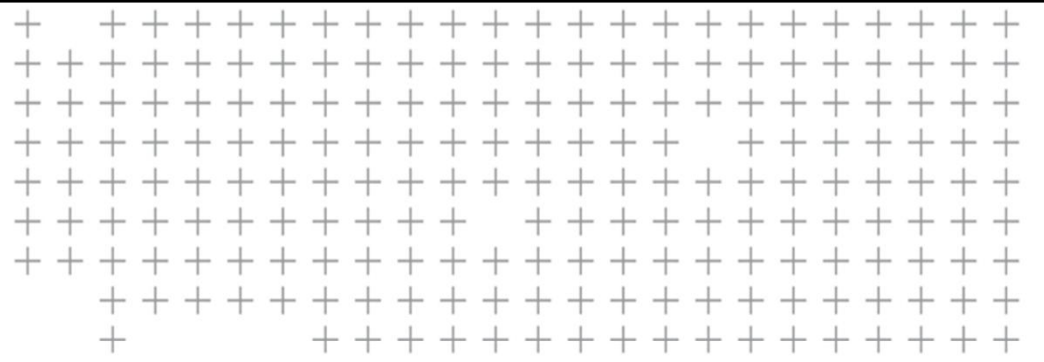


# Drawing API



Introduction



# What is the Drawing API

# Drawing API

## § Toolkit you need to connect to Tekla drawings

- Extend existing functionality
- Automate tasks
- Create new functionality
- Create new drawing type objects
- Personalize the presentation of model graphics

## § Benefits

- Automate tedious and unpleasant tasks
- Change the look and control over parts and annotations
- Ultimately save considerable time and money

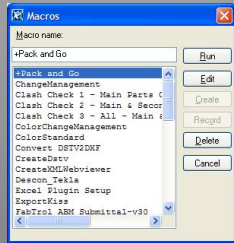
# Common Use Cases



## § Drawing API

- Access the drawing list.
- Create, modify and delete.
  - § GA drawings and views in any drawing.
  - § Dimensions, text and other basic objects.
- Interact with the user.
  - § Get currently selected drawing objects.
  - § Prompt user to pick objects and locations.
  - § Select and highlight objects for the user.
- Access model objects from the drawing.
  - § Select the parts found in a drawing in the model.

# Implementation Categories

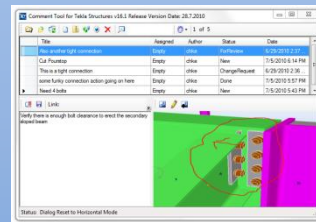
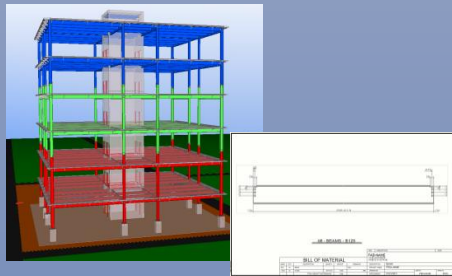


## Macros (\* .cs)

Recorded scripts calling commands in the Tekla Structures interface for Model and Drawings. (C#, API Library can be used here)

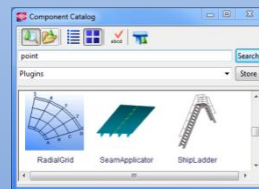
## Drawing API

Create, Modify, Delete, Extract Information, Set User Attributes, Navigate, Call Macros.



## External Applications (\* .exe)

Tools that connect and interact with Tekla Structures



## Plugins (\* .dll)

Parametric, run inside Tekla Structures, fast.

