns:
        //     The resulting mesh object.
        public T SweepAsMesh<T>(ICurve rail, double tol, bool merge, sweepMethodType methodType
= sweepMethodType.RotationMinimizingFrames, Mesh.natureType natureType =
Mesh.natureType.Smooth) where T : Mesh, new();
        //
        // Summary:
        //     Creates a mesh by sweeping the region along a rail.
        //
        // Parameters:
        //   rail:
        //     The rail curve
        //
        //   tol:
        //     The trim tolerance
        //
        //   methodType:
        //     The Sweep method
        //
        //   natureType:
        //     The desired mesh nature
        //
        // Returns:
        //     The resulting mesh object.
        public Mesh SweepAsMesh(ICurve rail, double tol, sweepMethodType methodType =
sweepMethodType.RotationMinimizingFrames, Mesh.natureType natureType =
Mesh.natureType.Smooth);
        //
```

```
// Summary:
//     Creates a mesh by sweeping the region along a rail.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tol:
//     The trim tolerance
//
//   methodType:
//     The Sweep method
//
//   natureType:
//     The desired mesh nature
//
// Returns:
//     The resulting mesh object.
public T SweepAsMesh<T>(ICurve rail, double tol, sweepMethodType methodType =
sweepMethodType.RotationMinimizingFrames, Mesh.natureType natureType =
Mesh.natureType.Smooth) where T : Mesh, new();
//
// Summary:
//     Creates a mesh by sweeping the region along a rail.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tol:
//     The trim tolerance
//
```

```
//   merge:
//     When true, it merges al the trimmed pieces. It Keeps them as separated objects
//     otherwise.
//
//   methodType:
//     The Sweep method
//
//   natureType:
//     The desired mesh nature
//
// Returns:
//     The resulting mesh object.
public Mesh[] SweepAsMesh(ICurve rail, double tol, bool merge, sweepMethodType
methodType = sweepMethodType.RotationMinimizingFrames, Mesh.natureType natureType =
Mesh.natureType.Smooth);
//
// Summary:
//     Sweeps this region along the provided trajectory.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tolerance:
//     The regeneration tolerance
//
//   merge:
//
//   sweepMethod:
//
// Returns:
//     The resulting GSolid object.
```

public T SweepAsSolid<T>(ICurve rail, double tolerance, bool merge, sweepMethodType sweepMethod = sweepMethodType.RotationMinimizingFrames) where T : Solid, new();

//

// Summary:

// Sweeps this region along the provided trajectory.

//

// Parameters:

// rail:

// The rail curve

//

// tolerance:

// The regeneration tolerance

//

// Returns:

// The resulting GSolid object.

public T SweepAsSolid<T>(ICurve rail, double tolerance, sweepMethodType sweepMethod = sweepMethodType.RotationMinimizingFrames) where T : Solid, new();

//

// Summary:

// Sweeps this region along the provided trajectory.

//

// Parameters:

// rail:

// The rail curve

//

// tolerance:

// The regeneration tolerance

//

// merge:

//

// sweepMethod:

//

```
// Returns:
//     The resulting GSolid object.
public Solid SweepAsSolid(ICurve rail, double tolerance, bool merge, sweepMethodType
sweepMethod = sweepMethodType.RotationMinimizingFrames);
//
// Summary:
//     Sweeps this region along the provided trajectory.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tolerance:
//     The regeneration tolerance
//
// Returns:
//     The resulting GSolid object.
public Solid SweepAsSolid(ICurve rail, double tolerance, sweepMethodType sweepMethod =
sweepMethodType.RotationMinimizingFrames);
//
// Summary:
//     Creates a surface by sweeping the region along a rail.
//
// Parameters:
//   rail:
//     The rail curve
//
//   tol:
//     The trim tolerance
//
//   methodType:
//     The Sweep method
```

```
        //
        // Returns:
        //     The resulting surface array.
        public Surface[] SweepAsSurface(ICurve rail, double tol, sweepMethodType methodType =
sweepMethodType.RotationMinimizingFrames);
        public override string ToString();
        public override void TransformBy(Transformation transform);
        //
        // Summary:
        //     Triangulates this Region.
        //
        // Parameters:
        //   elementSize:
        //     The desired element size
        //
        //   smoothingPasses:
        //     The number of smoothing passes
        //
        //   hardEdges:
        //     The internal constraint segments, can be null/Nothing.
        //
        // Returns:
        //     The resulting Mesh object, null/Nothing in case of failure.
        [Obsolete("Use the PlaneMesher class instead.")]
        public Mesh Triangulate(double elementSize, int smoothingPasses = 4, IList<LinearPath>
hardEdges = null);
        protected override void DrawEntity(RenderContextBase context, object myParams);
        protected override void InitGraphicsData(RenderContextBase renderContext);
        //
        // Summary:
        //     Clears the selectionFlag on the contours.
        //
```

```
        // Parameters:
        //   selectionFlag:
        //     The selection status flag to clear.
        protected internal void ClearSubContoursSelection(selectionStatusType selectionFlag);
        protected internal override void DrawEdges(DrawParams data);
        protected internal override void DrawForSelection(GfxDrawForSelectionParams data);
        protected internal override void DrawForSelectionSubContours(GfxDrawForSelectionParams
data);
        protected internal override void DrawForSelectionWireframe(GfxDrawForSelectionParams data);
        protected internal override void DrawForShadow(RenderParams data);
        protected internal override void DrawHiddenLines(DrawParams data);
        protected internal override void DrawIsocurves(DrawParams data);
        protected internal override void DrawNormals(DrawParams data);
        protected internal override void DrawSelected(DrawParams data);
        protected internal override void DrawWireframe(DrawParams data);
        protected internal override bool InsideOrCrossingFrustum(FrustumParams data);
        protected internal override bool InsideOrCrossingScreenPolygon(ScreenPolygonParams data);
        protected internal override bool SelectedInternal();
        protected internal override bool ThroughTriangle(FrustumParams data);
        protected internal override bool ThroughTriangleScreenPolygon(ScreenPolygonParams data);
        protected internal override void TransformAllVerticesRecursive(Transformation t);
    }
}

#region Assembly devDept.Eyeshot.v2023, Version=2023.1.353.0, Culture=neutral,
PublicKeyToken=9440825e8b4733bc
// C:\Program Files\devDept Software\Eyeshot 2023\Bin\net472\devDept.Eyeshot.v2023.dll
#endregion

using devDept.Geometry;
using devDept.Graphics;
using devDept.Serialization;
```

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Runtime.Serialization;
using System.Xml.Linq;

namespace devDept.Eyeshot.Entities
{
   //
   // Summary:
   //     Base class for all Eyeshot entities.
   public abstract class Entity : ISerializable, ICloneable, IDisposable, IEntity, ISelectableItem,
INotifyVisibleChanged
   {
      protected OrientedBoundingRect _localOB;
      //
      // Summary:
      //     Default graphics data.
      protected internal EntityGraphicsData drawData;
      //
      // Summary:
      //     Graphics data for line pattern.
      protected internal EntityGraphicsData drawPattern;

      //
      // Summary:
      //     Constructor for deserializing objects.
      //
      // Parameters:
      //   info:
```

```
//     A System.Runtime.Serialization.SerializationInfo instance that defines the serialized
//     data.
//
//   context:
//     A System.Runtime.Serialization.StreamingContext instance that contains the serialized
//     data.
public Entity(SerializationInfo info, StreamingContext context);
//
// Summary:
//     Empty constructor.
protected Entity();
//
// Summary:
//     Nature only constructor.
//
// Parameters:
//   nature:
//     The entity nature
protected Entity(entityNatureType nature);
protected Entity(Entity another);
//
// Summary:
//     Color and nature constructor.
//
// Parameters:
//   color:
//     A devDept.Eyeshot.Entities.Entity.Color structure that indicates the color of
//     this entity.
//
//   nature:
//     The entity nature
```

```
protected Entity(Color color, entityNatureType nature);

//
// Summary:
//     Gets or sets the line type scale.
//
// Remarks:
//     When the line type of an entity already added to the Workspace is changed the
//     devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions) must be called.
public virtual float LineTypeScale { get; set; }
//
// Summary:
//     Gets or sets the line type name of the Workspace.LineTypes collection. In use
//     only if the devDept.Eyeshot.Entities.Entity.LineTypeMethod is byEntity.
//
// Remarks:
//     When the line type of an entity already added to the Workspace is changed the
//     devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions) must be called.
public virtual string LineTypeName { get; set; }
//
// Summary:
//     Gets or sets the entity printing order. Entities with a lower value are printed
//     first. Default value is zero.
//
// Remarks:
//     Useful in case of overlapping objects to set which ones to print in front of
//     the others.
//     Only meaningful for leaf entities, this property is ignored on BlockReference.
public sbyte PrintOrder { get; set; }
//
// Summary:
```

```
    //     Gets or sets the entity group index.
    public int GroupIndex { get; set; }
    //
    // Summary:
    //     Gets or sets the entity visibility status for the top-level.
    //
    // Remarks:
    //     When the visibility of an entity changes, it's important to call
devDept.Eyeshot.Document.UpdateBoundingBox
    //     to properly update the visualization.
    //     To gets or sets the visibility status of the instances inside nested BlockReferences
    //     use
devDept.Eyeshot.Entities.Entity.GetVisibility(System.Collections.Generic.Stack{devDept.Eyeshot.Entiti
es.BlockReference})
    //     and
devDept.Eyeshot.Entities.Entity.SetVisibility(System.Boolean,System.Collections.Generic.Stack{devDe
pt.Eyeshot.Entities.BlockReference}).
    public virtual bool Visible { get; set; }
    //
    // Summary:
    //     Gets or sets the name of the referenced devDept.Eyeshot.Layer.
    //
    // Remarks:
    //     When the layer of an devDept.Eyeshot.Entities.ICurve already added to the Workspace
    //     is changed, the devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions)
    //     must be called if the devDept.Eyeshot.Entities.Entity.LineTypeMethod is
devDept.Eyeshot.Entities.colorMethodType.byLayer.
    public virtual string LayerName { get; set; }
    //
    // Summary:
    //     Gets or sets the value that tells if the top-level item can be selected.
    //
    // Remarks:
```

```
    //     To gets or sets the selectable status of the instances inside nested BlockReferences
    //     use
devDept.Eyeshot.Entities.Entity.GetSelectability(System.Collections.Generic.Stack{devDept.Eyeshot.Entities.BlockReference})
    //     and
devDept.Eyeshot.Entities.Entity.SetSelectability(System.Boolean,System.Collections.Generic.Stack{devDept.Eyeshot.Entities.BlockReference}).
    public virtual bool Selectable { get; set; }
    //
    // Summary:
    //     Gets or sets the entity line type source.
    //
    // Remarks:
    //     You may need to call devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions)
    //     to see your changes.
    public virtual colorMethodType LineTypeMethod { get; set; }
    //
    // Summary:
    //     Gets or sets the entity selection status.
    //
    // Remarks:
    //     It's equivalent to call
devDept.Eyeshot.Entities.Entity.GetSelection(System.Collections.Generic.Stack{devDept.Eyeshot.Entities.BlockReference})
    //     or
devDept.Eyeshot.Entities.Entity.SetSelection(System.Boolean,System.Collections.Generic.Stack{devDept.Eyeshot.Entities.BlockReference})
    //     with no parents.
    public bool Selected { get; set; }
    //
    // Summary:
    //     Gets or sets the Autodesk common properties.
    public virtual AutodeskMiscProperties AutodeskProperties { get; set; }
    //
```

```
// Summary:
//     Gets or sets the entity color.
public virtual Color Color { get; set; }
//
// Summary:
//     Gets or sets the entity line weight source.
public virtual colorMethodType LineWeightMethod { get; set; }
//
// Summary:
//     Gets or sets the entity material name.
public virtual string MaterialName { get; set; }
//
// Summary:
//     Gets or sets the entity custom data.
//
// Remarks:
//     If the type implements System.ICloneable or if it is a value type (int, float,
//     struct, etc.) it will be cloned/copied when the Entity is cloned.
public virtual object EntityData { get; set; }
//
// Summary:
//     Gets or sets the entity line weight.
//
// Remarks:
//     This value corresponds to pixels when drawing on screen and to mm when exporting
//     to vectorial formats.
public virtual float LineWeight { get; set; }
//
// Summary:
//     Gets or sets the entity regeneration mode.
public virtual regenType RegenMode { get; set; }
```

```
        //
        // Summary:
        //     Gets or sets entity's 3D vertices.
        public virtual Point3D[] Vertices { get; set; }
        //
        // Summary:
        //     Gets the minimum 3D extent of the entity.
        public Point3D BoxMin { get; }
        //
        // Summary:
        //     Gets the maximum 3D extent of the entity.
        public Point3D BoxMax { get; }
        //
        // Summary:
        //     Gets the 3D extent of the entity.
        public Size3D BoxSize { get; }
        //
        // Summary:
        //     A bounding volume that can fit the entity in a different orientation than AABB.
        //     It can be an OrientedBoundingBox or an OrientedBoundingRect(for planar entities).
        //
devDept.Eyeshot.Entities.Entity.UpdateOrientedBoundingBox(devDept.Eyeshot.TraversalParams)
        //
        // Remarks:
        //     If the owner entity is transformed, its OrientedBounding (when already built)
        //     will follow its transformation.
        public OrientedBoundingRect OrientedBounding { get; set; }
        //
        // Summary:
        //     Gets or sets the entity color source.
        public virtual colorMethodType ColorMethod { get; set; }
```

```
//
// Summary:
//     Tells if the entity is being compiled.
protected bool Compiling { get; set; }
//
// Summary:
//     Gets or sets the nature of the entity.
protected internal entityNatureType entityNature { get; set; }
//
// Summary:
//     Gets or sets the halo drawing mode for selection.
protected internal virtual haloType HaloMode { get; }

//
// Summary:
//     Occurs when the devDept.Eyeshot.Entities.Entity.Visible changes.
public event VisibleChangedEventHandler VisibleChanged;

//
// Summary:
//     Returns the entity's line type.
//
// Parameters:
//   ent:
//     The entity
//
//   lineTypes:
//     Line types collection
//
//   parentTypeName:
//     The line type name of the parent entity (if it exists)
```

```
        //
        //   layers:
        //     The layers list
        //
        // Returns:
        //     The devDept.Eyeshot.LineType object.
        public static LineType GetEntityLineType(Entity ent, LineTypeKeyedCollection lineTypes, string
parentTypeName, LayerKeyedCollection layers);
        //
        // Summary:
        //     Tells if the quad defined by the vertices is inside the selection area defined
        //     by the edge list.
        //
        // Parameters:
        //   data:
        //     The frustum parameters
        //
        //   vertices:
        //     The vertices of the quad
        //
        // Returns:
        //     True if the quad intersects the selection area
        public static bool ThroughTriangleQuad(FrustumParams data, IList<Point3D> vertices);
        //
        // Summary:
        //     Tells if the quad defined by the vertices is inside the selection area defined
        //     by the screen polygon.
        //
        // Parameters:
        //   vertices:
        //     The vertices of the quad
```

```
        //
        //   data:
        //     The screen polygon parameters
        //
        // Returns:
        //     True if the quad intersects the screen polygon
        public static bool ThroughTriangleScreenPolygonQuad(IList<Point3D> vertices,
ScreenPolygonParams data);
        protected static bool ComputeBoundingBox(TraversalParams data, float[] pointArray, int
skipPoints, out Point3D boxMin, out Point3D boxMax);
        //
        // Summary:
        //     Computes the entity's bounding box.
        //
        // Parameters:
        //   data:
        //     Bounding box data
        //
        //   entityVertices:
        //     The vertices to consider in the computation.
        //
        //   boxMin:
        //     The bounding box minmum point
        //
        //   boxMax:
        //     The bounding box maximum point
        //
        // Returns:
        //     True if the bounding box is valid.
        protected static bool ComputeBoundingBox(TraversalParams data, IList<Point3D> entityVertices,
out Point3D boxMin, out Point3D boxMax);
```

```
protected static bool FrustumEdgesTriangleIntersection(Segment3D[] edgeList, Point3D v1,
Point3D v2, Point3D v3);

protected static Vector3D GetClosestMainAxis(Vector3D normDir);

protected static double GetOffsetDistance(Vector3D extDir, Vector3D amount, double
draftAngleInRadians);

//

// Summary:

//     Checks whether a triangle intersects or is inside a 2D screen polygon.

//

// Parameters:

//   v1:

//     The first triangle point

//

//   v2:

//     The second triangle point

//

//   v3:

//     The third triangle point

//

//   screenPolygon:

//     The 2D screen polygon

//

//   modelViewProj:

//     The modelview projection matrix

//

//   viewFrame:

//     The viewport bounds

//

//   blocks:

//     The blocks dictionary

//

// Returns:
```

```
    //     True if the triangle intersects or is inside the polygon
    protected static bool ThroughTriangleScreenPolygon(Point3D v1, Point3D v2, Point3D v3,
ScreenPolygonParams data);
    protected internal static void ComputeOffsetOnCameraAxes(Transformation modelView, float[]
pointArray, int count, PointF m1, PointF m2, ref PointF minQ, ref PointF maxQ, int skipPoints);
    protected internal static void ComputeOffsetOnCameraAxes(OffsetOnCameraAxesParams data,
IList<Point3D> vertices, int vertexCount);
    protected internal static bool InsideFrustumPoint(PlaneEquation[] frustum, Transformation
transform, IList<Point3D> points, int count);
    protected internal static bool InsideOrCrossingFrustumInternal(PlaneEquation[] frustum,
Transformation transform, IList<Point3D> points, int count, int increment);
    protected internal static bool InsideOrCrossingScreenPolygonInternal(ScreenPolygonParams
data, IList<Point3D> points, int count, int increment);
    protected internal static bool InsideOrCrossingScreenPolygonPoint(ScreenPolygonParams data,
IList<Point3D> points, int count);
    //
    // Summary:
    //     Propagates the attributes to the entity passed as parameter.
    //
    // Parameters:
    //   srcEntity:
    //     Source entity
    //
    //   destEntity:
    //     Destination entity
    //
    //   force:
    //     If false, propagates the attributes only if the attributes are ByParent
    protected internal static void PropagateAttributes(Entity srcEntity, Entity destEntity, bool force);
    //
    // Summary:
    //     Sets the entity color or material depending on the color mode.
    //
```

```
// Parameters:
//   data:
protected internal static void SetEntityColorForFace(DrawParams data, Color color);
//
// Summary:
//     Sets the entity color or material depending on the color mode.
//
// Parameters:
//   data:
protected internal static void SetEntityColorForSelection(DrawParams data);
//
// Summary:
//     Sets the selection color or material depending on the color mode.
//
// Parameters:
//   data:
protected internal static void SetSelectionColorForSelection(DrawParams data);
//
// Summary:
//     Clear the entity selectability status for the defined nested instance.
//
// Parameters:
//   parents:
//     The parents stack that define a nested instance
public void ClearSelectability(Stack<BlockReference> parents);
//
// Summary:
//     Clears the selectability status for all instances.
public void ClearSelectabilityForAllInstances();
//
// Summary:
```

```
//     Clears the selection status for all instances.
public void ClearSelectionForAllInstances();
//
// Summary:
//     Clear the entity visibility status for the defined nested instance.
//
// Parameters:
//   parents:
//     The parents stack that define a nested instance
public void ClearVisibility(Stack<BlockReference> parents);
//
// Summary:
//     Clears the visibility status for all instances.
public void ClearVisibilityForAllInstances();
//
// Summary:
//     Creates a deep copy of this entity.
//
// Returns:
//     The new entity object.
public abstract object Clone();
//
// Summary:
//     Compiles the graphic representation of this entity.
//
// Parameters:
//   data:
//     The data needed for compilation
//
// Remarks:
```

```
    //     Overrides of this method have to call the base or the
devDept.Eyeshot.Entities.Entity.InitGraphicsData(devDept.Graphics.RenderContextBase)
    //     first.
    public virtual void Compile(CompileParams data);
    //
    // Summary:
    //     Converts the Entity to its surrogate, for serialization purpose.
    //
    // Returns:
    //     The surrogate.
    public abstract EntitySurrogate ConvertToSurrogate();
    //
    // Summary:
    //     Copies the following attributes to this entity: devDept.Eyeshot.Entities.Entity.Color,
    //     devDept.Eyeshot.Entities.Entity.ColorMethod, devDept.Eyeshot.Entities.Entity.GroupIndex,
    //     devDept.Eyeshot.Entities.Entity.LineTypeName,
devDept.Eyeshot.Entities.Entity.LineTypeMethod,
    //     devDept.Eyeshot.Entities.Entity.LineTypeScale, devDept.Eyeshot.Entities.Entity.LineWeight,
    //     devDept.Eyeshot.Entities.Entity.LineWeightMethod,
devDept.Eyeshot.Entities.Entity.LayerName
    //     and devDept.Eyeshot.Entities.Entity.MaterialName.
    //
    // Parameters:
    //   source:
    //     The source entity
    public void CopyAttributes(Entity source);
    //
    // Summary:
    //     Copies the following attributes to this entity: devDept.Eyeshot.Entities.Entity.Visible,
    //     devDept.Eyeshot.Entities.Entity.Color, devDept.Eyeshot.Entities.Entity.ColorMethod,
    //     devDept.Eyeshot.Entities.Entity.LayerName and
devDept.Eyeshot.Entities.Entity.MaterialName
```

```
//     attributes.
//
// Parameters:
//   source:
//     The source entity
public void CopyAttributesFast(Entity source);
//
// Summary:
//     Cleans up graphics resources, like display lists, textures, etc.
public virtual void Dispose();
//
// Summary:
//     Returns a description of this entity.
//
// Parameters:
//   linearUnits:
//     The length units.
//
//   massUnits:
//     The mass units.
//
//   layers:
//     The layers collection.
//
//   materials:
//     The materials collection.
//
//   blocks:
//     The blocks collection.
//
// Returns:
```

```
        //     A multiline string.
        //
        // Remarks:
        //     All of the results about the mass properties are displayed using the units of
        //     measurement provided as input.
        public virtual string Dump(linearUnitsType linearUnits = linearUnitsType.Unitless, massUnitsType massUnits = massUnitsType.Unitless, LayerKeyedCollection layers = null, MaterialKeyedCollection materials = null, BlockKeyedCollection blocks = null);
        //
        // Summary:
        //     Returns a small set of points that gives an idea of the entity bounding box.
        //     Returns BoxMin and BoxMax if the entity has already been regenerated.
        //
        // Parameters:
        //   blocks:
        //     Blocks collection
        //
        //   layers:
        //     Layers collection
        //
        // Returns:
        //     An array of 3D points.
        public abstract Point3D[] EstimateBoundingBox(BlockKeyedCollection blocks, LayerKeyedCollection layers);
        //
        // Summary:
        //     Returns the entity color without considering the devDept.Eyeshot.Entities.Entity.MaterialName.
        //
        // Parameters:
        //   parent:
        //     The parent devDept.Eyeshot.Entities.BlockReference, can be null/Nothing.
```

```
        //
        //   layers:
        //     The layer collection
        //
        // Returns:
        //     The entity color.
        public Color GetColor(BlockReference parent, LayerKeyedCollection layers, Document doc =
null);
        //
        // Summary:
        //     Returns the entity line weight.
        //
        // Parameters:
        //   parent:
        //     The parent devDept.Eyeshot.Entities.BlockReference, can be null/Nothing.
        //
        //   layers:
        //     The layer collection
        //
        // Returns:
        //     The entity line weight.
        public double GetLineWeight(BlockReference parent, LayerKeyedCollection layers, Document
doc = null);
        //
        // Summary:
        //     Returns the entity color considering the devDept.Eyeshot.Entities.Entity.MaterialName.
        //
        // Parameters:
        //   parent:
        //     The parent devDept.Eyeshot.Entities.BlockReference, can be null/Nothing.
        //
        //   layers:
```

```
        //     The layer list
        //
        //   materials:
        //     The material collection
        //
        // Returns:
        //     The material name
        public Material GetMaterial(BlockReference parent, LayerKeyedCollection layers,
MaterialKeyedCollection materials);
        //
        // Summary:
        //     Populates a System.Runtime.Serialization.SerializationInfo instance with the
        //     data needed to serialize the target object.
        //
        // Parameters:
        //   info:
        //     A System.Runtime.Serialization.SerializationInfo instance that defines the serialized
        //     data.
        //
        //   context:
        //     A System.Runtime.Serialization.StreamingContext instance that contains the serialized
        //     data.
        public virtual void GetObjectData(SerializationInfo info, StreamingContext context);
        //
        // Summary:
        //     Gets the entity selectability status.
        //
        // Parameters:
        //   parents:
        //     The parents stack that define a nested instance
        //
```

```
// Returns:
//     True if the entity is selectable
//
// Remarks:
//     If the selectability for the given parents stack has not been set, returns the
//     devDept.Eyeshot.Entities.Entity.Selectable value.
public bool GetSelectability(Stack<BlockReference> parents);
//
// Summary:
//     Gets the entity selection status.
//
// Parameters:
//   parents:
//     The parents stack that define a nested instance
//
// Returns:
//     True if the entity is selected
public bool GetSelection(Stack<BlockReference> parents = null);
//
// Summary:
//     Gets the entity visibility status.
//
// Parameters:
//   parents:
//     The parents stack that define a nested instance
//
// Returns:
//     True if the entity is visible
//
// Remarks:
```

```
    //     If the visibility for the given parents stack has not been set, returns the
devDept.Eyeshot.Entities.Entity.Visible
    //     value.
    public bool GetVisibility(Stack<BlockReference> parents);
    //
    // Summary:
    //     Tells if there is at least an instance selectable.
    //
    // Returns:
    //     True if there is at least an instance selectable.
    public bool IsAnyInstanceSelectable();
    //
    // Summary:
    //     Tells if there is at least an instance selected.
    //
    // Returns:
    //     True if there is at least an instance selected.
    public bool IsAnyInstanceSelected();
    //
    // Summary:
    //     Tells if there is at least an instance visible.
    //
    // Returns:
    //     True if there is at least an instance visible.
    public bool IsAnyInstanceVisible();
    //
    // Summary:
    //     Tells if the entity is inside the frustum planes.
    //
    // Parameters:
    //   data:
```

```
//     The parameters data
//
// Returns:
//     True if the entity is inside the frustum
//
// Remarks:
//     The frustum planes can be obtained with devDept.Eyeshot.IViewport.GetCameraFrustum
public bool IsInFrustum(FrustumParams data);
//
// Summary:
//     Tells if the sphere surrounding the entity is inside the frustum planes.
//
// Parameters:
//   data:
//     The frustum data
//
//   center:
//     Center of the sphere (transformed by the scene transformation)
//
//   radius:
//     Radius of the sphere (scaled by the maximum scale applied by the scene transformation)
//
// Returns:
//     True if the sphere is inside the frustum planes.
//
// Remarks:
//     It's important when applying an on-the-fly transformation in the
devDept.Eyeshot.Entities.BlockReference.MoveTo(devDept.Eyeshot.DrawParams)
//     to override this method and call the base method with the center transformed
//     in the same way, in order to avoid undesired clipping.
public virtual bool IsInFrustum(FrustumParams data, Point3D center, double radius);
```

```
//
// Summary:
//     Tells if the instance referred by the stack of parents is selected.
//
// Parameters:
//   parents:
//     The parents stack that identifies the component instance
//
// Returns:
//     True if the instance is selected, false otherwise
[Obsolete("Use GetSelection(parents) instead")]
public virtual bool IsSelected(Stack<BlockReference> parents = null);
//
// Summary:
//     Returns true if all the entity fields contain reasonable information.
//
// Returns:
//     True if the entity is valid, false otherwise.
public virtual bool IsValid();
//
// Summary:
//     Computes the curve or surface tessellation.
//
// Parameters:
//   deviation:
//     The maximum deviation
//
// Remarks:
//     Can't be called on BlockReference entities. Call one of the other overloads for
//     BlockReference entities.
public virtual void Regen(double deviation);
```

```
//
// Summary:
//     This method is used for several purposes. For example in arcs and circles is
//     used to generate the curve's linear approximation, in meshes to compute normals
//     and edges and in Nurbs surfaces to generate the triangulation.
//
// Parameters:
//   data:
public virtual void Regen(RegenParams data);
//
// Summary:
//     Rotates the entity around an arbitray axis by the specified angle.
//
// Parameters:
//   angleInRadians:
//     The angle in radians
//
//   axis:
//     The rotation axis
public void Rotate(double angleInRadians, Vector3D axis);
//
// Summary:
//     Rotates the entity around an arbitrary axis by the specified angle.
//
// Parameters:
//   angleInRadians:
//     The angle in radians
//
//   axis:
//     The rotation axis' direction
//
```

```
//   center:
//     The rotation axis' origin
public virtual void Rotate(double angleInRadians, Vector3D axis, Point3D center);
//
// Summary:
//     Rotates the entity around an arbitray axis by the specified angle.
//
// Parameters:
//   angleInRadians:
//     The angle in radians
//
//   axisStart:
//     The rotation axis' start point
//
//   axisEnd:
//     The rotation axis' end point
public void Rotate(double angleInRadians, Point3D axisStart, Point3D axisEnd);
//
// Summary:
//     Scales the entity of the specified scale factor.
//
// Parameters:
//   fixedPoint:
//     Base point
//
//   sx:
//     Scale factor along X-axis
//
//   sy:
//     Scale factor along Y-axis
//
```

```
//   sz:
//     Scale factor along Z-axis
public virtual void Scale(Point3D fixedPoint, double sx, double sy, double sz = 1);
//
// Summary:
//     Scales the entity of the specified scale factor.
//
// Parameters:
//   fixedPoint:
//     Base point
//
//   factor:
//     Scale factor
public void Scale(Point3D fixedPoint, double factor);
//
// Summary:
//     Scales the entity of the specified scale factor.
//
// Parameters:
//   factor:
//     Scale factor
public void Scale(double factor);
//
// Summary:
//     Scales the entity of the specified scale factor.
//
// Parameters:
//   sx:
//     Scale factor along X-axis
//
//   sy:
```

```
//     Scale factor along Y-axis
//
//   sz:
//     Scale factor along Z-axis
public void Scale(double sx, double sy, double sz = 1);
//
// Summary:
//     Scales the entity of the specified scale factor.
//
// Parameters:
//   sv:
//     Scale vector
public void Scale(Vector3D sv);
//
// Summary:
//     Line weight changer.
//
// Parameters:
//   renderContext:
//     The render context
//
//   lineWeight:
//     The line weight resolved from attributes propagation
public virtual void SetLineWeight(RenderContextBase renderContext, float lineWeight);
//
// Summary:
//     Sets the entity selectability status.
//
// Parameters:
//   status:
//     The selectability status
```

```
//
//   parents:
//     The parents stack, stored in the entity, that define a nested instance
//
// Remarks:
//     To change the status of a single instance use a full stack of parents starting
//     from the root level.
public void SetSelectability(bool status, Stack<BlockReference> parents);
//
// Summary:
//     Sets the entity selection status.
//
// Parameters:
//   status:
//     The selection status
//
//   parents:
//     The parents stack that define a nested instance
public void SetSelection(bool status, Stack<BlockReference> parents = null);
//
// Summary:
//     Sets the entity visibility status.
//
// Parameters:
//   status:
//     The visibility status
//
//   parents:
//     The parents stack, stored in the entity, that define a nested instance
//
// Remarks:
```

```
//     To change the status of a single instance use a full stack of parents starting
//     from the root level. To change the status of all specified instances of the entity
//     inside a given block use a partial stack of parents starting from the block level.
//     The parents stack is not cloned, it's stored in the entity and used to check
//     the visibility.
public void SetVisibility(bool status, Stack<BlockReference> parents);
//
// Summary:
//     Transforms all the entity's vertices by the specified transformation.
//
// Parameters:
//   transform:
//     The transformation to be applied
//
// Remarks:
//     You need to call devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions)
//     to see the effect of this command.
public virtual void TransformBy(Transformation transform);
//
// Summary:
//     Translates the entity.
//
// Parameters:
//   dx:
//     Amount in X
//
//   dy:
//     Amount in Y
//
//   dz:
//     Amount in Z
```

```
public virtual void Translate(double dx, double dy, double dz = 0);
//
// Summary:
//     Translates the entity.
//
// Parameters:
//   v:
//     Displacement vector
public void Translate(Vector3D v);
//
// Summary:
//     Updates the entity's bounding box.
//
// Parameters:
//   data:
//     Traversal data
public void UpdateBoundingBox(TraversalParams data);
//
// Summary:
//     Updates the entity's oriented bounding box or build it if not present.
//
// Parameters:
//   data:
//     Traversal data
//
// Remarks:
//     It needs to call Regen() method first(except for Mesh, Solid, FastPointCloud,
//     Joint, Bar and LinearPath). If a transformation is applied to the entity after
//     regen (except for non uniform scale transformation) the OBB is already update
//     (if it's not null).
public virtual void UpdateOrientedBoundingBox(TraversalParams data);
```

```
//
// Summary:
//     Updates the entity's oriented bounding box or build it if not present.
//
// Parameters:
//   data:
//     Traversal data
//
//   keepCurrent:
//     When true, it avoid to overrides the current OBB if not needed
//
// Remarks:
//     It needs to call Regen() method first(except for Mesh, Solid, FastPointCloud,
//     Joint, Bar and LinearPath). If a transformation is applied to the entity after
//     regen (except for non uniform scale transformation) the OBB is already update
//     (if it's not null).
public virtual void UpdateOrientedBoundingBox(TraversalParams data, bool keepCurrent);
//
// Summary:
//     For internal use only.
protected void AddVerticesToXElement(XElement parent);
//
// Summary:
//     Compiles the graphics representation of the pattern of wireframe entities.
//
// Parameters:
//   data:
//     The parameters
protected virtual void CompilePattern(CompileParams data);
//
// Summary:
```

```
//     Compiles the graphics representation of wireframe entities.
//
// Parameters:
//   data:
protected void CompileWire(CompileParams data);
//
// Summary:
//     Internal method that draws the entity.
//
// Parameters:
//   context:
//     The render context
//
//   myParams:
//     The parameters
//
// Remarks:
//     This method may be used both for compiling the Entity (OpenGL and Direct3D) and
//     to draw the Entity (in Direct3D only, for some complex entities like
devDept.Eyeshot.Entities.Dimension)
protected virtual void DrawEntity(RenderContextBase context, object myParams);
//
// Summary:
//     Draws a wireframe entity, resolving the devDept.Eyeshot.Entities.Entity.LineTypeName
//     for
devDept.Eyeshot.Entities.Entity.LineTypeMethoddevDept.Eyeshot.Entities.colorMethodType.byParen
t.
//
// Parameters:
//   data:
protected virtual void DrawWire(DrawParams data);
//
```

```
// Summary:
//     Draws the Wireframe entity.
//
// Parameters:
//   context:
//
//   myParams:
//
// Remarks:
//     Used to compile the wireframe entities. Override this to customize the wireframe
//     entities drawing.
protected virtual void DrawWireEntity(RenderContextBase context, object myParams);
protected virtual void DrawWithPattern(RenderContextBase renderContext, object myParams);
//
// Summary:
//     Tells if must evaluate the intersection of the edges with the frustum.
//
// Parameters:
//   data:
protected virtual bool EvaluateIntersectEdges(FrustumParams data);
//
// Summary:
//     Tells if must evaluate the intersection of the triangles with the frustum.
//
// Parameters:
//   data:
protected virtual bool EvaluateIntersectTriangles(FrustumParams data);
//
// Summary:
//     Gets the entity normal length.
//
```

```
        // Returns:
        //     The normal length
        protected double GetNormalLength();
        //
        // Summary:
        //     Initialize the graphics data needed to
devDept.Eyeshot.Entities.Entity.Compile(devDept.Eyeshot.CompileParams)
        //     the entity.
        //
        // Parameters:
        //   renderContext:
        //     The current render context
        protected virtual void InitGraphicsData(RenderContextBase renderContext);
        //
        // Summary:
        //     Write in Obj file format as set of lines connecting vertices.
        protected void WriteObjAsLines(ref int objectCount, TextWriter objTextWriter, ref int vCount, ref
int tcCount, ref int nCount, TextWriter mtlTextWriter, LayerKeyedCollection layers, string dir,
MaterialKeyedCollection materials, double tol, double maxAngle, BlockKeyedCollection blocks);
        protected void WriteUsemtl(string materialName, TextWriter objTextWriter);
        protected internal virtual bool AllVerticesInFrustum(FrustumParams data);
        //
        // Summary:
        //     Tells if an entity is fully contained inside a polygon defined in screen coordinates.
        //
        // Parameters:
        //   data:
        //     Screen polygon data
        //
        // Returns:
        //     True if all the vertices of the entity are contained in the screen polygon.
        protected internal virtual bool AllVerticesInScreenPolygon(ScreenPolygonParams data);
```

```
        //
        // Summary:
        //     Gives a chance to derived classes to change the entity position/rotation at each
        //     timer tick.
        //
        // Remarks:
        //     This method is executed in a background thread because the timer used to run
        //     animation is a System.Threading.Timer
        protected internal virtual void Animate(int frameNumber);
        //
        // Summary:
        //     Combines the entity's bounding box with the given bounding box.
        //
        // Parameters:
        //   transform:
        //     The transformation applied to the entity
        //
        //   boxMin:
        //     The minimum point of the bounding box
        //
        //   boxMax:
        //     The maximum point of the bounding box.
        //
        // Returns:
        //     True if successful, false otherwise.
        protected internal virtual bool CombineBoundingBox(Transformation transform, Point3D
boxMin, Point3D boxMax);
        //
        // Summary:
        //     Computes the entity's bounding box.
        //
```

```
        // Parameters:
        //   data:
        //     Bounding box data
        //
        //   boxMin:
        //     The bounding box minimum point
        //
        //   boxMax:
        //     The bounding box maximum point
        //
        // Returns:
        //     True if the bounding box is valid.
        protected internal virtual bool ComputeBoundingBox(TraversalParams data, out Point3D
boxMin, out Point3D boxMax);
        //
        // Summary:
        //     Computes the intersection of the lines passing from the vertices and oriented
        //     like the frustum planes with the X and Y axes of the camera.
        //
        // Parameters:
        //   data:
        //     Camera offset data
        protected internal virtual void ComputeOffsetOnCameraAxes(OffsetOnCameraAxesParams data);
        //
        // Summary:
        //     Draws the entity.
        //
        // Parameters:
        //   data:
        protected internal virtual void Draw(DrawParams data);
        //
```

```
// Summary:
//     Draws a small arrow to show the entity direction if Workspace.ShowCurveDirection
//     is true.
protected internal virtual void DrawDirection(DrawParams data);
//
// Summary:
//     Draws the entity's edges.
protected internal virtual void DrawEdges(DrawParams data);
//
// Summary:
//     Draws the entity in fast inaccurate transparency mode.
//
// Parameters:
//   data:
protected internal virtual void DrawFast(DrawParams data);
//
// Summary:
//     Draws entity in devDept.Eyeshot.displayType.FlatdevDept.Eyeshot.IViewport.DisplayMode.
//
// Parameters:
//   data:
//     Draw parameters
protected internal virtual void DrawFlat(DrawParams data);
//
// Summary:
//     Draws the entity in flat and fast inaccurate transparency mode.
//
// Parameters:
//   data:
protected internal virtual void DrawFlatFast(DrawParams data);
//
```

```
// Summary:
//     Draws entity selected in
devDept.Eyeshot.displayType.FlatdevDept.Eyeshot.IViewport.DisplayMode.
protected internal virtual void DrawFlatSelected(DrawParams drawParams);
//
// Summary:
//     Draws the entity without specifing any color.
//
// Parameters:
//   data:
protected internal virtual void DrawForSelection(GfxDrawForSelectionParams data);
//
// Summary:
//     Draws the entity edges in false-colors (for some kinds of entities only).
//
// Parameters:
//   data:
protected internal virtual void DrawForSelectionEdges(GfxDrawForSelectionParams data);
//
// Summary:
//     Draws the entity faces in false-colors (for some kinds of entities only).
//
// Parameters:
//   data:
protected internal virtual void DrawForSelectionFaces(GfxDrawForSelectionParams data);
//
// Summary:
//     Draws the entity curves in false-colors (for some kinds of entities only).
//
// Parameters:
//   data:
```

```
        protected internal virtual void DrawForSelectionSketchCurves(GfxDrawForSelectionParams
data);
        //
        // Summary:
        //     Draws the entity points in false-colors (for some kinds of entities only).
        //
        // Parameters:
        //   data:
        protected internal virtual void DrawForSelectionSketchPoints(GfxDrawForSelectionParams data);
        //
        // Summary:
        //     Draws the entity contours in false-colors (for some kinds of entities only).
        //
        // Parameters:
        //   data:
        protected internal virtual void DrawForSelectionSubContours(GfxDrawForSelectionParams data);
        //
        // Summary:
        //     Draws the entity subCurves in false-colors (for some kinds of entities only).
        //
        // Parameters:
        //   data:
        protected internal virtual void DrawForSelectionSubCurves(GfxDrawForSelectionParams data);
        //
        // Summary:
        //     Draws the entity vertices in false-colors (for some kinds of entities only).
        //
        // Parameters:
        //   data:
        protected internal virtual void DrawForSelectionVertices(GfxDrawForSelectionParams data);
        //
```

```
// Summary:
//     Draws entity as wires without specifing any color.
//
// Parameters:
//   data:
protected internal virtual void DrawForSelectionWireframe(GfxDrawForSelectionParams data);
//
// Summary:
//     Draws the entity planar shadow.
protected internal virtual void DrawForShadow(RenderParams data);
//
// Summary:
//     Draw the entity in the devDept.Eyeshot.displayType.HiddenLines display mode.
//
// Parameters:
//   data:
protected internal virtual void DrawHiddenLines(DrawParams data);
//
// Summary:
//     Draw the entity in the devDept.Eyeshot.displayType.HiddenLines display mode in
//     fast inaccurate transparency mode.
//
// Parameters:
//   data:
protected internal virtual void DrawHiddenLinesFast(DrawParams data);
protected internal virtual void DrawHiddenLinesMaterial(RenderParams data);
protected internal virtual void DrawHiddenLinesMaterialFast(RenderParams data);
//
// Summary:
//     Draws the entity iso curves. For Mesh entities this method draws internal wires.
protected internal virtual void DrawIsocurves(DrawParams data);
```

```
//
// Summary:
//     Draws the isocurves in Flat display mode.