# Drawing API

## § Toolkit you need to connect to Tekla drawings

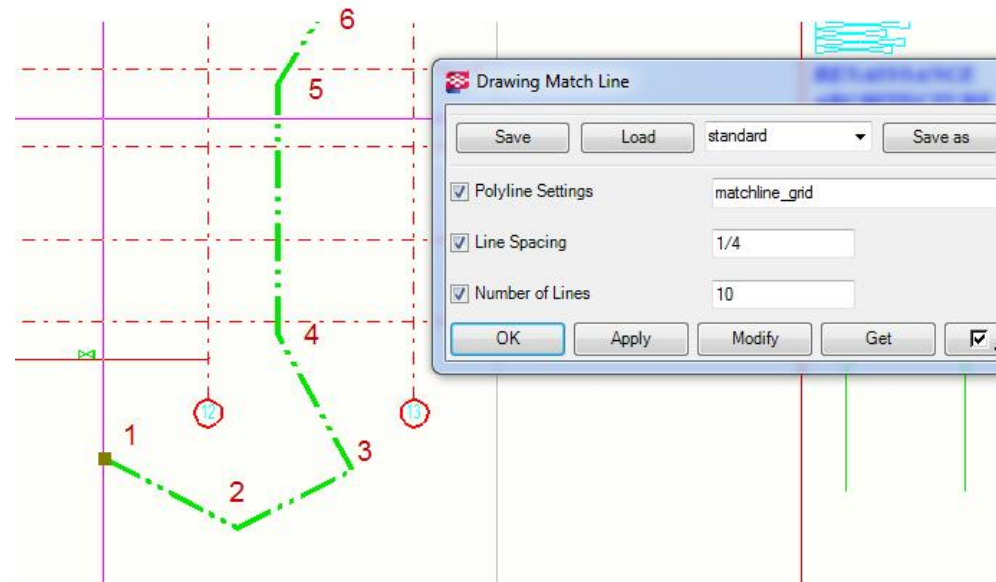
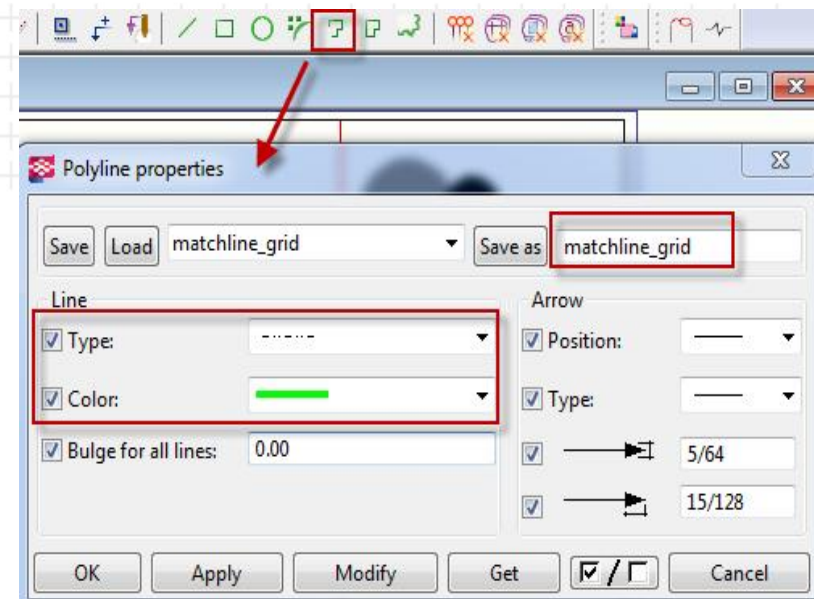
- Extend existing functionality
- Automate tasks
- Create new functionality
- Create new drawing type objects
- Personalize the presentation of model graphics

## § Benefits

- Automate tedious and unpleasant tasks
- Change the look and control over parts and annotations
- Ultimately save considerable time and money

# Drawing Match Line

- § Creates thick, custom polyline that is very easy to see on drawings
- § Illustrates where drawings get isolated
- § Multiple lines offset together, but grouped with its own user interface