//
// Parameters:
//   data:
//     The draw parameters
//
// Remarks:
//     Only the devDept.Eyeshot.Entities.Surface entities draws them.
protected internal virtual void DrawIsocurvesForFlat(DrawParams data);
//
// Summary:
//     Draws entity's normal vectors.
//
// Parameters:
//   data:
//     The draw data
protected internal virtual void DrawNormals(DrawParams data);
//
// Summary:
//     Draws extra things on screen, like the vertex indices (if IViewport.ShowVertexIndices
//     is true).
//
// Parameters:
//   camera:
//     The viewport camera
//
//   values:
//     The depth values
//
```

```
//   stride:
//     The stride for each row of values
//
//   viewFrame:
//     The viewport borders
//
//   digitTextures:
//     The textures of the 0..9 digits
//
//   leftBorder:
//     The viewport left border minus the vertex size
//
//   rightBorder:
//     The viewport right border minus the vertex size
//
//   bottomBorder:
//     The viewport bottom border minus the vertex size
//
//   topBorder:
//     The viewport top border minus the vertex size
//
//   vertexCount:
//     The number of vertices
protected internal void DrawOnScreen(DrawOnScreenParams myParams, int vertexCount);
//
// Summary:
//     Draws extra things on screen, like the vertex indices (if IViewport.ShowVertexIndices
//     is true).
//
// Parameters:
//   drawOnScreenParams:
```

```
        protected internal virtual void DrawOnScreen(DrawOnScreenParams drawOnScreenParams);
        //
        // Summary:
        //     Draws extra things on screen, like the vertex Numbers (if IViewport.ShowVertexIndices
        //     is true) in wireframe dispaly mode.
        //
        // Parameters:
        //   myParams:
        //
        //   vertexCount:
        //     The number of vertices
        protected internal void DrawOnScreenWireframe(DrawOnScreenWireframeParams myParams,
int vertexCount);
        //
        // Summary:
        //     Draws extra things on screen, like the vertex Numbers (if IViewport.ShowVertexIndices
        //     is true) in wireframe display mode.
        //
        // Parameters:
        //   myParams:
        protected internal virtual void DrawOnScreenWireframe(DrawOnScreenWireframeParams
myParams);
        //
        // Summary:
        //     Draws entity selected.
        protected internal virtual void DrawSelected(DrawParams data);
        //
        // Summary:
        //     Draws the entity selected vertices.
        //
        // Remarks:
        //     For devDept.Eyeshot.Entities.Brep only
```

```
protected internal virtual void DrawSelectedVertices(DrawParams data);
//
// Summary:
//     Draws Silhouettes.
//
// Parameters:
//   data:
protected internal virtual void DrawSilhouettes(DrawSilhouettesParams data);
//
// Summary:
//     Draws entity's vertices.
protected internal virtual void DrawVertices(DrawParams data);
//
// Summary:
//     Draws entity as wires.
//
// Parameters:
//   data:
protected internal virtual void DrawWireframe(DrawParams data);
//
// Summary:
//     Draws entity as selected wires.
protected internal virtual void DrawWireframeSelected(DrawParams data);
//
// Summary:
//     Gets tessellation vertices as float coordinates of the entity.
//
// Parameters:
//   data:
//
//   verticesCoords:
```

```
        //
        // Remarks:
        //     The output vertices coords are used to build ConvexHulls, when Design.ZoomFitMode
        //     != devDept.Eyeshot.zoomFitType.Standard
        protected internal virtual bool GetAllVertices(TraversalParams data, out IList<float>
verticesCoords);

        protected internal virtual bool InsideOrCrossingFrustum(FrustumParams data);

        protected internal virtual bool InsideOrCrossingScreenPolygon(ScreenPolygonParams data);

        //
        // Summary:
        //     Tells if the entity is inside or crossing the given planes and edges.
        //
        // Returns:
        //     True if the entity is selected
        protected internal virtual bool IsCrossing(FrustumParams data);

        //
        // Summary:
        //     Tells if an entity is fully or partially contained inside a polygon defined in
        //     screen coordinates.
        //
        // Parameters:
        //   data:
        //     Screen polygon data
        //
        // Returns:
        //     True if at least one of the vertices of the entity is contained in the screen
        //     polygon.
        protected internal virtual bool IsCrossingScreenPolygon(ScreenPolygonParams data);

        //
        // Summary:
        //     Checks if an entity is small and can be skipped during the drawing.
```

```
        //
        // Parameters:
        //   data:
        //     The data
        //
        // Returns:
        //     True, if the entity is small classified as small.
        protected internal virtual bool IsSmall(IsSmallParams data);
        //
        // Summary:
        //     Check if the entity is visible.
        //
        // Parameters:
        //   parents:
        //     The parents visibility stack
        //
        //   layers:
        //     The layer collection
        //
        //   attributeReferenceMode:
        //     The AttributeReference mode
        protected internal virtual bool IsVisible(Stack<BlockReference> parents, LayerKeyedCollection
layers, attributeReferenceVisibilityType attributeReferenceMode);
        //
        // Summary:
        //     Check if the entity is visible.
        //
        // Parameters:
        //   parents:
        //     The parents stack
        //
```

```
        //   layers:
        //     The layers collection
        //
        //   attributeReferenceMode:
        //     The AttributeReference visibility mode
        protected internal virtual bool IsVisibleAndInFrustum(Stack<BlockReference> parents,
LayerKeyedCollection layers, attributeReferenceVisibilityType attributeReferenceMode);
        //
        // Summary:
        //     Renders the entity.
        //
        // Parameters:
        //   data:
        protected internal virtual void Render(RenderParams data);
        //
        // Summary:
        //     Renders the entity in fast inaccurate transparency mode.
        //
        // Parameters:
        //   data:
        protected internal virtual void RenderFast(RenderParams data);
        //
        // Summary:
        //     Tells if the entity has internal parts selected
        protected internal virtual bool SelectedInternal();
        protected internal virtual void SetLineWeightForEdges(DrawParams data);
        protected internal virtual void SetLineWeightForSilhouettes(DrawSilhouettesParams data);
        //
        // Summary:
        //     Sets a Shader before drawing the entity.
        //
```

```
        // Parameters:
        //   data:
        //     data for entity drawing
        protected internal virtual void SetShader(DrawParams data);
        protected internal virtual bool ThroughTriangle(FrustumParams data);
        protected internal virtual bool ThroughTriangleScreenPolygon(ScreenPolygonParams data);
        protected internal void TransformAllVertices(Point3D[] vertices, Transformation t, bool
translationOnly = false, bool containsScaling = true);
        protected internal virtual void TransformAllVertices(Transformation t, bool translationOnly =
false, bool containsScaling = true);
        protected internal virtual void TransformAllVerticesRecursive(Transformation t);
        protected internal void UpdateBoundingBoxSphere();
        protected internal void WriteUsemtl(int objectCount, TextWriter objTextWriter, TextWriter
mtlTextWriter, LayerKeyedCollection layers);
    }
}

#region Assembly devDept.Eyeshot.Control.Win.v2023, Version=2023.1.353.0, Culture=neutral,
PublicKeyToken=f3cd437f0d8061b5
// C:\Program Files\devDept Software\Eyeshot
2023\Bin\net472\devDept.Eyeshot.Control.Win.v2023.dll
#endregion

using devDept.Eyeshot.Control.Mouse3D;
using devDept.Eyeshot.Control.MultiTouch;
using devDept.Eyeshot.Entities;
using devDept.Geometry;
using devDept.Graphics;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
```

```
using System.Drawing.Imaging;
using System.Drawing.Printing;
using System.IO;
using System.Reflection;
using System.Windows.Forms;

namespace devDept.Eyeshot.Control
{
   //
   // Summary:
   //     Base abstract class for Eyeshot controls.
   public abstract class Workspace : WorkspaceBase, IWorkspace, IWorkspaceInternal,
ISupportWorkManager
   {
      //
      // Summary:
      //     For internal use only.
      protected const string ROOT_SCENE_PREFIX = "Root Scene";
      public Dictionary<shaderType, IShaderTechnique> StandardShaders;
      protected BackgroundWorker backgroundWorker;
      protected ShortcutKeysSettings shortcutKeys;
      //
      // Summary:
      //     Framerate per second counter.
      protected float fps;
      //
      // Summary:
      //     Hold the focus status.
      protected internal bool hasFocus;

      //
```

```
        // Summary:
        //     Empty constructor.
        protected Workspace();


        //
        // Summary:
        //     Ambient Occlusion settings, shared by all viewports.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public AmbientOcclusionSettings AmbientOcclusion { get; set; }
        //
        // Summary:
        //     If true, curve direction is displayed.
        [Category("Workspace")]
        [Description("If true, curve direction is displayed.")]
        public bool ShowCurveDirection { get; set; }
        //
        // Summary:
        //     Gets or sets the style used by the devDept.Eyeshot.Control.ToolBar and
devDept.Eyeshot.Control.Workspace.ProgressBar
        //     buttons.
        //
        // Remarks:
        //     You need to call the Viewport.CompileUserInterfaceElements() to see the effect
        //     of this command.
        [Category("Workspace - User Interface")]
        public ButtonSettings ButtonStyle { get; set; }
        //
        // Summary:
        //     The interval, in milliseconds, of CameraChanged events firing.
        [Browsable(false)]
```

```
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public int CameraChangedFrequency { get; set; }
//
// Summary:
//     Gets or sets the Multitouch settings.
[Category("Workspace - Input Devices")]
[Description("Multitouch settings.")]
public MultiTouchSettings MultiTouch { get; set; }
//
// Summary:
//     Returns true if Full Screen Anti-Aliasing is available.
//
// Remarks:
//     This value is written during the control's initialization, so it must be read
//     after the control is correctly loaded.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool IsAntiAliasingAvailable { get; }
//
// Summary:
//     Returns true if OpenGL hardware acceleration is currently in use.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool IsHardwareAccelerated { get; }
//
// Summary:
//     Enables or disables the screen-space ambient occlusion effect.
//
// Remarks:
//     Screen-space ambient occlusion is available only in devDept.Eyeshot.displayType.Rendered
//     display mode.
```

```
//     Use devDept.Eyeshot.Control.Workspace.AmbientOcclusion to get access to additional
//     settings.
[Category("Ambient Occlusion")]
[Description("Ambient Occlusion settings, shared by all viewports.")]
public bool EnableAmbientOcclusion { get; set; }
//
// Summary:
//     Forces accelerated hardware modes. By default, Eyeshot automatically disables
//     hardware acceleration on some embedded GPUs to obtain the maximum level of graphical
//     output consistency between all hardware. In these cases, enabling this property
//     will forcibly activate hardware acceleration, though the resulting graphics output
//     may be unpredictable.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool ForceHardwareAcceleration { get; set; }
//
// Summary:
//     Indicates whether the best adapter is available for the renderer initialization
//     or not.
//
// Remarks:
//     This value is written during the control's initialization, so it must be read
//     after the control is correctly loaded.
//     This property is not valid when the devDept.Eyeshot.Control.WorkspaceBase.Renderer
//     is devDept.Eyeshot.Control.rendererType.OpenGL.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool IsBestAdapterAvailable { get; }
//
// Summary:
//     Gets the devDept.Eyeshot.Document.
```

```
        public Document Document { get; }
        //
        // Summary:
        //     Gets or sets the selection settings, shared by all viewports.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public SelectionSettings Selection { get; set; }
        //
        // Summary:
        //     Gets or sets the factor applied to the line weight for the selected wireframe
        //     entities or devDept.Eyeshot.Entities.Brep.Edges.
        [Category("Workspace - Selection")]
        [Description("The line weight scale factor for selected entities, shared by all viewports.")]
        [Obsolete("Use SelectionSettings Workspace.Selection instead.")]
        public float SelectionLineWeightScaleFactor { get; set; }
        //
        // Summary:
        //     3D mouse settings.
        public Mouse3DSettings Mouse3D { get; set; }
        //
        // Summary:
        //     Gets or sets the color for the dynamic selection, shared by all viewports.
        //
        // Remarks:
        //     The devDept.Eyeshot.Material.Ambient, devDept.Eyeshot.Material.Specular and
devDept.Eyeshot.Material.Shininess
        //     components are taken from the devDept.Eyeshot.Control.Design.DefaultMaterial.
        [Category("Workspace - Selection")]
        [Description("Color of dynamic selection, shared by all viewports.")]
        [Obsolete("Use SelectionSettings Workspace.Selection instead.")]
        public Color SelectionColorDynamic { get; set; }
```

```
//
// Summary:
//     Gets the EntityList of the devDept.Eyeshot.Control.Workspace.CurrentBlock. This
//     collection contains the entities displayed in the viewport.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public EntityList Entities { get; }
//
// Summary:
//     Gets or sets the frame per second rate text visibility status.
//
// Remarks:
//     "WF" stands for WindowsForms
[Category("Workspace")]
[Description("Frame per second rate text visibility status.")]
public bool ShowFps { get; set; }
//
// Summary:
//     Gets or sets the wait cursor mode. Controls if and when the wait cursor is displayed.
[Category("Workspace")]
[Description("Wait cursor mode. Controls if and when the wait cursor is displayed.")]
public waitCursorType WaitCursorMode { get; set; }
//
// Summary:
//     If true, OpenGL accelerated hardware modes are requested during viewport initialization.
//
// Remarks:
//     Has no effect at runtime.
[Category("Workspace - Initialization")]
[Description("If true, OpenGL accelerated hardware modes are requested during viewport
initialization. Set to false if the OpenGL hardware acceleration causes display problems.")]
```

```
        public bool AskForHardwareAcceleration { get; set; }

        //
        // Summary:
        //     If true, The Direct3D feature level 9_3 is requested during viewport initialization
        //     instead of the highest feature level available.
        //
        // Remarks:
        //     Used for compatibility problems with old cards.
        [Category("Workspace - Initialization")]
        [Description("Asks for the Direct3D feature level 9_3.")]
        public bool AskForDirect3DLevel9_3 { get; set; }

        //
        // Summary:
        //     If true, Full Screen Anti-Aliasing modes are requested during viewport initialization.
        //     The number of samples can be set with the
devDept.Eyeshot.Control.Workspace.AntiAliasingSamples
        //     property.
        //
        // Remarks:
        //     Has no effect at runtime for the OpenGL renderer.
        [Category("Workspace - Initialization")]
        [Description("If true, Full Screen Anti-Aliasing modes are requested during viewport
initialization. The number of samples can be set with the AntiAliasingSamples property.")]
        public bool AskForAntiAliasing { get; set; }

        //
        // Summary:
        //     Gets or sets a value indicating if full screen anti-aliasing is enabled.
        [Category("Workspace")]
        [Description("Gets or sets a value indicating if full screen anti-aliasing is enabled. Available only if
AskForAntiAliasing property is set to true in the constructor.")]
        public bool AntiAliasing { get; set; }

        //
```

```
        // Summary:
        //     Gets or sets the AttributeReferences visibility mode.
        [Category("Workspace")]
        [Description("Gets or sets the AttributeReferences visibility mode.")]
        public attributeReferenceVisibilityType AttributeReferenceVisibilityMode { get; set; }
        //
        // Summary:
        //     Gets the company responsible for this GL implementation. This name does not change
        //     from release to release.
        //
        // Remarks:
        //     OpenGL only.
        [Category("Workspace - OpenGL")]
        [Description("The company responsible for this GL implementation. This name does not change
from release to release.")]
        public string RendererVendor { get; }
        //
        // Summary:
        //     Gets OpenGL Shading Language version.
        //
        // Remarks:
        //     OpenGL only.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Version ShadingLanguageVersion { get; }
        //
        // Summary:
        //     Gets or sets the color of selected entities, shared by all viewports.
        //
        // Remarks:
        //     The devDept.Eyeshot.Material.Ambient, devDept.Eyeshot.Material.Specular and
devDept.Eyeshot.Material.Shininess
```

```
        //     components are taken from the devDept.Eyeshot.Control.Design.DefaultMaterial.
        [Category("Workspace - Selection")]
        [Description("Color of selected entities, shared by all viewports.")]
        [Obsolete("Use SelectionSettings Workspace.Selection instead.")]
        public Color SelectionColor { get; set; }
        //
        // Summary:
        //     Temporary Entity collection. This collection contains the entities displayed
        //     in the viewport on top of the others as temporary entities.
        //
        // Remarks:
        //     Only IFace and ICurve types are supported into this collection. Each entity added
        //     to this list needs to the regenerated first.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public TempEntityList TempEntities { get; set; }
        //
        // Summary:
        //     Gets or sets the line types collection.
        //
        // Exceptions:
        //   T:devDept.EyeshotException:
        //     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Workspace.Document.
        //
        // Remarks:
        //     "ByBlock", "ByLayer", "Continuous" are reserved names and should not be used
        //     to avoid issues when exporting.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public LineTypeKeyedCollection LineTypes { get; set; }
```

```
        //
        // Summary:
        //     Gets the root block.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Block RootBlock { get; }
        //
        // Summary:
        //     If true, the devDept.Eyeshot.Control.rendererType.OpenGL renderer uses Phong
        //     shading for better lighting, has the
devDept.Eyeshot.Control.DisplayModeSettingsRendered.PlanarReflections
        //     with a fading out effect and better performances for the
devDept.Graphics.shadowType.Realistic
        //     shadows.
        //
        // Remarks:
        //     With the devDept.Eyeshot.Control.rendererType.Direct3D renderer it doesn't have
        //     any effect.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool UseShaders { get; set; }
        //
        // Summary:
        //     When true, the Accurate Transparency mode is activated. Accurate Transparency
        //     provides slower but more accurate semi-transparent object drawing with support
        //     for non-concave objects and for objects intersection.
        //
        // Remarks:
        //     For better rendering of transparent entities with
devDept.Eyeshot.Control.Workspace.AccurateTransparency
        //     turned off, override the
devDept.Eyeshot.Control.Workspace.DrawViewport(devDept.Eyeshot.DrawSceneParams)
```

```
    //     and call the
devDept.Eyeshot.Control.Workspace.SortEntitiesForTransparency(devDept.Eyeshot.Control.Viewport,
System.Collections.Generic.IList{devDept.Eyeshot.Entities.Entity}).
    //     It does not works with selection inside components or of the faces.
    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public bool AccurateTransparency { get; set; }
    //
    // Summary:
    //     Gets or sets the FastZPR settings. When FastZPR is active a simplified representation
    //     of the current geometry is drawn during dynamic movements (Zoom/Pan/Rotate).
    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public TurboSettings Turbo { get; set; }
    //
    // Summary:
    //     Gets the number of loaded character definitions.
    //
    // Returns:
    //     The character definitons count.
    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public int AllocatedCharDefs { get; }
    //
    // Summary:
    //     True if the OpenBlock is the RootBlock, false otherwise.
    [Browsable(false)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
    public bool IsOpenRootLevel { get; }
    //
    // Summary:
    //     Gets the current BlockReference.
```

```
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public BlockReference CurrentBlockReference { get; }
        //
        // Summary:
        //     Gets the current block if a BlockReference is set as current, the
devDept.Eyeshot.Control.Workspace.OpenBlock
        //     otherwise.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Block CurrentBlock { get; }
        //
        // Summary:
        //     When true the frame buffer objects are used. Affects the quality of Realistic
        //     shadows. For debugging purpose only.
        //
        // Remarks:
        //     When using Direct3D renderer, disabling this will disable also the realistic
        //     shadows.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool UseFrameBufferObject { get; set; }
        //
        // Summary:
        //     Gets or sets the settings for the magnifying glass displayed under the mouse
        //     cursor.
        [Category("Workspace")]
        [Description("The colors used to draw the selection box and polygon.")]
        public MagnifyingGlassSettings MagnifyingGlass { get; set; }
        //
        // Summary:
```

```
        //     Gets or sets the colors used to draw the selection box and polygon.
        [Category("Workspace - Selection")]
        [Description("The colors used to draw the selection box and polygon.")]
        public SelectionBoxColorsSettings SelectionBoxColors { get; set; }
        //
        // Summary:
        //     Gets a value indicating if an error occurred during paint.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool ErrorInPaint { get; }
        //
        // Summary:
        //     Gets the open block.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public Block OpenBlock { get; }
        //
        // Summary:
        //     Gets or sets the blocks collection.
        //
        // Exceptions:
        //   T:devDept.EyeshotException:
        //     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Workspace.Document.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public BlockKeyedCollection Blocks { get; set; }
        //
        // Summary:
        //     Gets the accumulated devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     transformation (including its parents transformations).
```

```
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public Transformation CurrentTransformation { get; }
//
// Summary:
//     Indicates whether the Isolate feature is currently available.
//
// Remarks:
//     The feature allows to focus on a part of the model while still seeing the rest
//     of the scene in a transparent. It may be not available due to hardware limitations
//     or when the devDept.Eyeshot.Control.Design.MinimumFramerate is disabled.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public bool IsolateAvailable { get; }
//
// Summary:
//     Gets or sets the keyboard shortcuts.
[Category("Workspace")]
[Description("Keyboard shortcuts.")]
public ShortcutKeysSettings ShortcutKeys { get; set; }
//
// Summary:
//     Gets or sets the pick box size in pixel units.
[Category("Workspace - Selection")]
public int PickBoxSize { get; set; }
//
// Summary:
//     If true, the selection ActionModes work as if the Control key was pressed, selecting
//     multiple entities.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
```

```
public bool MultipleSelection { get; set; }
//
// Summary:
//     The cursor types dictionary.
//
// Remarks:
//     If devDept.Eyeshot.Control.WorkspaceBase.Renderer is different from rendererType.Native,
//     use the WorkspaceBase.SetCursor(cursorType, Stream) instead (WPF Eyeshot control
//     only)
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public Dictionary<cursorType, Cursor> CursorTypes { get; }
//
// Summary:
//     Gets or sets viewport font (WinForms).
[Description("Affects the font of the fps string.")]
public override Font Font { get; set; }
//
// Summary:
//     For internal use only. Gets the instance id for the logging.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public string InstanceId { get; }
//
// Summary:
//     Gets or sets the text styles collection.
//
// Exceptions:
//   T:devDept.EyeshotException:
//     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Workspace.Document.
```

```
        //
        // Remarks:
        //     The "Default" text style is added if missing.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public TextStyleKeyedCollection TextStyles { get; set; }
        //
        // Summary:
        //     Gets or sets the active viewport action.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public actionType ActionMode { get; set; }
        //
        // Summary:
        //     Gets or sets the hatch patterns collection.
        //
        // Exceptions:
        //   T:devDept.EyeshotException:
        //     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Workspace.Document.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public HatchPatternKeyedCollection HatchPatterns { get; set; }
        //
        // Summary:
        //     Gets a space-separated list of supported extensions to OpenGL.
        //
        // Remarks:
        //     OpenGL only.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
```

```
        public string OpenglExtensions { get; }

        //

        // Summary:

        //     Gets or sets the layers collection.

        //

        // Exceptions:

        //   T:devDept.EyeshotException:

        //     Thrown when trying to set a new collection already linked to another
devDept.Eyeshot.Control.Workspace.Document.

        //

        // Remarks:

        //     If the collection is empty, the "Default" layer is added.

        [Browsable(false)]

        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]

        public LayerKeyedCollection Layers { get; set; }

        //

        // Summary:

        //     Gets the stack of the parents of the current BlockReference.

        //

        // Remarks:

        //     The BlockReference on top of the stack is the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference.

        [Browsable(false)]

        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]

        public Stack<BlockReference> Parents { get; }

        //

        // Summary:

        //     Gets the maximum texture size allowed on current OpenGL implementation (in pixel).

        [Category("Workspace - Graphics")]

        [Description("The maximum texture size allowed on current OpenGL implementation (in
pixel).")]

        public int MaxTextureSize { get; }
```

```
        //
        // Summary:
        //     Gets or sets the number of desired samples for Full Screen Anti-Aliasing. The
        //     Full Screen Anti-alasing can be set with the
devDept.Eyeshot.Control.Workspace.AskForAntiAliasing
        //     property.
        [Category("Workspace - Initialization")]
        [Description("Gets or sets the number of desired samples for Full Screen Anti-Aliasing. The Full
Screen Anti-alasing can be set with the AskForAntiAliasing property.")]
        public antialiasingSamplesNumberType AntiAliasingSamples { get; set; }
        //
        // Summary:
        //     Gets graphics API version.
        [Category("Workspace - Graphics")]
        [Description("Renderer version.")]
        public Version RendererVersion { get; }
        //
        // Summary:
        //     Gets background worker thread status.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool IsBusy { get; }
        //
        // Summary:
        //     Gets or sets the normals visibility status.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool ShowNormals { get; set; }
        //
        // Summary:
        //     Print document name. It also appears during print preview preparation.
        [Category("Workspace - Printing")]
```

```
        [Description("Print document name. It also appears during print preview preparation.")]
        public string PrintDocumentName { get; set; }
        //
        // Summary:
        //     If true, animates the camera in the commands that change its position or orientation.
        [Category("Workspace")]
        [Description("If true, animates the camera in the commands that change its position or
orientation.")]
        public bool AnimateCamera { get; set; }
        //
        // Summary:
        //     Gets or sets the duration in milliseconds of the camera animations.
        [Category("Workspace")]
        [Description("Gets or sets the duration of the camera animation.")]
        public int AnimateCameraDuration { get; set; }
        //
        // Summary:
        //     If true, wireframe entities are compiled, otherwise they are drawn on the fly
        //     using a buffered approach.
        //
        // Remarks:
        //     Set false to reduce memory usage with devDept.Eyeshot.Control.rendererType.Direct3D
        //     renderer and a huge numbers of wireframe entities.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool CompileWires { get; set; }
        //
        // Summary:
        //     Suspends the set of the color by the
devDept.Eyeshot.Control.Workspace.SetColorDrawForSelectionAndUpdateIdItemsMap``1(devDept.E
yeshot.GfxDrawForSelectionParams,devDept.Eyeshot.ISelectableItem,System.Int32,System.Int32).
        //
```

```
        // Remarks:
        //     Used to perform custom selection techniques, like the selection of the triangles
        //     inside a Mesh.
        [Browsable(false)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
        public bool SuspendSetColorForSelection { get; set; }
        //
        // Summary:
        //     Gets or sets the progress bar settings.
        [Category("Workspace - User Interface")]
        [Description("Progress bar settings.")]
        public ProgressBar ProgressBar { get; set; }
        //
        // Summary:
        //     Gets or sets the modality used by IsInFrustum() method.
        [Category("Workspace - Performance")]
        [Description("Gets or sets the modality used by IsInFrustum() method.")]
        public Camera.perspectiveFitType IsInFrustumMode { get; set; }
        //
        // Summary:
        //     Gets or sets the maximum number of hatch pattern lines allowed for each
devDept.Eyeshot.Entities.Hatch.
        //
        // Remarks:
        //     Over this bias the Hatch is drawn with solid pattern.
        [Category("Workspace - Performance")]
        [Description("Gets or sets the maximum number of hatch pattern lines allowed for an Hatch.")]
        public int MaxHatchPatternLines { get; set; }
        //
        // Summary:
        //     Gets or sets the maximum number of pattern repetitions allowed between two vertices
```

```
        //     of a curve.

        //

        // Remarks:

        //     Over this bias the line is drawn continuous without pattern.
devDept.Eyeshot.Entities.Entity.LineTypeName

        [Category("Workspace - Performance")]

        [Description("Gets or sets the maximum number of pattern repetitions allowed between two
vertices of a curve.")]

        public int MaxPatternRepetitions { get; set; }

        //

        // Summary:

        //     Gets the name of the renderer. This name is typically specific to a particular

        //     configuration of a hardware platform. It does not change from release to release.

        [Category("Workspace - Graphics")]

        [Description("Name of the renderer. This name is typically specific to a particular configuration
of a hardware platform. It does not change from release to release.")]

        public string RendererName { get; }

        //

        // Summary:

        //     Gets current frames per second rate.

        [Browsable(false)]

        public float FramesPerSecond { get; }

        //

        // Summary:

        //     Tells if the viewport is performing a zoom-pan-rotate operation.

        protected internal bool Moving { get; }


        //

        // Summary:

        //     Occurs when a multitouch surface is double-clicked.

        [Category("Workspace")]

        [Description("Occurs when a multitouch surface is double-clicked.")]
```

```
public event MouseEventHandler MultiTouchDoubleClick;
//
// Summary:
//     Occurs when a finger touches and holds a multitouch surface.
[Category("Workspace")]
[Description("Occurs when a finger touches and holds a multitouch surface.")]
public event MouseEventHandler MultiTouchClick;
//
// Summary:
//     Occurs when the scene bounding box has changed.
[Category("Workspace")]
[Description("Occurs when the scene bounding box has changed.")]
public event BoundingBoxChangedHandler BoundingBoxChanged;
//
// Summary:
//     Occurs when an error happens during the drawing.
//
// Remarks:
//     When an error occurs during the drawing, to make the control usable again it's
//     necessary to call the devDept.Eyeshot.Control.Workspace.Clear
[Category("Workspace")]
[Description("Occurs when an error happens during the drawing.")]
public event ErrorEventHandler ErrorOccurred;
//
// Summary:
//     Occurs when a multitouch surface is released.
[Category("Workspace")]
[Description("Occurs when a multitouch surface is released.")]
public event MultiTouchEventHandler MultiTouchUp;
//
// Summary:
```

```
//     Occurs during the scene navigation with the keyboard.
//
// Remarks:
//     Each key press of the navigation keys starts a small animation to move the camera.
//     Subscribe to this event do something each time the camera is moved.
public event NavigationTimerHandler NavigationTimerTick;
//
// Summary:
//     Occurs when a camera Zoom/Pan/Rotate camera movement begins.
[Description("Occurs when a Zoom/Pan/Rotate camera movement begins.")]
public event CameraMoveEventHandler CameraMoveBegin;
//
// Summary:
//     Occurs when a Zoom/Pan/Rotate camera movement ends.
[Description("Occurs when a Zoom/Pan/Rotate camera movement ends.")]
public event CameraMoveEventHandler CameraMoveEnd;
//
// Summary:
//     Occurs when camera changes its position.
[Description("Occurs when camera changes its position.")]
public event CameraMoveEventHandler CameraChanged;
//
// Summary:
//     Occurs when the background work has completed.
[Category("Workspace")]
[Description("Occurs when the background work has completed.")]
public event WorkUnit.WorkCompletedEventHandler WorkCompleted;
//
// Summary:
//     Occurs when the background work is cancelled.
[Category("Workspace")]
```

```
[Description("Occurs when the background work is cancelled.")]
public event WorkUnit.WorkCancelledEventHandler WorkCancelled;
//
// Summary:
//     Occurs when the background work has failed.
[Category("Workspace")]
[Description("Occurs when the background work has failed.")]
public event WorkUnit.WorkFailedEventHandler WorkFailed;
//
// Summary:
//     Occurs when the devDept.Eyeshot.Control.Workspace.SetView(devDept.Eyeshot.viewType)
//     is called.
[Category("Workspace")]
[Description("Occurs when the SetView is called.")]
public event ViewChangedEventHandler ViewChanged;
//
// Summary:
//     Occurs when entity selection has changed.
[Category("Workspace")]
[Description("Occurs when entity selection has changed.")]
public event SelectionChangedEventHandler SelectionChanged;
//
// Summary:
//     Occurs when the read/write progress has changed.
[Category("Workspace")]
[Description("Occurs when the read/write progress has changed.")]
public event WorkUnit.ProgressChangedEventHandler ProgressChanged;
//
// Summary:
//     Occurs when a multitouch surface is pressed.
[Category("Workspace")]
```

```
[Description("Occurs when a multitouch surface is pressed.")]
public event MultiTouchEventHandler MultiTouchDown;
//
// Summary:
//     Occurs when a multitouch movement is performed.
[Category("Workspace")]
[Description("Occurs when a multitouch movement is performed.")]
public event MultiTouchEventHandler MultiTouchMove;
//
// Summary:
//     Occurs when a Mouse3D movement operation is performed.
[Category("Workspace")]
[Description("Occurs when a Mouse3D movement operation is performed.")]
public event MoveEventHandler Mouse3DMove;
//
// Summary:
//     Occurs when a Mouse3D button is pressed.
[Category("Workspace")]
[Description("Occurs when a Mouse3D button is released.")]
public event ButtonEventHandler Mouse3DButtonUp;
//
// Summary:
//     Occurs when a Mouse3D button is pressed.
[Category("Workspace")]
[Description("Occurs when a Mouse3D button is pressed.")]
public event ButtonEventHandler Mouse3DButtonDown;

public static IntPtr DeleteObject(IntPtr bmp);
//
// Summary:
//     Evaluates the bounding box of a collection of entities without taking care of
```

```
    //     entity visibility.
    //
    // Parameters:
    //   entList:
    //     The collection of entities
    //
    //   globalMin:
    //     Output the min corner
    //
    //   globalMax:
    //     Output the max corner
    public static void EvaluateBoundingBox(ICollection<Entity> entList, out Point3D globalMin, out
Point3D globalMax);
    //
    // Summary:
    //     Gets the currently loaded assembly.
    //
    // Parameters:
    //   product:
    //     Gets assembly product
    //
    //   title:
    //     Gets assembly title
    //
    //   company:
    //     Gets assembly company
    //
    //   version:
    //     Gets assembly version
    //
    //   edition:
```

```
        //     Gets assembly edition
        //
        // Returns:
        //     The currently loaded assembly.
        public static Assembly GetAssembly(out string product, out string title, out string company, out
Version version, out string edition);
        //
        // Summary:
        //     Gets the Eyeshot installation folder.
        //
        // Returns:
        //     The full path where the Eyeshot assemblies are located.
        public static string GetInstallFolder();
        //
        // Summary:
        //     Gets the Eyeshot samples path.
        //
        // Returns:
        //     The full path where Eyeshot's source code samples are located.
        public static string GetSamplesFolder();
        //
        // Summary:
        //     Analyzes a number of entities and returns a closed loop of points starting from
        //     the entity at startIndex. You can provide an array of 100 entities an get a 3D
        //     loop of a square made up of 4 lines. Entities connection is checked using chordalError
        //     parameter, if the end points of two entities are farther of this value they will
        //     not be connected.
        //
        // Parameters:
        //   curveList:
        //     A list of curves
```

```
        //
        //   startIndex:
        //     The first entity to analyze
        //
        //   chordalError:
        //     The chordal error
        //
        //   reverse:
        //     If true, reverses the loop orientation.
        //
        // Returns:
        //     The resulting loop of points.
        [Obsolete("Use Mesh.CreatePlanar() method instead.")]
        public static Point3D[] MakeLoop(IList<ICurve> curveList, int startIndex, double chordalError,
bool reverse);
        public static IntPtr SelectObject(IntPtr hdc, IntPtr bmp);
        //
        // Summary:
        //     Gets the entity line weight.
        //
        // Parameters:
        //   ent:
        //     The entity
        //
        //   parent:
        //     The parent BlockReference
        //
        //   layers:
        //     The layers list
        //
        // Returns:
```

```
//     The line entity.
//
// Remarks:
//     If the parent is not null and the devDept.Eyeshot.Entities.Entity.LineWeightMethod
//     is devDept.Eyeshot.Entities.colorMethodType.byParent it recursively computes
//     the LineWeight of the parent.
    protected static float GetEntityLineWeight(Entity ent, Entity parent, LayerKeyedCollection
layers);
//
// Summary:
//     Gets the outlined text image.
//
// Parameters:
//   text:
//     The text
//
//   font:
//     The text font
//
//   color:
//     The text color
//
//   outlineColor:
//     The outline color
//
//   rotateFlip:
//     The rotation and flip to apply to the bitmap
//
//   thickness:
//     The outline thickness
//
```

```
// Returns:
//     The text bitmap
protected internal static Bitmap GetTextOutlinedImage(string text, Font font, Color color, Color outlineColor, RotateFlipType rotateFlip, float thickness);
//
// Summary:
//     Adjusts camera near and far planes based on design extents and on origin symbol,
//     grid and shadow visibility status.
public void AdjustNearAndFarPlanes();
public override void BeginInit();
//
// Summary:
//     Stops the asynchronous background work.
public void CancelWork();
//
// Summary:
//     Clears all the viewport master collections: entities, tempEntities, blocks, layers,
//     materials, text styles, line types and labels.
public virtual void Clear();
//
// Summary:
//     Close the devDept.Eyeshot.Control.Workspace.OpenBlock and goes back to the visualization
//     of the previously opened block.
//
// Parameters:
//   updateBoundingBox:
//     If true, updates the scene bounding box. Set it to false for better performances
//     in off-screen methods.
public void CloseOpenBlock(bool updateBoundingBox = true);
//
// Summary:
```

```
    //     Compiles 3D user interface elements like the origin symbol, the bitmap background,
    //     the FEM restraint and load symbols, etc.
    public void CompileUserInterfaceElements();
    //
    // Summary:
    //     Compiles 3D user interface elements like the origin symbol, the bitmap background,
    //     the FEM restraint and load symbols, etc.
    public void CompileUserInterfaceElements(Viewport viewport);
    //
    // Summary:
    //     Copies all items inside control master collections (devDept.Eyeshot.Control.Viewport.Labels,
    //     devDept.Eyeshot.Control.Workspace.Blocks, devDept.Eyeshot.Control.Workspace.Layers,
    //     devDept.Eyeshot.Control.Design.Materials, devDept.Eyeshot.Control.Workspace.TextStyles,
    //     devDept.Eyeshot.Control.Workspace.LineTypes,
devDept.Eyeshot.Control.Workspace.HatchPatterns)
    //     to the specified Workspace control.
    //
    // Parameters:
    //   destination:
    //     The Workspace control that is the destination of the elements copied.
    //
    //   replaceRootBlock:
    //     >When true the root block of the destination Workspace is replaced.
    //
    // Remarks:
    //     If the devDept.Eyeshot.Control.Design.Viewports collection is empty then a new
    //     devDept.Eyeshot.Control.Viewport is added to it. You need to call
devDept.Eyeshot.Control.Workspace.Invalidate
    //     to see the effect of this command in the destination Workspace control.
    public void CopyTo(Workspace destination, bool replaceRootBlock = true);
    //
    // Summary:
```

```
//     Copies a raster image of the current viewport scene on the clipboard.
//
// Parameters:
//   drawScale:
//     The image scale
//
//   lineWeightFactor:
//     The factor applied to the line weight of the wire entities
//
// Remarks:
//     The copied image will be a 24-bit bitmap.
public void CopyToClipboardRaster(float drawScale, double lineWeightFactor);
//
// Summary:
//     Copies a raster image of the current viewport scene on the clipboard.
//
// Parameters:
//   drawScale:
//     The image scale
//
// Remarks:
//     The copied image will be a 24-bit bitmap.
public void CopyToClipboardRaster(float drawScale);
//
// Summary:
//     Copies a raster image of the current viewport scene on the clipboard.
//
// Parameters:
//   bitmapSize:
//     The size in which the viewport will be fitted.
//
```

```
    //   lineWeightFactor:
    //     The factor applied to the line weight of the wire entities
    //
    //   drawBackground:
    //     If true, the drawing of the background is skipped
    //
    //   hdwAcceleration:
    //     If true, use the hardware acceleration (openGL only)
    //
    // Remarks:
    //     The copied image will be a 24-bit bitmap.
    public void CopyToClipboardRaster(Size bitmapSize, double lineWeightFactor, bool
drawBackground, bool hdwAcceleration = true);
    //
    // Summary:
    //     Copies a raster image of the current viewport scene on the clipboard.
    //
    // Parameters:
    //   bitmapSize:
    //     The size in which the viewport will be fitted.
    //
    //   drawBackground:
    //     If true, the drawing of the background is skipped
    //
    //   hdwAcceleration:
    //     If true, use the hardware acceleration (openGL only)
    //
    // Remarks:
    //     The copied image will be a 24-bit bitmap.
    public void CopyToClipboardRaster(Size bitmapSize, bool drawBackground, bool
hdwAcceleration = true);
    //
```

```
// Summary:
//     Copies a raster image of the current viewport scene on the clipboard.
//
// Parameters:
//   bitmapSize:
//     The size in which the viewport will be fitted.
//
// Remarks:
//     The copied image will be a 24-bit bitmap.
public void CopyToClipboardRaster(Size bitmapSize);
//
// Summary:
//     Copies a raster image of the current viewport scene on the clipboard.
public void CopyToClipboardRaster();
//
// Summary:
//     Copies a vectorial image of the current scene of the active viewport on the clipboard.
//
// Parameters:
//   fit:
//     If true fits the entire model in the viewport area to captures the whole scene,
//     else uses the current camera
public void CopyToClipboardVector(bool fit);
//
// Summary:
//     Accomplishes the work and call devDept.WorkUnit.WorkCompleted(System.Object)
//     method.
public void DoWork(WorkUnit workUnit);
public override void EndInit();
//
// Summary:
```

```
        //     Looks for the model vertex whose 2D screen projection is closer to the mouse
        //     cursor.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        //   closest:
        //     The closest vertex as a 3D point
        //
        // Returns:
        //     The entity index if the closest vertex is nearer than maxDistance, -1 otherwise.
        //
        // Remarks:
        //     The closest point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     so, if there is a current BlockReference, to get the 3D world coordinates of
        //     the closest point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
        public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, out Point3D
closest);
        //
        // Summary:
        //     Looks for the model vertex whose 2D screen projection is closer to the mouse
        //     cursor.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
```

```
        //
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        //   closestIndex:
        //     The closest vertex index
        //
        // Returns:
        //     The entity index if the closest vertex is nearer than maxDistance, -1 otherwise.
        //
        // Remarks:
        //     The returned entity index refers to the devDept.Eyeshot.Control.Workspace.Entities
        //     if there is no current BlockReference, else it refers to the devDept.Eyeshot.Block.Entities
        //     of the Block referred by the devDept.Eyeshot.Control.Workspace.CurrentBlockReference.
        //     Also, the closestIndex point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     so, if there is a current BlockReference, to get the 3D world coordinates of
        //     the closestIndex point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
        public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, out int
closestIndex);
        //
        // Summary:
        //     Looks for the model vertex whose 2D screen projection is closer to the mouse
        //     cursor skipping entities different from entType.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
        //   maxDistance:
        //     Limit the search to points at this distance from mouse cursor position
```

```
        //
        //   entType:
        //     Entity type
        //
        //   closestIndex:
        //     The closest vertex index
        //
        // Returns:
        //     The entity index if the closest vertex is nearer than maxDistance and the entity
        //     type is entType, -1 otherwise.
        public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, Type
entType, out int closestIndex);
        //
        // Summary:
        //     Looks for the model vertex whose 2D screen projection is closer to the mouse
        //     cursor skipping entities different from entType.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        //   entType:
        //     The type of entities to process
        //
        //   closestVertex:
        //     The closest vertex, with the full information to retrieve the vertex from a
devDept.Eyeshot.Entities.Brep
        //     or devDept.Eyeshot.Entities.Solid
        //
```

```
        // Returns:
        //     The entity index if the closest vertex is nearer than maxDistance and the entity
        //     type is entType, -1 otherwise.
        //
        // Remarks:
        //     The closestVertex point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     so, if there is a current BlockReference, to get the 3D world coordinates of
        //     the closestVertex point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
        public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, Type
entType, out HitVertex closestVertex);
        //
        // Summary:
        //     Looks for the model vertex whose 2D screen projection is closer to the mouse
        //     cursor.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        //   closestVertex:
        //     The closest vertex
        //
        // Returns:
        //     The entity index if the closest vertex is nearer than maxDistance, -1 otherwise.
        //
        // Remarks:
```

```
    //     The closestVertex point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
    //     so, if there is a current BlockReference, to get the 3D world coordinates of
    //     the closestVertex point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
    public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, out
HitVertex closestVertex);
    //
    // Summary:
    //     Looks for the model vertex whose 2D screen projection is closer to the mouse
    //     cursor skipping entities different from entType.
    //
    // Parameters:
    //   mousePos:
    //     Mouse position (zero on top)
    //
    //   maxDistance:
    //     Limit the search to points at this distance in pixels from mouse cursor position
    //
    //   entType:
    //     The type of entities to process
    //
    //   closest:
    //     The closest vertex as a 3D point
    //
    // Returns:
    //     The entity index if the closest vertex is nearer than maxDistance and the entity
    //     type is entType, -1 otherwise.
    //
    // Remarks:
    //     The returned entity index refers to the devDept.Eyeshot.Control.Workspace.Entities
    //     if there is no current BlockReference, else it refers to the devDept.Eyeshot.Block.Entities
```

```
    //     of the Block referred by the devDept.Eyeshot.Control.Workspace.CurrentBlockReference.
    //     Also, the closest point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
    //     so, if there is a current BlockReference, to get the 3D world coordinates of
    //     the closest point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
    public int FindClosestVertex(System.Drawing.Point mousePos, double maxDistance, Type
entType, out Point3D closest);
    //
    // Summary:
    //     Looks for the vertex of the specified entity whose 2D screen projection is closer
    //     to the mouse cursor.
    //
    // Parameters:
    //   entity:
    //     The entity to process
    //
    //   mousePos:
    //     Mouse position (zero on top)
    //
    //   maxDistance:
    //     Limit the search to points at this distance in pixels from mouse cursor position
    //
    // Returns:
    //     The closest vertex, if any is found.
    //
    // Remarks:
    //     The returned point is in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
    //     so, if there is a current BlockReference, to get the 3D world coordinates of
    //     the point you need to transform it by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
```

```
public HitVertex FindClosestVertex(Entity entity, System.Drawing.Point mousePos, double
maxDistance);
//
// Summary:
//     Looks for the model vertices of the specified entity whose 2D screen projections
//     are closer to the mouse cursor.
//
// Parameters:
//   entity:
//     The entity to process
//
//   mousePos:
//     Mouse position (zero on top)
//
//   maxDistance:
//     Limit the search to points at this distance in pixels from mouse cursor position
//
// Returns:
//     The list of vertices of the specified entity nearer than maxDistance.
//
// Remarks:
//     The returned points are in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
//     so, if there is a current BlockReference, to get the 3D world coordinates of
//     the points you need to transform them by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
public List<HitVertex> FindClosestVertices(Entity entity, System.Drawing.Point mousePos, double
maxDistance);
//
// Summary:
//     Looks for the model vertices whose 2D screen projections are closer to the mouse
//     cursor skipping entities different from entType.
```

```
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        //   entType:
        //     The type of entities to process
        //
        // Returns:
        //     The list of vertices nearer than maxDistance and belonging to entities of type
        //     entType.
        //
        // Remarks:
        //     The returned points are in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     so, if there is a current BlockReference, to get the 3D world coordinates of
        //     the points you need to transform them by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
        public List<HitVertex> FindClosestVertices(System.Drawing.Point mousePos, double
maxDistance, Type entType);
        //
        // Summary:
        //     Looks for the model vertices whose 2D screen projections are closer to the mouse
        //     than the maxDistance.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position (zero on top)
        //
```

```
        //   maxDistance:
        //     Limit the search to points at this distance in pixels from mouse cursor position
        //
        // Returns:
        //     The list of vertices nearer than maxDistance.
        //
        // Remarks:
        //     The returned points are in the coordinate space of the
devDept.Eyeshot.Control.Workspace.CurrentBlockReference
        //     so, if there is a current BlockReference, to get the 3D world coordinates of
        //     the points you need to transform them by the
devDept.Eyeshot.Control.Workspace.CurrentTransformation.
        public List<HitVertex> FindClosestVertices(System.Drawing.Point mousePos, double
maxDistance);
        //
        // Summary:
        //     Returns the list of all the visible and selectable entities geometrically crossing
        //     the selection box, regardless of their actual visibility on screen.
        //
        // Parameters:
        //   selectionBox:
        //     Selection rectangle's in screen coordinates
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
        //     no.
        //
        // Returns:
        //     The list of entity indices.
        //
        // Remarks:
        //     All entities in the selectionBox are considered, even the ones covered by others.
```

```
public int[] GetAllCrossingEntities(Rectangle selectionBox, bool selectableOnly = true);
//
// Summary:
//     Selects all entities completely enclosed in the specified selection rectangle.
//
// Parameters:
//   selectionBox:
//     Selection rectangle's in screen coordinates
//
// Returns:
//     The list of entity indices.
public int[] GetAllEnclosedEntities(Rectangle selectionBox);
//
// Summary:
//     Returns all the visible (on screen) top level and selectable entities (which
//     may include the parents stack for nested entities) under the mouse cursor in
//     the active viewport.
//
// Parameters:
//   mousePos:
//     Mouse position in screen coordinates
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
//     no.
//
// Returns:
//     The list of entity indices.
//
// Remarks:
//     The sensitivity is affected by the devDept.Eyeshot.Control.Workspace.PickBoxSize
```

```
        //     parameter. Only the entities that are visible on screen and in the selectionBox
        //     are considered, the ones covered by other entities are not considered.
        public int[] GetAllEntitiesUnderMouseCursor(System.Drawing.Point mousePos, bool
selectableOnly = true);
        //
        // Summary:
        //     Returns all the visible (on screen) and selectable items (which may include the
        //     parents stack for nested entities) under the mouse cursor in the active viewport.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position in screen coordinates
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
        //     no.
        //
        // Returns:
        //     The list of entity indices.
        //
        // Remarks:
        //     The sensitivity is affected by the devDept.Eyeshot.Control.Workspace.PickBoxSize
        //     parameter.
        public SelectedItem[] GetAllItemsUnderMouseCursor(System.Drawing.Point mousePos, bool
selectableOnly = true);
        //
        // Summary:
        //     Returns all the visible labels under the mouse cursor in the active viewport.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position in screen coordinates
```

```
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Control.Labels.Label.Selectable property,
        //     otherwise no.
        //
        // Returns:
        //     The list of label indices.
        //
        // Remarks:
        //     The sensitivity is affected by the devDept.Eyeshot.Control.Workspace.PickBoxSize
        //     parameter. Only the labels that are visible on screen and in the selectionBox
        //     are considered, the ones covered by other labels are not considered.
        public int[] GetAllLabelsUnderMouseCursor(System.Drawing.Point mousePos, bool
selectableOnly = true);
        //
        // Summary:
        //     Returns the list of all visible entities in the specified selection box in the
        //     active viewport.
        //
        // Parameters:
        //   selectionBox:
        //     Selection rectangle in screen coordinates
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
        //     no.
        //
        // Returns:
        //     The list of entity indices.
        public int[] GetAllVisibleEntities(Rectangle selectionBox, bool selectableOnly = true);
        //
```

```
// Summary:
//     Returns the list of all visible items (which may include the parents stack for
//     nested entities) in the specified selection box in the active viewport.
//
// Parameters:
//   selectionBox:
//     Selection rectangle in screen coordinates
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
//     no.
//
// Returns:
//     The selected items array.
//
// Remarks:
//     • The devDept.Eyeshot.Control.Design.SelectionFilterMode determines the type
//     of the item selected.
//     • The devDept.Eyeshot.Control.Design.AssemblySelectionMode determines whether
//     the top level entities or nested entities are returned.
public SelectedItem[] GetAllVisibleItems(Rectangle selectionBox, bool selectableOnly = true);
//
// Summary:
//     Returns the list of all visible labels in the specified selection box.
//
// Parameters:
//   selectionBox:
//     Selection rectangle in screen coordinates
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Control.Labels.Label.Selectable property,
```

```
//     otherwise no.
//
// Returns:
//     The list of label indices in the selection box.
public int[] GetAllVisibleLabels(Rectangle selectionBox, bool selectableOnly = true);
//
// Summary:
//     Returns the list of all the visible and selectable entities crossing the specified
//     selection box.
//
// Parameters:
//   selectionBox:
//     Selection Rectangle box in screen coordinates
//
//   entList:
//     A custom list of entities
//
//   firstOnly:
//     When true, returns immediately after selecting the first entity
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
//     no.
//
//   accParentTransform:
//     Accumulated parent transformation or null/Nothing
//
// Returns:
//     An array of indices representing the selected entity position.
//
// Remarks:
```

```
        //     All entities in the selectionBox are considered, even the ones covered by others.
        public virtual int[] GetCrossingEntities(Rectangle selectionBox, IList<Entity> entList, bool
firstOnly, bool selectableOnly = true, Transformation accParentTransform = null);
        //
        // Summary:
        //     Gets the default cursor for the viewport control.
        public Cursor GetDefaultCursor();
        //
        // Summary:
        //     Returns the index of the first top level entity under the mouse cursor.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position in screen coordinates
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
        //     no.
        //
        // Returns:
        //     The index of the entity, -1 otherwise.
        //
        // Remarks:
        //     The sensitivity is affected by the devDept.Eyeshot.Control.Workspace.PickBoxSize
        //     parameter.
        public int GetEntityUnderMouseCursor(System.Drawing.Point mousePos, bool selectableOnly =
true);
        //
        // Summary:
        //     Gets the first item under the mouse cursor (which may include the parents stack
        //     for nested entities).
        //
```

```
        // Returns:
        //     The item
        //
        // Remarks:
        //     The devDept.Eyeshot.Control.Design.AssemblySelectionMode determines whether the
        //     top level entities or nested entities are returned.
        public SelectedItem GetItemUnderMouseCursor(System.Drawing.Point mousePos, bool
selectableOnly = true);
        //
        // Summary:
        //     Returns the index of the first label under the mouse cursor.
        //
        // Parameters:
        //   mousePos:
        //     Mouse position in screen coordinates
        //
        //   selectableOnly:
        //     When true, checks the devDept.Eyeshot.Control.Labels.Label.Selectable property,
        //     otherwise no.
        //
        // Returns:
        //     The index of the label, -1 otherwise.
        //
        // Remarks:
        //     The sensitivity is affected by the devDept.Eyeshot.Control.Workspace.PickBoxSize
        //     parameter.
        public int GetLabelUnderMouseCursor(System.Drawing.Point mousePos, bool selectableOnly =
true);
        //
        // Summary:
        //     Gets the color of the specified pixel in this viewport
        //
```

```
// Parameters:
//   x:
//     The x window coordinate of the pixel to retrieve
//
//   y:
//     The y window coordinate of the pixel to retrieve (0 on bottom)
//
// Returns:
//     The color of the pixel
public Color GetPixel(int x, int y);
//
// Summary:
//     Returns a thumbnail bitmap of the whole Workspace control. The maximum size is
//     256x256.
//
// Parameters:
//   thumbnailSize:
//     Desired size of the biggest dimension of the Workspace control.
//
// Returns:
//     The bitmap of the current Workspace control with viewports' borders included.
//
// Remarks:
//     For internal use only.
public Bitmap GetPresetManagerThumbnail(int thumbnailSize);
//
// Summary:
//     Gets the cumulative transformation of a blockreference stack.
//
// Parameters:
//   parents:
```

```
    //     The stack of BlockReferences to evaluate (the one on top of the stack is the
    //     last parent of the hierarchy)
    public Transformation GetStackTransformation(Stack<BlockReference> parents);
    //
    // Summary:
    //     Gives the chance to overriders to get temporary face selection data and optionally
    //     to skip temporary face selection drawing.
    //
    // Parameters:
    //   parents:
    //     The stack of parents
    //
    //   ent:
    //     The entity
    //
    //   faceSelInfo:
    //     The list of devDept.Eyeshot.SelectionInfoSubItems
    //
    //   selStatus:
    //     The selection status
    //
    // Returns:
    //     True to skip standard drawing, false otherwise.
    public virtual bool GetTemporarySelectionFaceData(Stack<BlockReference> parents, Entity ent,
List<SelectionInfoSubItems> faceSelInfo, selectionStatusType selStatus);
    //
    // Summary:
    //     Gets the visual refinement deviation value used by the
devDept.Eyeshot.EntityList.Regen(devDept.Eyeshot.RegenOptions)
    //     method. It's estimated automatically from the root block extents.
    //
    // Returns:
```

```
//     The estimated deviation
//
// Remarks:
//     Can be overridden to return a custom value.
public double GetVisualRefinementDeviation();
//
// Summary:
//     Ensures that at least one devDept.Eyeshot.Control.Viewport is present in the
//     devDept.Eyeshot.Design.Viewports collection.
//
// Remarks:
//     This method must be called only when creating the devDept.Eyeshot.Design object
//     at run time. It must not be called when creating the object at design time.
public override void InitializeViewports();
//
// Summary:
//     Invalidates the entire surface of the control and causes the control to be redrawn.
public void Invalidate();
//
// Summary:
//     Gets a boolean indicating whether the control is running at design time.
//
// Remarks:
//     For internal use only.
[Browsable(false)]
[DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)]
public sealed override bool IsDesignMode();
public override bool IsErrorInPaint();
//
// Summary:
//     Isolate a collection of blocks by showing the others in a transparent state.
```

```
//     Pass null to unset the isolated blocks.
//
// Parameters:
//   blocks:
//     The blocks to isolate
//
// Exceptions:
//   T:devDept.EyeshotException:
//     Throw when the feature is not available
//
//   T:devDept.EyeshotException:
//     Throw when a BlockReference is set as current.
//
// Remarks:
//     This method is useful to focus on a part of the model while still seeing the
//     rest of the scene in a transparent state.
//     Check the devDept.Eyeshot.Control.Workspace.IsolateAvailable property before
//     calling this method.
public void IsolateBlocks(IReadOnlyCollection<Block> blocks);
//
// Summary:
//     Isolate a collection of entities by showing the others in a transparent state.
//     Pass null to unset the isolated entities.
//
// Parameters:
//   instances:
//     The entities instance to isolate
//
// Exceptions:
//   T:devDept.EyeshotException:
//     Throw when the feature is not available
```

```
//
//   T:devDept.EyeshotException:
//     Throw when a BlockReference is set as current.
//
// Remarks:
//     This method is useful to focus on a part of the model while still seeing the
//     rest of the scene in a transparent state.
//     Check the devDept.Eyeshot.Control.Workspace.IsolateAvailable property before
//     calling this method.
public void IsolateInstances(IReadOnlyList<Tuple<Stack<BlockReference>, Entity>> instances);
//
// Summary:
//     Isolate a collection of entities by showing the others in a transparent state.
//     Pass null to unset the isolated entities.
//
// Parameters:
//   instances:
//     The entities instance to isolate
//
// Exceptions:
//   T:devDept.EyeshotException:
//     Throw when the feature is not available
//
//   T:devDept.EyeshotException:
//     Throw when a BlockReference is set as current.
//
// Remarks:
//     This method is useful to focus on a part of the model while still seeing the
//     rest of the scene in a transparent state.
//     Check the devDept.Eyeshot.Control.Workspace.IsolateAvailable property before
//     calling this method.
```

```
public void IsolateSelected(IReadOnlyList<SelectedItemBase> instances);
//
// Summary:
//     Gets the Pick Cursor stream.
public override Stream LoadPickCursor();
//
// Summary:
//     Restores the current scene from disk, including entities, textures, blocks and
//     labels.
//
// Parameters:
//   stream:
//     The file stream
//
// Remarks:
//     This command can be very attractive to save and load models in you program but
//     has a serious pitfall: it is based on standard .NET Serialization. This means
//     that any change on the Eyeshot binaries (dll versions, dll names, type names,
//     etc.) will invalidate all your existing files. We strongly recommend to use the
//     Eyeshot proprietary file format or to implement your own file format (simply
//     writing entities properties and recreating entities during file loading) if you
//     need continuity between different Eyeshot DLL versions. Another option can be
//     to use one of the standard format supported (DWG/DXF/IGES) where applicable.
//     The closing of the stream is delegated to you.
[Obsolete("This method is deprecated, use OpenFile instead.")]
public virtual void LoadScene(Stream stream);
//
// Summary:
//     Restores the current scene from disk, including entities, textures, blocks and
//     labels.
//
```

```
        // Parameters:
        //   fileName:
        //     File name
        //
        // Remarks:
        //     This command can be very attractive to save and load models in you program but
        //     has a serious pitfall: it is based on standard .NET Serialization. This means
        //     that any change on the Eyeshot binaries (dll versions, dll names, type names,
        //     etc.) will invalidate all your existing files. We strongly recommend to use the
        //     Eyeshot proprietary file format or to implement your own file format (simply
        //     writing entities properties and recreating entities during file loading) if you
        //     need continuity between different Eyeshot DLL versions. Another option can be
        //     to use one of the standard format supported (DWG/DXF/IGES) where applicable.
        [Obsolete("This method is deprecated, use OpenFile instead.")]
        public virtual void LoadScene(string fileName);
        //
        // Summary:
        //     Restores the next view previously saved on the
devDept.Eyeshot.Control.Viewport.SavedViews
        //     stack.
        //
        // Remarks:
        //     devDept.Eyeshot.Control.Viewport.SavedViews
        public void NextView();
        //
        // Summary:
        //     Open devDept.Eyeshot.Control.Workspace.CurrentBlock. The
devDept.Eyeshot.Control.OriginSymbol
        //     and devDept.Eyeshot.Control.Grid are in the Block reference system.
        //
        // Parameters:
        //   updateBoundingBox:
```

```
//     If true, updates the scene bounding box. Set it to false for better performances
//     in off-screen methods.
//
// Remarks:
//     When a Block is "open" only its entities are shown, the rest of the scene is
//     hidden.
//     Entities are shown in the Block reference system, without the parents transformation.
public void OpenCurrentBlock(bool updateBoundingBox = true);
//
// Summary:
//     Defines the Page Setup.
//
// Parameters:
//   allowMargins:
//     When true, the margins section of the dialog box is enabled.
//
//   showDialog:
//     When true, the PageSetup dialog is shown.
//
//   defaultMargins:
//     The default margins
//
//   paperSize:
//     The paper size
//
//   landscape:
//     The orientation mode
//
// Returns:
//     True if the user has confirmed the settings.
```

```
public bool PageSetup(bool allowMargins = false, bool showDialog = true, int? defaultMargins = null, PaperSize paperSize = null, bool? landscape = null);
//
// Summary:
//     Pans the view from one screen point to the other.
//
// Parameters:
//   from:
//     StartPoint point
//
//   to:
//     End point
//
//   animate:
//     If true performs an animation when changing the view
public virtual void PanCamera(System.Drawing.Point from, System.Drawing.Point to, bool animate);
//
// Summary:
//     Pans the view from one screen point to the other.
//
// Parameters:
//   from:
//     StartPoint point
//
//   to:
//     End point
public virtual void PanCamera(System.Drawing.Point from, System.Drawing.Point to);
//
// Summary:
//     Pans the view downwards.
//
```

```
// Parameters:
//   amount:
//     Amount of pan (in pixels)
public void PanDown(int amount);
//
// Summary:
//     Pans the view to the left.
//
// Parameters:
//   amount:
//     Amount of pan (in pixels)
public void PanLeft(int amount);
//
// Summary:
//     Pans the view to the right.
//
// Parameters:
//   amount:
//     Amount of pan (in pixels)
public void PanRight(int amount);
//
// Summary:
//     Pans the view upwards.
//
// Parameters:
//   amount:
//     Amount of pan (in pixels)
public void PanUp(int amount);
public override bool PreFilterMessage(int msg, IntPtr lParam);
//
// Summary:
```

```
    //     Restores the previous view on the devDept.Eyeshot.Control.Viewport.SavedViews
    //     stack.
    //
    // Remarks:
    //     devDept.Eyeshot.Control.Viewport.SavedViews
    public void PreviousView();
    //
    // Summary:
    //     Selects each entity crossing the specified selection rectangle.
    //     This method is deprecated. Use the overload with the
devDept.Eyeshot.Control.SelectionChangedEventArgs
    //     parameter.
    //
    // Parameters:
    //   selectionBox:
    //     Selection rectangle in screen coordinates
    //
    //   added:
    //     The list of entity indices of entities added to selection
    //
    //   removed:
    //     The list of entity indices of entities removed from selection
    //
    //   firstOnly:
    //     When true, returns immediately after selecting the first entity
    //
    //   invert:
    //     When true, inverts current selection status
    //
    //   selectableOnly:
    //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
```

```
//     no.
//
// Remarks:
//     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
    [Obsolete("Use the overload with the SelectionChangedEventArgs parameter.")]
    public virtual void ProcessSelection(Rectangle selectionBox, bool firstOnly, bool invert, out int[]
added, List<int> removed, bool selectableOnly = true);
//
// Summary:
//     Selects each entity crossing the specified selection rectangle.
//
// Parameters:
//   selectionBox:
//     Selection rectangle in screen coordinates
//
//   eventArgs:
//     The selection changed data
//
//   firstOnly:
//     When true, returns immediately after selecting the first entity
//
//   invert:
//     When true, inverts current selection status
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
//     no.
//
// Remarks:
//     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
    public virtual void ProcessSelection(Rectangle selectionBox, bool firstOnly, bool invert,
SelectionChangedEventArgs eventArgs, bool selectableOnly = true);
```

```
//
// Summary:
//     Selects each entity crossing the specified selection polygon.
//
// Parameters:
//   selectionPolygon:
//     Selection polygon in camera screen coordinates
//
//   invert:
//     If true, inverts current selection status
//
//   eventArgs:
//     The selection changed data
//
// Remarks:
//     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
public virtual void ProcessSelectionByPolygon(List<Point2D> selectionPolygon, bool invert,
SelectionChangedEventArgs eventArgs);
//
// Summary:
//     Selects each entity completely enclosed in the specified selection rectangle.
//
// Parameters:
//   selectionPolygon:
//     Selection polygon in screen coordinates
//
//   eventArgs:
//     The selection changed data
//
//   invert:
//     If true, inverts current selection status
```

```
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        public virtual void ProcessSelectionByPolygonEnclosed(List<Point2D> selectionPolygon, bool
invert, SelectionChangedEventArgs eventArgs);
        //
        // Summary:
        //     Selects each visible entity in the specified selection polygon.
        //
        // Parameters:
        //   selectionPolygon:
        //     Selection polygon in screen coordinates
        //
        //   eventArgs:
        //     The selection changed data
        //
        //   invert:
        //     If true, inverts current selection status
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        public virtual void ProcessSelectionByPolygonVisibleOnly(List<Point2D> selectionPolygon, bool
invert, SelectionChangedEventArgs eventArgs);
        //
        // Summary:
        //     Selects each entity completely enclosed in the specified selection rectangle.
        //     This method is deprecated. Use the overload with the
devDept.Eyeshot.Control.SelectionChangedEventArgs
        //     parameter.
        //
        // Parameters:
        //   selectionBox:
```

```
        //     Selection rectangle's in screen coordinates
        //
        //   added:
        //     The list of entity indices of entities added to selection
        //
        //   removed:
        //     The list of entity indices of entities removed from selection
        //
        //   firstOnly:
        //     If true, returns immediately after selecting the first entity
        //
        //   invert:
        //     If true, inverts current selection status
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        [Obsolete("Use the overload with the SelectionChangedEventArgs parameter.")]
        public virtual void ProcessSelectionEnclosed(Rectangle selectionBox, bool firstOnly, bool invert,
out int[] added, List<int> removed);
        //
        // Summary:
        //     Selects each entity completely enclosed in the specified selection rectangle.
        //
        // Parameters:
        //   selectionBox:
        //     Selection rectangle's in screen coordinates
        //
        //   eventArgs:
        //     The selection changed data
        //
        //   firstOnly:
```

```
    //     If true, returns immediately after selecting the first entity
    //
    //   invert:
    //     If true, inverts current selection status
    //
    // Remarks:
    //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
    public virtual void ProcessSelectionEnclosed(Rectangle selectionBox, bool firstOnly, bool invert,
SelectionChangedEventArgs eventArgs);
    //
    // Summary:
    //     Selects only visible entities in the specified rectangle.
    //
    // Parameters:
    //   selectionBox:
    //     Selection rectangle in screen coordinates
    //
    //   firstOnly:
    //     When true, returns immediately after selecting the first entity
    //
    //   invert:
    //     If true, inverts current selection status
    //
    //   eventArgs:
    //     The selection changed data
    //
    //   selectableOnly:
    //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
    //     no.
    //
    //   temporarySelection:
```

```
    //     Tells if the selection is part of a dynamic selection session (see
devDept.Eyeshot.actionType.SelectVisibleByPickDynamic)
    //
    // Remarks:
    //     Call Control.Invalidate() to see the effect of this function.
    public virtual void ProcessSelectionVisibleOnly(Rectangle selectionBox, bool firstOnly, bool
invert, SelectionChangedEventArgs eventArgs, bool selectableOnly = true, bool temporarySelection =
false);
    //
    // Summary:
    //     Selects only visible labels in the specified rectangle.
    //
    // Parameters:
    //   selectionBox:
    //     Selection rectangle in screen coordinates
    //
    //   firstOnly:
    //     When true, returns immediately after selecting the first entity
    //
    //   invert:
    //     When true, inverts current selection status
    //
    //   eventArgs:
    //     The data where the added and removed labels are written
    //
    //   selectableOnly:
    //     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
    //     no.
    //
    // Remarks:
    //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
```

```
public virtual void ProcessSelectionVisibleOnlyLabels(Rectangle selectionBox, bool firstOnly, bool invert, SelectionChangedEventArgs eventArgs, bool selectableOnly = true);
//
// Summary:
//     Performs semi-transparent entities pre-processing.
//
// Remarks:
//     It is meaningful only when devDept.Eyeshot.Control.Workspace.AccurateTransparency
//     is true.
public void ProcessSemiTransparent();
//
// Summary:
//     Removes unused items from the Workspace master collections (Layers, Blocks, Materials,
//     etc.).
//
// Remarks:
//     The original order of the items is kept inside each collection and the removed
//     items are disposed. Any BlockReference set as current will be unset.
public void Purge();
//
// Summary:
//     Creates a block with the selected devDept.Eyeshot.Control.Workspace.Entities,
//     computes the center of their bounding box, translates them all of the negative
//     of that quantity and returns a BlockReference to that block.
//
// Parameters:
//   blockName:
//     The name of the block that will contain the selected entities.
//
// Returns:
//     The blockReference referring to the created block.
```

```
//
// Remarks:
//     When the entities are very far from the origin their appearance on screen may
//     become very compromised, and the camera may shake badly when moving/rotating
//     the view. To fix this issue you can select these entities and call this method.
public BlockReference RemoveJittering(string blockName = null);
//
// Summary:
//     Gets the blockreference entities, computes the center of their bounding box,
//     translates them all of the negative of that quantity and translates the blockreference
//     back to the original position.
//
// Parameters:
//   blockReference:
//     The blockreference name of the block that will contain the selected entities.
//
// Remarks:
//     When the entities are very far from the origin their appearance on screen may
//     become very compromised, and the camera may shake badly when moving/rotating
//     the view. To fix this issue you can select these entities and call this method.
public void RemoveJittering(BlockReference blockReference);
//
// Summary:
//     Returns an image of the current devDept.Eyeshot.Control.Workspace scene of the
//     active viewport. The image can be scaled even to a poster size one.
//
// Parameters:
//   rectangle:
//     The rectangle of the viewport to capture in screen coordinates (zero on top)
//
//   drawScaleFactor:
```

```
        //     The bitmap scale factor
        //
        //   lineWeightFactor:
        //     The factor applied to the line weight of the wire entities
        //
        //   drawBackground:
        //     If false, the drawing of the background is skipped
        //
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
        //     The 24-bit image.
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
        public Bitmap RenderToBitmap(Rectangle rectangle, double drawScaleFactor, double
lineWeightFactor, bool drawBackground, bool hdwAcceleration = true);
        //
        // Summary:
        //     Returns an image of the current devDept.Eyeshot.Control.Workspace scene of the
        //     active viewport. The image can be scaled even to a poster size one.
        //
        // Parameters:
        //   drawScaleFactor:
        //     The bitmap scale
        //
        //   drawBackground:
        //     If false, the drawing of the background is skipped
        //
```

```
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
        //     The 24-bit image.
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
        public Bitmap RenderToBitmap(double drawScaleFactor, bool drawBackground, bool
hdwAcceleration = true);
        //
        // Summary:
        //     Returns an image of the current devDept.Eyeshot.Control.Workspace scene of the
        //     active viewport. The image can be scaled even to a poster size one.
        //
        // Parameters:
        //   drawScaleFactor:
        //     The bitmap scale
        //
        //   lineWeightFactor:
        //     The factor applied to the line weight of the wire entities
        //
        //   drawBackground:
        //     If false, the drawing of the background is skipped
        //
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
        //     The 24-bit image.
```

```
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
        public Bitmap RenderToBitmap(double drawScaleFactor, double lineWeightFactor, bool
drawBackground, bool hdwAcceleration = true);
        //
        // Summary:
        //     Returns an image of the current devDept.Eyeshot.Control.Workspace scene of the
        //     active viewport. The image can be scaled even to a poster size one.
        //
        // Parameters:
        //   rectangle:
        //     The rectangle of the viewport to capture in screen coordinates (zero on top)
        //
        //   drawScaleFactor:
        //     The bitmap scale factor
        //
        //   drawBackground:
        //     If false, the drawing of the background is skipped
        //
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
        //     The 24-bit image.
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
```

```
public Bitmap RenderToBitmap(Rectangle rectangle, double drawScaleFactor, bool
drawBackground, bool hdwAcceleration = true);
//
// Summary:
//     Returns an image of the current devDept.Eyeshot.Control.Workspace scene. The
//     image can be scaled even to a poster size one.
//
// Parameters:
//   rectangle:
//     The rectangle of the viewport to capture in screen coordinates (zero on top)
//
//   bitmapSize:
//     The size in which the viewport will be fitted.
//
//   drawBackground:
//     If true, the drawing of the background is skipped
//
//   hdwAcceleration:
//     If true, use the hardware acceleration (OpenGL only)
//
// Returns:
//     The 24-bit image.
//
// Remarks:
//     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
//     to devDept.Graphics.backgroundStyleType.None.
public Bitmap RenderToBitmap(RectangleF rectangle, Size bitmapSize, bool drawBackground,
bool hdwAcceleration);
//
// Summary:
//     Returns an image of the current viewport scene. The image can be scaled even
```

```
//     to a poster size one.
//
// Parameters:
//   drawScaleFactor:
//     The bitmap scale
//
// Returns:
//     The 24-bit image.
//
// Remarks:
//     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
//     to devDept.Graphics.backgroundStyleType.None.
public Bitmap RenderToBitmap(double drawScaleFactor);
//
// Summary:
//     Returns an image of the current devDept.Eyeshot.Control.Workspace scene. The
//     image can be scaled even to a poster size one.
//
// Parameters:
//   rectangle:
//     The rectangle of the viewport to capture in screen coordinates (zero on top)
//
//   bitmapSize:
//     The size in which the viewport will be fitted.
//
//   lineWeightFactor:
//     The factor applied to the line weight of the wire entities
//
//   drawBackground:
//     If true, the drawing of the background is skipped
```

```
        //
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
        //     The 24-bit image.
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
        public Bitmap RenderToBitmap(RectangleF rectangle, Size bitmapSize, double lineWeightFactor,
bool drawBackground, bool hdwAcceleration = true);
        //
        // Summary:
        //     Returns an image of the current devDept.Eyeshot.Control.Workspace scene. The
        //     image can be scaled even to a poster size one.
        //
        // Parameters:
        //   bitmapSize:
        //     The size in which the viewport will be fitted.
        //
        //   lineWeightFactor:
        //     The factor applied to the line weight of the wire entities
        //
        //   drawBackground:
        //     If true, the drawing of the background is skipped
        //
        //   hdwAcceleration:
        //     If true, use the hardware acceleration (OpenGL only)
        //
        // Returns:
```

```
//     The 24-bit image.
//
// Remarks:
//     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
//     to devDept.Graphics.backgroundStyleType.None.
public Bitmap RenderToBitmap(Size bitmapSize, double lineWeightFactor, bool drawBackground,
bool hdwAcceleration = true);
//
// Summary:
//     Returns an image of the current devDept.Eyeshot.Control.Workspace scene. The
//     image can be scaled even to a poster size one.
//
// Parameters:
//   bitmapSize:
//     The size in which the viewport will be fitted.
//
//   drawBackground:
//     If true, the drawing of the background is skipped
//
//   hdwAcceleration:
//     If true, use the hardware acceleration (OpenGL only)
//
// Returns:
//     The 24-bit image.
//
// Remarks:
//     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
//     to devDept.Graphics.backgroundStyleType.None.
public Bitmap RenderToBitmap(Size bitmapSize, bool drawBackground, bool hdwAcceleration =
true);
```

```
        //
        // Summary:
        //     Returns an image of the current devDept.Eyeshot.Control.Workspace scene. The
        //     image can be scaled even to a poster size one.
        //
        // Parameters:
        //   bitmapSize:
        //     The size in which the viewport will be fitted.
        //
        // Returns:
        //     The 24-bit image.
        //
        // Remarks:
        //     To get a 32-bit image with transparent background set the
devDept.Eyeshot.Control.BackgroundSettings.StyleMode
        //     to devDept.Graphics.backgroundStyleType.None.
        public Bitmap RenderToBitmap(Size bitmapSize);
        //
        // Summary:
        //     Close all opened blocks up to the root one.
        //
        // Parameters:
        //   updateBoundingBox:
        //     If true, updates the scene bounding box. Set it to false for better performances
        //     in off-screen methods.
        public void ResetOpenBlocks(bool updateBoundingBox = true);
        //
        // Remarks:
        //     You need to subscribe to the ResizeBegin event of the parent control and call
        //     this method there.
        public void ResizeBegin();
```

```
        //
        // Summary:
        //     Restores the previously saved view in the active viewport.
        //
        // Parameters:
        //   saved:
        //     The devDept.Eyeshot.Camera object previously initialized by
devDept.Eyeshot.Control.Workspace.SaveView(devDept.Eyeshot.Camera@)
        //
        // Remarks:
        //     If devDept.Eyeshot.Control.Workspace.AnimateCamera is true, the operation animates
        //     the Camera using the time interval specified by
devDept.Eyeshot.Control.Workspace.AnimateCameraDuration.
        public void RestoreView(Camera saved);
        //
        // Summary:
        //     Sets the view direction as the normal of the plane under the mouse cursor.
        //
        // Parameters:
        //   mouseLocation:
        //     Location of the mouse
        public void RotateCamera(System.Drawing.Point mouseLocation);
        //
        // Summary:
        //     Saves the current scene on disk, including entities, textures, blocks and layouts.
        //
        // Parameters:
        //   stream:
        //     The file stream
        //
        // Remarks:
        //     This command can be very attractive to save and load models in you program but
```

```
//     has a serious pitfall: it is based on standard .NET Serialization. This means
//     that any change on the Eyeshot binaries (dll versions, dll names, type names,
//     etc.) will invalidate all your existing files. We strongly recommend to use the
//     Eyeshot proprietary file format or to implement your own file format (simply
//     writing entities properties and recreating entities during file loading) if you
//     need continuity between different Eyeshot DLL versions. Another option can be
//     to use one of the standard format supported (DWG/DXF/IGES) where applicable.
//     The closing of the stream is delegated to you.
[Obsolete("This method is deprecated, use SaveFile instead.")]
public virtual void SaveScene(Stream stream);
//
// Summary:
//     Saves the current scene on disk, including entities, textures, blocks and layouts.
//
// Parameters:
//   fileName:
//     File name
//
// Remarks:
//     This command can be very attractive to save and load models in you program but
//     has a serious pitfall: it is based on standard .NET Serialization. This means
//     that any change on the Eyeshot binaries (dll versions, dll names, type names,
//     etc.) will invalidate all your existing files. We strongly recommend to use the
//     Eyeshot proprietary file format or to implement your own file format (simply
//     writing entities properties and recreating entities during file loading) if you
//     need continuity between different Eyeshot DLL versions. Another option can be
//     to use one of the standard format supported (DWG/DXF/IGES) where applicable.
[Obsolete("This method is deprecated, use SaveFile instead.")]
public virtual void SaveScene(string fileName);
//
// Summary:
```

```
    //     Saves the current view on the devDept.Eyeshot.Control.Viewport.SavedViews stack.
    //
    // Remarks:
    //     devDept.Eyeshot.Control.Viewport.SavedViews
    public void SaveView();
    //
    // Summary:
    //     Stores the current view in the active viewport.
    //
    // Parameters:
    //   saved:
    //     Will hold a copy of the active devDept.Eyeshot.Camera object
    public void SaveView(out Camera saved);
    //
    // Summary:
    //     Scales the element for high DPI settings.
    //
    // Remarks:
    //     This method needs to be called if the element is added outside of the
InitializeComponents()
    //     to adjust the rendering for high dpi settings.
    public void ScaleForDPI();
    //
    // Summary:
    //     Maps screen coordinates to world coordinates in the active viewport.
    //
    // Parameters:
    //   mousePointList:
    //     2D mouse point list (zero on top)
    //
    //   pe:
```

```
//     The plane equation
//
// Returns:
//     The associated 3D world point list.
public Point3D[] ScreenToPlane(IList<System.Drawing.Point> mousePointList, PlaneEquation pe);
//
// Summary:
//     Maps screen coordinates to world coordinates in the active viewport.
//
// Parameters:
//   mousePos:
//     Mouse cursor position (zero on top)
//
//   plane:
//     The plane
//
//   intPoint:
//     The intersection point. null/Nothing if the plane perpendicular to the screen.
//
// Returns:
//     True if the mapping succeeded, false otherwise.
public bool ScreenToPlane(System.Drawing.Point mousePos, Plane plane, out Point3D intPoint);
//
// Summary:
//     Maps screen coordinates to world coordinates in the active viewport.
//
// Parameters:
//   mousePos:
//     Mouse cursor position (zero on top)
//
//   pe:
```

```
        //     The plane equation
        //
        //   intPoint:
        //     The intersection point. null/Nothing if the plane perpendicular to the screen.
        //
        // Returns:
        //     True if the mapping succeeded, false otherwise.
        public bool ScreenToPlane(System.Drawing.Point mousePos, PlaneEquation pe, out Point3D
intPoint);
        //
        // Summary:
        //     Maps screen coordinates to world coordinates in the active viewport.
        //
        // Parameters:
        //   mousePointList:
        //     2D mouse point list (zero on top)
        //
        //   plane:
        //     The plane
        //
        // Returns:
        //     The associated 3D world point list.
        public Point3D[] ScreenToPlane(IList<System.Drawing.Point> mousePointList, Plane plane);
        //
        // Summary:
        //     Maps screen coordinates to world coordinates.
        //
        // Parameters:
        //   mousePos:
        //     Mouse cursor position (zero on top)
        //
```

```
// Returns:
//     The associated 3D world point if there is geometry in the point position, null
//     otherwise.
//
// Remarks:
//     The depth is read from the depth buffer, so its precision affects accuracy.
public Point3D ScreenToWorld(System.Drawing.Point mousePos);
//
// Summary:
//     Maps screen coordinates to world coordinates.
//
// Parameters:
//   mousePointList:
//     2D mouse point list (zero on top)
//
// Returns:
//     The associated 3D world point list.
//
// Remarks:
//     Positions that don't have geometry underneath will return null.
public Point3D[] ScreenToWorld(IList<System.Drawing.Point> mousePointList);
//
// Summary:
//     Sets a coded color depending on the entity id.
//
// Parameters:
//   currentEntityId:
//     The id of the entity
//
// Remarks:
//     Used by visible selection methods.
```

```
public void SetColorDrawForSelection(int currentEntityId);
//
// Summary:
//     Sets a coded color depending on the entity id.
//
// Parameters:
//   data:
//     The selection parameters
//
//   item:
//     The item corresponding to the
devDept.Eyeshot.GfxDrawForSelectionParams.FalseColorIndex.
//     If null there is no conversion to perform on the falseColorIndex.
//
//   partIndex:
//     Index of the part (edge, face, vertex). If -1, the item is not a subPart
//
//   shellIndex:
//     Index of the shell (Brep only)
//
// Remarks:
//     Used by visible selection methods.
public void SetColorDrawForSelectionAndUpdateIdItemsMap<T>(GfxDrawForSelectionParams
data, ISelectableItem item, int partIndex = -1, int shellIndex = -1) where T : SelectedItem, new();
//
// Summary:
//     Sets a BlockReference as current.
//
// Parameters:
//   blockReference:
//     The BlockReference to set as current
//
```

```
        //   updateBoundingBox:
        //     If true, updates the scene bounding box. Set it to false for better performances
        //     in off-screen methods that call this to just use the
devDept.Eyeshot.Entities.BlockReference.AccumulatedParentsTransform
        //
        // Remarks:
        //     If it is null, the current BlockReference is unset.
        //     The BlockReference must be a first-level child of the one that is currently set
        //     (i.e. it must be in the entities of the devDept.Eyeshot.Control.Workspace.CurrentBlock).
        //     When a BlockReference is set as current, only the referenced block entities are
        //     active on the scene, others are still visible but in a "frozen" state.
        public void SetCurrent(BlockReference blockReference, bool updateBoundingBox = true);
        //
        // Summary:
        //     Sets a sequence of BlockReferences as current.
        //
        // Parameters:
        //   parents:
        //     The stack of BlockReferences to set as current (the bottom-most is the first
        //     parent, which must be in the entities of the
devDept.Eyeshot.Control.Workspace.CurrentBlock,
        //     the top-most is the last parent of the hierarchy). If null or empty, the current
        //     BlockReference is unset.
        //
        //   updateBoundingBox:
        //     If true, updates the scene bounding box. Set it to false for better performances
        //     in off-screen methods that call this to just use the
devDept.Eyeshot.Entities.BlockReference.AccumulatedParentsTransform
        public void SetCurrentStack(Stack<BlockReference> parents, bool updateBoundingBox = true);
        //
        // Summary:
        //     Sets the cursor.
```

```
        //
        // Parameters:
        //   cursor:
        //     The cursor
        //
        // Remarks:
        //     If devDept.Eyeshot.Control.WorkspaceBase.Renderer is different from rendererType.Native,
        //     use WorkspaceBase.SetCursor(Stream) (WPF only)
        public void SetCursor(Cursor cursor);
        //
        // Summary:
        //     Sets the default cursor for the viewport control.
        //
        // Parameters:
        //   cursor:
        //     The cursor to set as default
        //
        // Remarks:
        //     If devDept.Eyeshot.Control.WorkspaceBase.Renderer is different from rendererType.Native,
        //     to set cursors different from the ones specified in devDept.Eyeshot.Control.cursorType
        //     use the WorkspaceBase.SetDefaultCursor(Stream) instead (WPF only)
        public void SetDefaultCursor(Cursor cursor);
        //
        // Summary:
        //     Sets the parent of the current BlockReference (if it exists) as current.
        //
        // Parameters:
        //   updateBoundingBox:
        //     If true, updates the scene bounding box. Set it to false for better performances
        //     in off-screen methods that call this to just use the
devDept.Eyeshot.Entities.BlockReference.AccumulatedParentsTransform
```

```
public void SetParentAsCurrent(bool updateBoundingBox = true);
//
// Summary:
//     Sets the handle of the parent control.
//
// Parameters:
//   hwnd:
//     The handle of the parent to set.
public override void SetParentHandle(IntPtr hwnd);
//
// Summary:
//     Set the printer settings.
//
// Parameters:
//   printerSettings:
//     The printer settings.
//
//   showDialog:
//     When true the print dialog is shown before printing, otherwise no.
//
// Remarks:
//     If you also need to use the PageSetup() method, then this method should be called
//     first.
public void SetPrinterSettings(PrinterSettings printerSettings, bool showDialog = true);
//
// Summary:
//     Sets the selected BlockReference as current (if there is one).
//
// Parameters:
//   updateBoundingBox:
//     If true, updates the scene bounding box. Set it to false for better performances
```

```
    //     in off-screen methods that call this to just use the
devDept.Eyeshot.Entities.BlockReference.AccumulatedParentsTransform
    public void SetSelectionAsCurrent(bool updateBoundingBox = true);
    //
    // Summary:
    //     Sets the view direction of the camera to the specified direction.
    //
    // Parameters:
    //   direction:
    //     The new camera direction.
    //
    //   upVector:
    //     The camera up vector
    //
    //   fit:
    //     if true fits the scene in the viewport
    //
    //   animate:
    //     If true performs an animation when changing the view
    //
    //   margin:
    //     Pixels margin from the border, if fit is true
    //
    //   selectedOnly:
    //     If fit is true, fits only the selected entities
    //
    // Remarks:
    //     The orientation of the camera is performed with an animation if
devDept.Eyeshot.Control.Workspace.AnimateCamera
    //     is true The direction points outside the screen, so the Camera is actually looking
    //     in the opposite direction.
```

```
public void SetView(Vector3D direction, Vector3D upVector, bool fit, bool animate, int margin =
10, bool selectedOnly = false);

//

// Summary:

//     Sets the view direction of the camera to the specified direction.

//

// Parameters:

//   direction:

//     The new camera direction.

//

//   upVector:

//     The camera up vector

//

//   fit:

//     if true fits the scene in the viewport

//

//   margin:

//     Pixels margin from the border, if fit is true

//

//   selectedOnly:

//     If fit is true, fits only the selected entities

//

// Remarks:

//     The orientation of the camera is performed with an animation if
devDept.Eyeshot.Control.Workspace.AnimateCamera

//     is true The direction points outside the screen, so the Camera is actually looking

//     in the opposite direction.

public void SetView(Vector3D direction, Vector3D upVector, bool fit, int margin = 10, bool
selectedOnly = false);

//

// Summary:

//     Sets the view direction of the camera to the specified direction.
```

```
        //
        // Parameters:
        //   direction:
        //     The new camera direction.
        //
        //   fit:
        //     if true fits the scene in the viewport
        //
        //   animate:
        //     If true performs an animation when changing the view
        //
        //   margin:
        //     Pixels margin from the border, if fit is true
        //
        //   selectedOnly:
        //     If fit is true, fits only the selected entities
        //
        // Remarks:
        //     The direction points outside the screen, so the Camera is actually looking in
        //     the opposite direction.
        public void SetView(Vector3D direction, bool fit, bool animate, int margin = 10, bool
selectedOnly = false);
        //
        // Summary:
        //     Sets the view direction of the camera to the specified direction.
        //
        // Parameters:
        //   direction:
        //     The new camera direction.
        //
        //   fit:
```

```
        //     if true fits the scene in the viewport
        //
        //   margin:
        //     Pixels margin from the border, if fit is true
        //
        //   selectedOnly:
        //     If fit is true, fits only the selected entities
        //
        // Remarks:
        //     The orientation of the camera is performed with an animation if
devDept.Eyeshot.Control.Workspace.AnimateCamera
        //     is true. The direction points outside the screen, so the Camera is actually looking
        //     in the opposite direction.
        public void SetView(Vector3D direction, bool fit, int margin = 10, bool selectedOnly = false);
        //
        // Summary:
        //     Sets the specified view in the active viewport.
        //
        // Parameters:
        //   rotation:
        //     The new camera rotation
        //
        //   target:
        //     The new cameratarget
        //
        //   distance:
        //     The new cameradistance
        //
        //   zoomFactor:
        //     The new camera zoomFactor
        //
```

```
    //   animate:
    //     If true performs an animation when changing the view
    public void SetView(Quaternion rotation, Point3D target, double distance, double zoomFactor,
bool animate);
    //
    // Summary:
    //     Sets the specified view in the active viewport.
    //
    // Parameters:
    //   view:
    //     View type
    //
    //   fit:
    //     If true fits the view
    //
    //   animate:
    //     If true performs an animation when changing the view
    public void SetView(viewType view, bool fit, bool animate);
    //
    // Summary:
    //     Sets the specified view in the active viewport.
    //
    // Parameters:
    //   view:
    //     View type
    //
    //   fit:
    //     If true fits the view
    //
    //   margin:
    //     Pixels margin from the border, if fit is true
```

```
//
//   animate:
//     If true performs an animation when changing the view
//
//   selectedOnly:
//     If fit is true, fits only the selected entities
public void SetView(viewType view, bool fit, bool animate, int margin, bool selectedOnly = false);
//
// Summary:
//     Sets the specified view in the active viewport by doing an animation.
//
// Parameters:
//   rotation:
//     The new camera rotation
//
//   target:
//     The new cameratarget
//
//   distance:
//     The new cameradistance
//
//   zoomFactor:
//     The new camera zoomFactor
public void SetView(Quaternion rotation, Point3D target, double distance, double zoomFactor);
//
// Summary:
//     Sets the specified view in the active viewport.
//
// Parameters:
//   view:
//     View type
```

```
public void SetView(viewType view);
//
// Summary:
//     Starts the work asynchronously.
//
// Remarks:
//     It can handle just one WorkUnit at the time. Every additional call when the first
//     thread is not completed will raise an Exception.
//     For handling a queue of WorkUnits, use devDept.WorkManager`1 instead.
public void StartWork(WorkUnit workUnit);
//
// Summary:
//     Suspends the updates of Workspace control.
//
// Parameters:
//   suspend:
//     If true suspends the updates, else resumes them
//
// Remarks:
//     For internal use only.
public void SuspendUpdate(bool suspend);
//
// Summary:
//     Computes the scene bounding box, rebuilds simplified representation and updates
//     some important variables. If the bounding box, grid or shadow are visible, it
//     updates them too.
//
// Remarks:
//     Shadow is updated only if it is visible and if the design has the bounding box
//     height bigger than zero.
//     It's important to call this method when the visibility status of some entities
```

```
        //     changes.
        //     Visual refinement deviation is updated as well but you need to set
devDept.Eyeshot.Entities.Entity.RegenMode
        //     as devDept.Eyeshot.Entities.regenType.RegenAndCompile for all entities and call
        //     Workspace.Entities.Regen() method to see objects with new tessellation.
        public void UpdateBoundingBox();
        //
        // Summary:
        //     This method needs to be called before using one of the visible selection action
        //     modes or one of the methods that get the entities (or labels) under the mouse
        //     cursor if the devDept.Eyeshot.Entities.Entity.Visible or the
devDept.Eyeshot.Entities.Entity.Selectable
        //     (or devDept.Eyeshot.Control.Labels.Label.Visible or
devDept.Eyeshot.Control.Labels.Label.Selectable
        //     in case of labels selection) changed and the camera was not moved.
        public void UpdateVisibleSelection();
        //
        // Summary:
        //     Updates the workspace control in both desing-time and run-time mode. If necessary
        //     adjust near and far planes to accomodate origin symbols of various sizes.
        public void UpdateWorkspace();
        public override IntPtr WndProcMouse3D(IntPtr hwnd, int msg, IntPtr wParam, IntPtr lParam, ref
bool handled);
        //
        // Summary:
        //     Maps world coordinates to screen coordinates.
        //
        // Parameters:
        //   x:
        //     3D point's x-coordinate
        //
        //   y:
```

```
//     3D point's y-coordinate
//
//   z:
//     3D point's z-coordinate
//
// Returns:
//     The associated projected screen point (zero on bottom).
//
// Remarks:
//     The z-component of the returned point is in the normalized device coordinate
//     space [0,1]. A value outside the [0,1] range means that the point is outside
//     the near-far clipping planes of the camera.
public Point3D WorldToScreen(double x, double y, double z);
//
// Summary:
//     Maps world coordinates to screen coordinates.
//
// Parameters:
//   point:
//     The 3D point to project on screen
//
// Returns:
//     The associated projected screen point (zero on bottom)
//
// Remarks:
//     The z-component of the returned point is in the normalized device coordinate
//     space [0,1]. A value outside the [0,1] range means that the point is outside
//     the near-far clipping planes of the camera.
public Point3D WorldToScreen(Point3D point);
//
// Summary:
```

```
//     Maps world coordinates to screen coordinates.
//
// Parameters:
//   pointList:
//     3D point list
//
// Returns:
//     The associated projected screen point list (zero on bottom).
//
// Remarks:
//     The z-component of the returned point is in the normalized device coordinate
//     space [0,1]. A value outside the [0,1] range means that the point is outside
//     the near-far clipping planes of the camera.
public Point3D[] WorldToScreen(IList<Point3D> pointList);
//
// Summary:
//     Saves a raster image of the current viewport scene on disk.
//
// Parameters:
//   drawScaleFactor:
//     The image scale
//
//   lineWeightFactor:
//     The factor applied to the line weight of the wire entities
//
//   fileName:
//     A string that contains the name of the file to which to save the raster image
//
//   format:
//     The file format of the raster image
//
```

```
//   drawBackground:
//     If false, the drawing of the background is skipped
//
//   hdwAcceleration:
//     If true, use the hardware acceleration (OpenGL only)
//
// Remarks:
//     The saved file will contain a 24-bit image.
    public void WriteToFileRaster(float drawScaleFactor, double lineWeightFactor, string fileName,
ImageFormat format, bool drawBackground, bool hdwAcceleration = true);
//
// Summary:
//     Saves a raster image of the current viewport scene on disk.
//
// Parameters:
//   drawScaleFactor:
//     The image scale
//
//   fileName:
//     A string that contains the name of the file to which to save the raster image
//
//   format:
//     The file format of the raster image
//
//   drawBackground:
//     If false, the drawing of the background is skipped
//
//   hdwAcceleration:
//     If true, use the hardware acceleration (OpenGL only)
//
// Remarks:
```

```
//     The saved file will contain a 24-bit image.
public void WriteToFileRaster(float drawScaleFactor, string fileName, ImageFormat format, bool drawBackground, bool hdwAcceleration = true);
//
// Summary:
//     Saves a raster image of the current viewport scene on disk.
//
// Parameters:
//   drawScaleFactor:
//     The image scale factor
//
//   fileName:
//     A string that contains the name of the file to which to save the raster image
//
//   format:
//     The file format of the raster image
//
// Remarks:
//     The saved file will contain a 24-bit image.
public void WriteToFileRaster(float drawScaleFactor, string fileName, ImageFormat format);
//
// Summary:
//     Saves a vectorial image of the current scene of the active viewport on disk in
//     EMF format.
//
// Parameters:
//   fit:
//     If true fits the entire model in the viewport area to captures the whole scene,
//     else uses the current camera
//
//   fileName:
```

```
//     The name of the file to save
//
// Remarks:
//     If the file exists, it's overwritten.
public void WriteToFileVector(bool fit, string fileName);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   dy:
//     Zoom amount
public virtual void ZoomCamera(int dy);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   dy:
//     Zoom amount
//
//   animate:
//     If true performs an animation when changing the view
public virtual void ZoomCamera(int dy, bool animate);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   mousePos:
//     The screen point to keep fixed
```

```
//
//   dy:
//     The zoom amount
//
//   animate:
//     If true performs an animation when changing the view
public virtual void ZoomCamera(System.Drawing.Point mousePos, int dy, bool animate);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   dy:
//     Zoom amount
//
//   zoomSpeed:
//     Zoom speed
public virtual void ZoomCamera(int dy, double zoomSpeed);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   dy:
//     Zoom amount
//
//   zoomSpeed:
//     Zoom speed
//
//   animate:
//     If true performs an animation when changing the view
```

```
public virtual void ZoomCamera(int dy, double zoomSpeed, bool animate);
//
// Summary:
//     Zooms the view of the specified amount.
//
// Parameters:
//   mousePos:
//     The screen point to keep fixed
//
//   dy:
//     The zoom amount
public virtual void ZoomCamera(System.Drawing.Point mousePos, int dy);
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
//
// Parameters:
//   entList:
//     The list of entities to fit
//
//   selectedOnly:
//     If true, fits only selected entities.
//
//   margin:
//     Pixels margin from the border
public virtual void ZoomFit(IList<Entity> entList, bool selectedOnly, int margin);
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
//
// Parameters:
```

```
//   selectedOnly:
//     If true, fits only selected entities.
//
//   margin:
//     Pixels margin from the border
public virtual void ZoomFit(bool selectedOnly, int margin);
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
public virtual void ZoomFit();
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
//
// Parameters:
//   margin:
//     Pixels margin from the border
public virtual void ZoomFit(int margin);
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
//
// Parameters:
//   selectedOnly:
//     If true, fits only selected entities.
public virtual void ZoomFit(bool selectedOnly);
//
// Summary:
//     Fits the entire scene in the viewport control's client area.
//
// Parameters:
```

```
//   entList:
//     The list of entities to fit
//
//   selectedOnly:
//     If true, fits only selected entities.
public virtual void ZoomFit(IList<Entity> entList, bool selectedOnly);
//
// Summary:
//     Fits the selected items.
//
// Parameters:
//   items:
//     The collection of entities to fit
//
//   margin:
//     Pixels margin from the border
public virtual void ZoomFit(IList<SelectedItem> items, int margin);
//
// Summary:
//     Fits the selected items.
//
// Parameters:
//   items:
//     The collection of entities to fit
//
//   margin:
//     Pixels margin from the border
//
//   perspectiveFitMode:
//     Zoom fit accuracy in perspective projection mode
```

```
public virtual void ZoomFit(IList<SelectedItem> items, int margin, Camera.perspectiveFitType perspectiveFitMode);
//
// Summary:
//     Fits the selected items.
//
// Parameters:
//   items:
//     The collection of entities to fit
public virtual void ZoomFit(IList<SelectedItem> items);
//
// Summary:
//     Fits the selected entities, including the ones inside BlockReferences.
//
// Parameters:
//   margin:
//     Pixels margin from the border
//
//   perspectiveFitMode:
//     Zoom fit accuracy in perspective projection mode
public virtual void ZoomFitSelectedLeaves(int margin, Camera.perspectiveFitType perspectiveFitMode);
//
// Summary:
//     Fits the selected entities, including the ones inside BlockReferences.
//
// Parameters:
//   margin:
//     Pixels margin from the border
public virtual void ZoomFitSelectedLeaves(int margin);
//
// Summary:
```

```
//     Fits the selected entities, including the ones inside BlockReferences.
public virtual void ZoomFitSelectedLeaves();
//
// Summary:
//     Zooms the view in.
//
// Parameters:
//   amount:
//     Amount of zoom (in pixels)
public void ZoomIn(int amount);
//
// Summary:
//     Zooms the view out.
//
// Parameters:
//   amount:
//     Amount of zoom (in pixels)
public void ZoomOut(int amount);
//
// Summary:
//     Zooms to the specified window.
//
// Parameters:
//   p1:
//     Window's diagonal start point in screen coordinates
//
//   p2:
//     Window's diagonal end point in screen coordinates
public virtual void ZoomWindow(System.Drawing.Point p1, System.Drawing.Point p2);
//
// Summary:
```

```
    //     Adds the context menu items related to the Mouse3D.
    protected virtual void AddMouse3DContextMenuItems();
    protected override AccessibleObject CreateAccessibilityInstance();
    //
    // Summary:
    //     For internal use only.
    protected ShaderParameters CreateShaderParams(DrawSceneParams myParams);
    //
    // Summary:
    //     Clean up any resources being used.
    //
    // Parameters:
    //   disposing:
    //     true if managed resources should be disposed; otherwise, false.
    protected override void Dispose(bool disposing);
    //
    // Summary:
    //     Draws design, shadow and bounding box.
    protected virtual void Draw3D(DrawSceneParams myParams);
    protected virtual void DrawForSelection(DrawEntitiesParams myParams);
    //
    // Summary:
    //     Draws overlaying UI elements.
    //
    // Parameters:
    //   data:
    //     The draw data
    protected virtual void DrawOverlay(DrawSceneParams data);
    protected virtual void DrawOverlayBlended(DrawSceneParams myParams, bool
isCurrentViewport);
    //
```

```
// Summary:
//     Main drawing method.
//
// Parameters:
//   viewport:
//     The viewport to draw
//
//   drawScale:
//     Image resolution scale
//
//   lineWeightFactor:
//     The factor applied to the line weight of the wire entities
//
//   zoomRect:
//     Zoom rectangle
//
//   drawOverlay:
//     If true, the overlaying UI elements are drawn
//
//   swapBuffer:
//     If true, front and back OpenGL buffers will be swapped
//
//   designTime:
protected void DrawScene(Viewport viewport, float drawScale, float lineWeightFactor,
RectangleF zoomRect, bool drawOverlay, bool swapBuffer, bool designTime);
//
// Summary:
//     Draws the design shadow
protected virtual void DrawShadow(float drawScale);
//
// Summary:
```

```
        //     Draws a texture.
        //
        // Parameters:
        //   texture:
        //     The texture to draw
        //
        //   x:
        //     The texture x position
        //
        //   y:
        //     The texture y position
        //
        //   align:
        //     The texture alignment
        //
        //   flipY:
        //     If true, flips the image in the Y direction
        protected void DrawTexture(TextureBase texture, int x, int y, ContentAlignment align, bool flipY =
false);
        protected virtual void DrawVertexIndices(DrawSceneParams mySceneParams);
        //
        // Summary:
        //     Draws the vertices of the entities.
        //
        // Remarks:
        //     When devDept.Eyeshot.Control.Viewport.ShowVertices is true.
        protected virtual void DrawVertices(DrawEntitiesParams myParams);
        //
        // Summary:
        //     Draws the viewport background and reflections.
        //
```

```
// Parameters:
//   data:
//     The drawing parameters
//
// Remarks:
//     Override this to draw some custom graphics before the scene is drawn.
protected virtual void DrawViewportBackground(DrawSceneParams data);
//
// Summary:
//     Ends a sequence of Zoom-Pan-Rotate movements.
protected void EndZoomPanRotate(Viewport viewport);
//
// Summary:
//     Raises the ProgressChanged event.
//
// Parameters:
//   e:
//     A devDept.WorkUnit.ProgressChangedEventArgs that contains the event data.
protected void FireProgressChanged(WorkUnit.ProgressChangedEventArgs e);
//
// Summary:
//     Raises the WorkCancelled event.
//
// Parameters:
//   e:
//     A devDept.WorkUnitEventArgs that contains the event data.
protected void FireWorkCancelled(WorkUnitEventArgs e);
//
// Summary:
//     Raises the WorkCompleted event.
//
```

```
// Parameters:
//   e:
//     A devDept.WorkCompletedEventArgs that contains the event data.
protected void FireWorkCompleted(WorkCompletedEventArgs e);
//
// Summary:
//     Raises the WorkCancelled event.
//
// Parameters:
//   e:
//     A System.EventArgs that contains the event data.
protected void FireWorkFailed(WorkFailedEventArgs e);
protected override void FreeCursors();
//
// Summary:
//     Returns the list of all the visible and selectable entities crossing the specified
//     selection box.
//
// Parameters:
//   selectionBox:
//     Selection Rectangle box in screen coordinates
//
//   firstOnly:
//     When true, returns immediately after selecting the first entity
//
//   selectableOnly:
//     When true, checks the devDept.Eyeshot.Entities.Entity.Selectable property, otherwise
//     no.
//
// Returns:
//     An array of indices representing the selected entity position.
```

```
        //
        // Remarks:
        //     All entities in the selectionBox are considered, even the ones covered by others.
        //     Affected by selection scope (Design.SetSelectionScope()).
        protected virtual int[] GetCrossingEntities(Rectangle selectionBox, bool firstOnly, bool
selectableOnly = true);
        protected virtual Viewport GetDefaultViewport();
        //
        // Summary:
        //     Gets the handle of the parent control.
        //
        // Returns:
        //     The handle of the parent.
        protected IntPtr GetParentHandle();
        //
        // Summary:
        //     Gets the text bitmap.
        //
        // Parameters:
        //   text:
        //     The text
        //
        //   font:
        //     The text font
        //
        //   color:
        //     The text color
        //
        //   fillColor:
        //     The background color
        //
```

```
//   textAlign:
//     The text alignment (used to clip the image when the background is transparent)
//
//   rotateFlip:
//     The rotation and flip to apply to the bitmap
//
//   antialias:
//     If true, the text is antialiased
//
// Returns:
//     The text bitmap
    protected Bitmap GetTextImage(string text, Font font, Color color, Color fillColor,
ContentAlignment textAlign, RotateFlipType rotateFlip, bool antialias = true);
//
// Summary:
//     Tells if the current devDept.Eyeshot.Control.Workspace.ActionMode requires dragging.
//
// Returns:
//     True if the current devDept.Eyeshot.Control.Workspace.ActionMode requires dragging.
    protected bool IsDragAction();
    protected override bool IsInputKey(Keys keyData);
//
// Summary:
//     Tells if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     selection ones.
//
// Returns:
//     True if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     selection ones.
    protected bool IsSelectAction();
//
```

```
// Summary:
//     Tells if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByBox ones.
//
// Returns:
//     True if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByBox ones.
protected bool IsSelectByBoxAction();
//
// Summary:
//     Tells if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByPick ones.
//
// Returns:
//     True if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByPick ones.
protected bool IsSelectByPickAction();
//
// Summary:
//     Tells if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByPolygon ones.
//
// Returns:
//     True if the current devDept.Eyeshot.Control.Workspace.ActionMode is one of the
//     ByPolygon ones.
protected bool IsSelectByPolygonAction();
//
// Summary:
//     Occurs every timer tick. Call base class method when overriding.
protected virtual void OnAnimationTimerTick(object stateInfo);
protected override void OnDoubleClick(EventArgs e);
```

```
        protected override void OnGotFocus(EventArgs e);
        protected override void OnHandleCreated(EventArgs e);
        protected override void OnHandleDestroyed(EventArgs e);
        protected override void OnKeyDown(KeyEventArgs e);
        protected override void OnKeyUp(KeyEventArgs e);
        //
        // Summary:
        //     Synchronize hasFocus parameter, reset action, redraw the control.
        protected override void OnLostFocus(EventArgs e);
        protected override void OnMouseClick(MouseEventArgs e);
        protected override void OnMouseDown(MouseEventArgs e);
        protected override void OnMouseEnter(EventArgs e);
        protected override void OnMouseLeave(EventArgs e);
        protected override void OnMouseMove(MouseEventArgs e);
        protected override void OnMouseUp(MouseEventArgs e);
        protected override void OnMouseWheel(MouseEventArgs e);
        protected override void OnPaint(PaintEventArgs e);
        protected override void OnPaint();
        protected override void OnResize(EventArgs e);
        protected void PaintDesignError(System.Drawing.Graphics g);
        protected void PostRemoveJitteringForFastZPR(RenderContextBase renderContext);
        protected void PreRemoveJitteringForFastZPR(DrawSceneParams myParams, bool
removeSceneTransformation = false);
        //
        // Summary:
        //     Selects each entity crossing the specified selection polygon.
        //     This method is deprecated. Use the overload with the
devDept.Eyeshot.Control.SelectionChangedEventArgs
        //     parameter.
        //
        // Parameters:
        //   selectionPolygon:
```

```
        //     Selection polygon in camera screen coordinates
        //
        //   invert:
        //     If true, inverts current selection status
        //
        //   added:
        //     The list of entity indices of entities added to selection
        //
        //   removed:
        //     The list of entity indices of entities removed from selection
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        [Obsolete("Use the overload with the SelectionChangedEventArgs parameter.")]
        protected virtual void ProcessSelectionByPolygon(List<Point2D> selectionPolygon, bool invert,
out int[] added, List<int> removed);
        //
        // Summary:
        //     Selects each entity completely enclosed in the specified selection rectangle.
        //     This method is deprecated. Use the overload with the
devDept.Eyeshot.Control.SelectionChangedEventArgs
        //     parameter.
        //
        // Parameters:
        //   selectionPolygon:
        //     Selection polygon in screen coordinates
        //
        //   added:
        //     The list of entity indices of entities added to selection
        //
        //   removed:
        //     The list of entity indices of entities removed from selection
```

```
        //
        //   invert:
        //     If true, inverts current selection status
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        [Obsolete("Use the overload with the SelectionChangedEventArgs parameter.")]
        protected virtual void ProcessSelectionByPolygonEnclosed(List<Point2D> selectionPolygon, bool
invert, out int[] added, List<int> removed);
        //
        // Summary:
        //     Selects each visible entity in the specified selection polygon.
        //     This method is deprecated. Use the overload with the
devDept.Eyeshot.Control.SelectionChangedEventArgs
        //     parameter.
        //
        // Parameters:
        //   selectionPolygon:
        //     Selection polygon in screen coordinates
        //
        //   added:
        //     The list of entity indices of entities added to selection
        //
        //   removed:
        //     The list of entity indices of entities removed from selection
        //
        //   invert:
        //     If true, inverts current selection status
        //
        // Remarks:
        //     Call Control.Invalidate(System.Drawing.Region) to see the effect of this function.
        [Obsolete("Use the overload with the SelectionChangedEventArgs parameter.")]
```

```
protected virtual void ProcessSelectionByPolygonVisibleOnly(List<Point2D> selectionPolygon, bool invert, out int[] added, List<int> removed);

protected virtual bool PropagateAttributesAndProcessBlockReferenceForSelection(DrawEntitiesParams myParams, Entity ent, GfxAttributesWire originalAttributes, WorkspaceDrawForSelectionCallback callBack);

//

// Summary:

//     Removes the context menu items related to the Mouse3D.

protected virtual void RemoveMouse3DContextMenuItems();

//

// Summary:

//     Resizes the Viewports.

protected void ResizeViewports();

protected bool SimplifyOnFastZPR(DrawSceneParams myParams);

//

// Summary:

//     Sorts the transparent entities from the farthest to the nearest to the camera

//     for better rendering. Don't use if devDept.Eyeshot.Control.Design.MinimumFramerate

//     is enabled.

//

// Parameters:

//   viewport:

//     The viewport

//

// Returns:

//     The sorted list of entities.

//

// Remarks:

//     It can be called in an Workspace derived class in the devDept.Eyeshot.Control.Workspace.DrawViewport(devDept.Eyeshot.DrawSceneParams)

//     method when devDept.Eyeshot.Control.Workspace.AccurateTransparency is false.

protected IList<Entity> SortEntitiesForTransparency(Viewport viewport, IList<Entity> ents);
```

```
//
// Summary:
//     Gets the color used to draw the given non-current entity.
//
// Parameters:
//   entity:
//     The entity being processed
//
//   color:
//     The color of the entity
//
// Returns:
//     The color used to draw the entity not belonging to the current BlockReference.
//
// Remarks:
//     See
devDept.Eyeshot.Control.Workspace.SetCurrent(devDept.Eyeshot.Entities.BlockReference,System.Bo
olean)
    protected internal virtual Color ComputeNonCurrentEntityColor(Entity entity, Color color, bool
edge = false, bool forBlending = false);
//
// Summary:
//     Draws the Background of the Workspace control.
    protected internal virtual void DrawBackground(DrawSceneParams myParams);
//
// Summary:
//     Draws an image on the devDept.Eyeshot.Control.Workspace's client area.
//
// Parameters:
//   x:
//     The image's x corrdinate
//
```

```
        //   y:
        //     The image's y coordinate (from the bottom of the viewport)
        //
        //   image:
        //     The image to draw
        //
        // Remarks:
        //     Alpha channel is supported. Use the
System.Drawing.Image.RotateFlip(System.Drawing.RotateFlipType)
        //     method to flip the image upside down.
        protected internal void DrawImage(int x, int y, Bitmap image);
        protected internal virtual void DrawOMWithPreview(DrawSceneParams data);
        //
        // Summary:
        //     Draws a text on the devDept.Eyeshot.Control.Workspace's client area.
        //
        // Parameters:
        //   x:
        //     The text x coordinate
        //
        //   y:
        //     The text y coordinate (from the bottom of the viewport)
        //
        //   text:
        //     The text string to draw
        //
        //   textFont:
        //     The text font
        //
        //   textColor:
        //     The text color
```

```
        //
        //   fillColor:
        //     The background color (can be semi-transparent)
        //
        //   textAlign:
        //     The text alignment
        //
        //   rotateFlip:
        //     The text rotation
        //
        // Returns:
        //     The size of the text bitmap
        //
        // Remarks:
        //     For better performances use
devDept.Eyeshot.Control.Workspace.GetTextImage(System.String,System.Drawing.Font,System.Drawing.Color,System.Drawing.Color,System.Drawing.ContentAlignment,System.Drawing.RotateFlipType,System.Boolean),
        //     create the texture and draw it directly.
        protected internal Size DrawText(int x, int y, string text, Font textFont, Color textColor, Color fillColor, ContentAlignment textAlign, RotateFlipType rotateFlip);
        //
        // Summary:
        //     Draws a text on the devDept.Eyeshot.Control.Workspace's client area.
        //
        // Parameters:
        //   x:
        //     The text's x coordinate
        //
        //   y:
        //     The text's y coordinate (from the bottom of the viewport)
        //
```

```
        //   text:
        //     The text string to draw
        //
        //   textFont:
        //     The text font
        //
        //   textColor:
        //     The text color
        //
        //   textAlign:
        //     The text alignment
        //
        // Returns:
        //     The size of the text bitmap
        protected internal Size DrawText(int x, int y, string text, Font textFont, Color textColor,
ContentAlignment textAlign);
        //
        // Summary:
        //     Draws a text on the devDept.Eyeshot.Control.Workspace's client area.
        //
        // Parameters:
        //   x:
        //     The text's x coordinate
        //
        //   y:
        //     The text's y coordinate (from the bottom of the viewport)
        //
        //   text:
        //     The text string to draw
        //
        //   textFont:
```

```
        //     The text font
        //
        //   textColor:
        //     The text color
        //
        //   fillColor:
        //     The background color (can be semi-transparent)
        //
        //   textAlign:
        //     The text alignment
        //
        // Returns:
        //     The size of the text bitmap
        //
        // Remarks:
        //     For better performances use the
devDept.Eyeshot.Control.Workspace.GetTextImage(System.String,System.Drawing.Font,System.Drawi
ng.Color,System.Drawing.Color,System.Drawing.ContentAlignment,System.Drawing.RotateFlipType,Sy
stem.Boolean)
        //     to get the text image, then create a texture and draw the texture directly
        protected internal Size DrawText(int x, int y, string text, Font textFont, Color textColor, Color
fillColor, ContentAlignment textAlign);
        //
        // Summary:
        //     Draws a text on the devDept.Eyeshot.Control.Workspace's client area.
        //
        // Parameters:
        //   x:
        //     The text's x coordinate
        //
        //   y:
        //     The text's y coordinate (from the bottom of the viewport)
```

```
//
//   text:
//     The text string to draw
//
//   textFont:
//     The text font
//
//   textColor:
//     The text color
//
//   outlineColor:
//     The outline color
//
//   outlineThickness:
//     The outline thickness
//
//   textAlign:
//     The text alignment
//
// Remarks:
//     For better performances use
devDept.Eyeshot.Control.Workspace.GetTextOutlinedImage(System.String,System.Drawing.Font,Syst
em.Drawing.Color,System.Drawing.Color,System.Drawing.RotateFlipType,System.Single),
//     create the texture and draw it directly.
    protected internal void DrawTextOutlined(int x, int y, string text, Font textFont, Color textColor,
Color outlineColor, float outlineThickness, ContentAlignment textAlign);
//
// Summary:
//     Draws the viewport.
//
// Parameters:
//   myParams:
```

```
//     The parameters
protected internal virtual void DrawViewport(DrawSceneParams myParams);
//
// Summary:
//     Returns the viewport under the given mouse position.
//
// Parameters:
//   mousePos:
//     The mouse position
//
// Returns:
//     The viewport under the mouse.
protected internal virtual int GetViewportUnderMouse(System.Drawing.Point mousePos);
//
// Summary:
//     Returns the indices of the entities read from the back buffer in the selectionBox.
//
// Parameters:
//   viewport:
//     The viewport
//
//   stride:
//     The number of bytes per image row
//
//   bpp:
//     Bytes per pixel
//
//   selectionBox:
//     Selection rectangle in screen coordinates
//
//   firstOnly:
```

```
        //     If true, returns the first entity that is found in the selectionBox
        //
        //   rgbValues:
        //     The array of pixels drawn in false colors
        //
        // Returns:
        //     The array of the indices of the visible entities found in the selection box.
        protected internal virtual int[] GetVisibleEntitiesFromBackBuffer(Viewport viewport, byte[]
rgbValues, int stride, int bpp, Rectangle selectionBox, bool firstOnly);
        //
        // Summary:
        //     Tells if the projected point, whose square distance from the mouse cursor is
        //     squareDistance, is closer than the currentMinimumSquareDistance.
        //
        // Parameters:
        //   projectedPt:
        //     The point in projected Camera coordinates
        //
        //   squareDistance:
        //     The 2D square distance from the mouse position
        //
        //   currentMinimumSquareDistance:
        //     The current minimum 2D square distance from the mouse position
        //
        // Returns:
        //     True if the point is closer, false otherwise.
        //
        // Remarks:
        //     This method is used by the
devDept.Eyeshot.Control.Workspace.FindClosestVertex(System.Drawing.Point,System.Double,devDe
pt.Geometry.Point3D@)
        //     to compute the closest vertex to the mouse in 2D. Override this method to consider
```

```
    //     also the Z of the projectedPt.
    protected internal virtual bool IsCloserVertex(Point3D projectedPt, double squareDistance,
double currentMinimumSquareDistance);
    //
    // Summary:
    //     Occurs every devDept.Eyeshot.Control.Workspace.Mouse3DButtonDown event.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     The event data
    protected internal virtual void OnMouse3DButtonDown(object sender, ButtonEventArgs e);
    //
    // Summary:
    //     Occurs every devDept.Eyeshot.Control.Workspace.Mouse3DButtonUp event.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     The event data
    protected internal virtual void OnMouse3DButtonUp(object sender, ButtonEventArgs e);
    //
    // Summary:
    //     Occurs every devDept.Eyeshot.Control.Workspace.Mouse3DMove event.
    //
    // Parameters:
    //   sender:
```

```
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMouse3DMove(object sender, MoveEventArgs e);
//
// Summary:
//     Occurs every devDept.Eyeshot.Control.Workspace.MultiTouchClick event.
//
// Parameters:
//   sender:
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMultiTouchClick(object sender, MouseEventArgs e);
//
// Summary:
//     Occurs every devDept.Eyeshot.Control.Workspace.MultiTouchDoubleClick event.
//
// Parameters:
//   sender:
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMultiTouchDoubleClick(object sender, MouseEventArgs e);
//
// Summary:
//     Occurs every devDept.Eyeshot.Control.Workspace.MultiTouchDown event.
//
```

```
// Parameters:
//   sender:
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMultiTouchDown(object sender, TouchEventArgs e);
//
// Summary:
//     Occurs every devDept.Eyeshot.Control.Workspace.MultiTouchMove event.
//
// Parameters:
//   sender:
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMultiTouchMove(object sender, TouchEventArgs e);
//
// Summary:
//     Occurs every devDept.Eyeshot.Control.Workspace.MultiTouchUp event.
//
// Parameters:
//   sender:
//     The source of the event.
//
//   e:
//     The event data
protected internal virtual void OnMultiTouchUp(object sender, TouchEventArgs e);
//
// Summary:
```

```
//     Paints the viewport surface without redrawing the whole scene.
protected internal void PaintBackBuffer();
protected internal void RestoreCursor(CursorContainer prev);
protected internal CursorContainer SetWaitCursor();
//
// Summary:
//     Starts a sequence of Zoom-Pan-Rotate movements if the proper keys/mouse buttons
//     are pressed.
//
// Parameters:
//   e:
//     The mouse event args
//
//   viewport:
//     The viewport
protected internal void StartZoomPanRotate(MouseEventArgs e, Viewport viewport);
//
// Summary:
//     Swaps the front and back buffers.
protected internal void SwapBuffers();


//
// Summary:
//     The assembly selection type.
public enum assemblySelectionType
{
   //
   // Summary:
   //     Selects the entities at the current BlockReference level
   Branch = 0,
   //
```

```
        // Summary:
        //     Selects the entities at the deepest level of the BlockReference hierarchy
        //
        // Remarks:
        //     Not supported by devDept.Eyeshot.Control.Workspace.AccurateTransparency and all
        //     geometry selection types.
        Leaf = 1
    }

    //
    // Summary:
    //     Structure that holds the Cursor (or Cursors) used by the control.
    public struct CursorContainer : ICursorContainer
    {
        //
        // Summary:
        //     Main control Cursor.
        public Cursor Cursor;

        public CursorContainer(Cursor cursor);
    }

    //
    // Summary:
    //     Provides data for the devDept.Eyeshot.Control.Workspace.ErrorOccurred event.
    public class ErrorOccurredEventArgs : EventArgs
    {
        public ErrorOccurredEventArgs(string message, string stackTrace, EntityGraphicsData
graphicsDataWithError);

        //
```

```
    // Summary:
    //     Gets the error message.
    public string Message { get; }
    //
    // Summary:
    //     Gets the stack trace of the error.
    public string StackTrace { get; }
    //
    // Summary:
    //     Gets the object whose not compiled status caused the error.
    public object ObjectNotCompiled { get; }
}
//
// Summary:
//     Class that holds the arguments for devDept.Eyeshot.Control.Workspace.ViewChanged
//     event.
public class ViewChangedEventArgs : HandledEventArgs
{
    //
    // Summary:
    //     Standard constructor.
    //
    // Parameters:
    //   viewType:
    //     One of the devDept.Eyeshot.viewType values that indicates which view was set
    public ViewChangedEventArgs(viewType viewType);

    //
    // Summary:
    //     Gets which view was set.
    public viewType ViewType { get; }
```

```
    }

    public delegate void BoundingBoxChangedHandler(object sender);
    //
    // Summary:
    //     Represents the method that will handle the
devDept.Eyeshot.Control.Workspace.CameraMoveBegin
    //     and devDept.Eyeshot.Control.Workspace.CameraMoveEnd events.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     A devDept.Eyeshot.Control.CameraMoveEventArgs object that contains the event
    //     data.
    public delegate void CameraMoveEventHandler(object sender, CameraMoveEventArgs e);
    public delegate void DesignTimeFuncHandler();
    //
    // Summary:
    //     Represents the method that will handle the
devDept.Eyeshot.Control.Workspace.ErrorOccurred
    //     event.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     A devDept.Eyeshot.Control.Workspace.ErrorOccurred that contains the event data.
    public delegate void ErrorEventHandler(object sender, ErrorOccurredEventArgs args);
    //
```

```
    // Summary:
    //     Represents the method that will handle the
devDept.Eyeshot.Control.Workspace.NavigationTimerTick
    //     event.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     The event data.
    public delegate void NavigationTimerHandler(object sender, EventArgs e);
    //
    // Summary:
    //     Represents the method that will handle the
devDept.Eyeshot.Control.Workspace.SelectionChanged
    //     and devDept.Eyeshot.Control.Viewport.LabelSelectionChangedevent.
    //
    // Parameters:
    //   sender:
    //     The source of the event.
    //
    //   e:
    //     A devDept.Eyeshot.Control.SelectionChangedEventArgs object that contains the
    //     event data.
    public delegate void SelectionChangedEventHandler(object sender, SelectionChangedEventArgs
e);
    //
    // Summary:
    //     Represents the method that will handle the
devDept.Eyeshot.Control.Workspace.ViewChanged
    //     event.
```

```
        //
        // Parameters:
        //   sender:
        //     The source of the event
        //
        //   e:
        //     A devDept.Eyeshot.Control.Workspace.ViewChangedEventArgs object that contains
        //     the event data
        public delegate void ViewChangedEventHandler(object sender, ViewChangedEventArgs e);
    }
}
```