# Macros : What are they



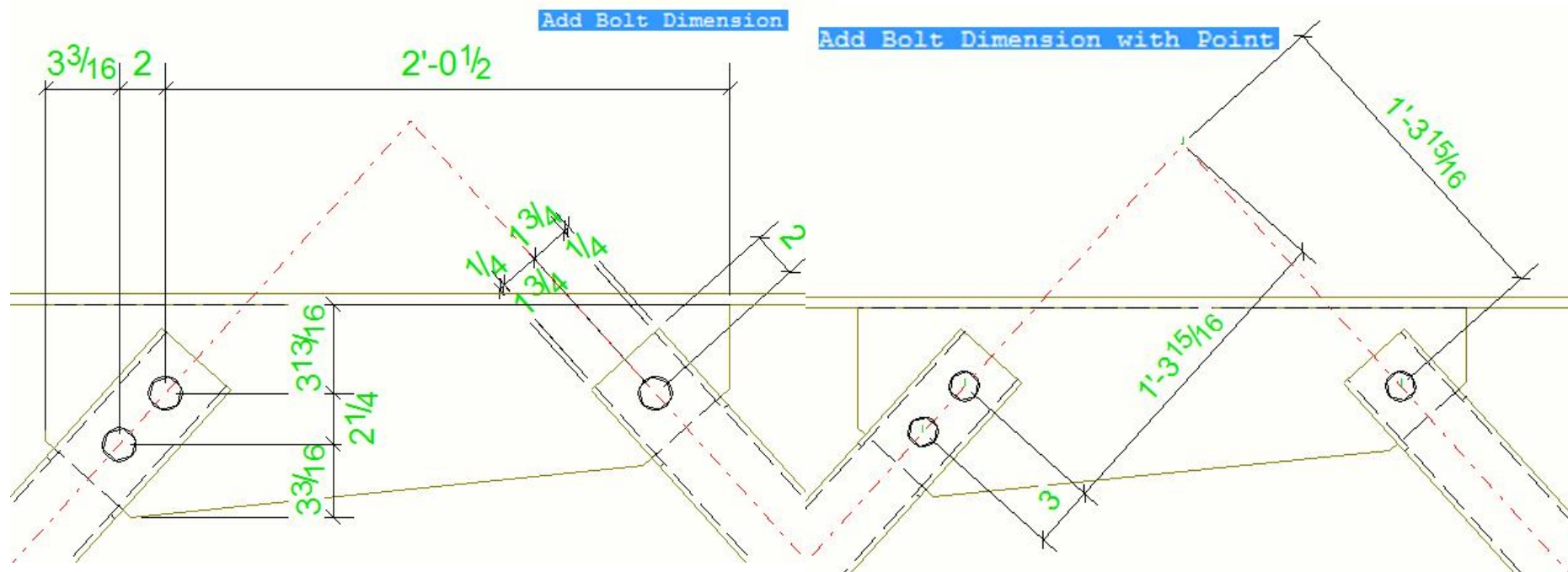
- § Scripts recorded from Tekla directly, can automate some things easily
- § Can run directly un-compiled c# code
- § Independent of versions
- § Easy to create
- § Low maintenance
- § Not good for User Interfaces, but great for smaller tools that do not need dialogs
- § Freeze Tekla Structures until finished running
- § Sample template you can now download under UM 2012



# Examples of Macros

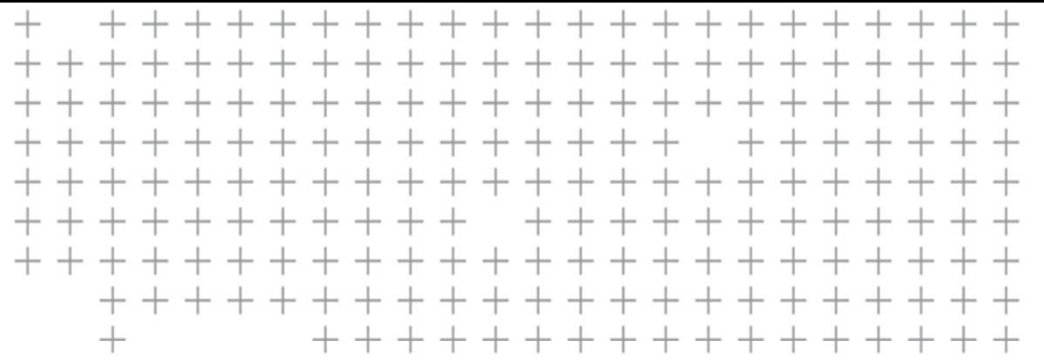
- § Remove Change Clouds\*
- § Re-Open drawing, forces templates to update\*
- § Start Point-Checker, Creates symbols at the physical start end of a model beam into the drawing\*
- § Top in Form Symbol - Creates symbols on the face that is set to the 'up in form' side of the column or other type concrete part
- § Simple, modular dimensioning tools
  - Dimension strand or rebar
  - Dimension bolt group

# Bolt Dimension Tool



# Sample Code In Macro – Reopen drawing

```
public ReopenDrawing()  
{  
    var drawingHandler = new DrawingHandler();  
    var activeDrawing = drawingHandler.GetActiveDrawing();  
    if (activeDrawing != null)  
    {  
        drawingHandler.SaveActiveDrawing();  
        drawingHandler.SetActiveDrawing(activeDrawing, true);  
        return;  
    }  
    Trace.WriteLine("No drawing found, macro failed.");  
}
```



# Drawing Applications

# Drawing Applications : What are they

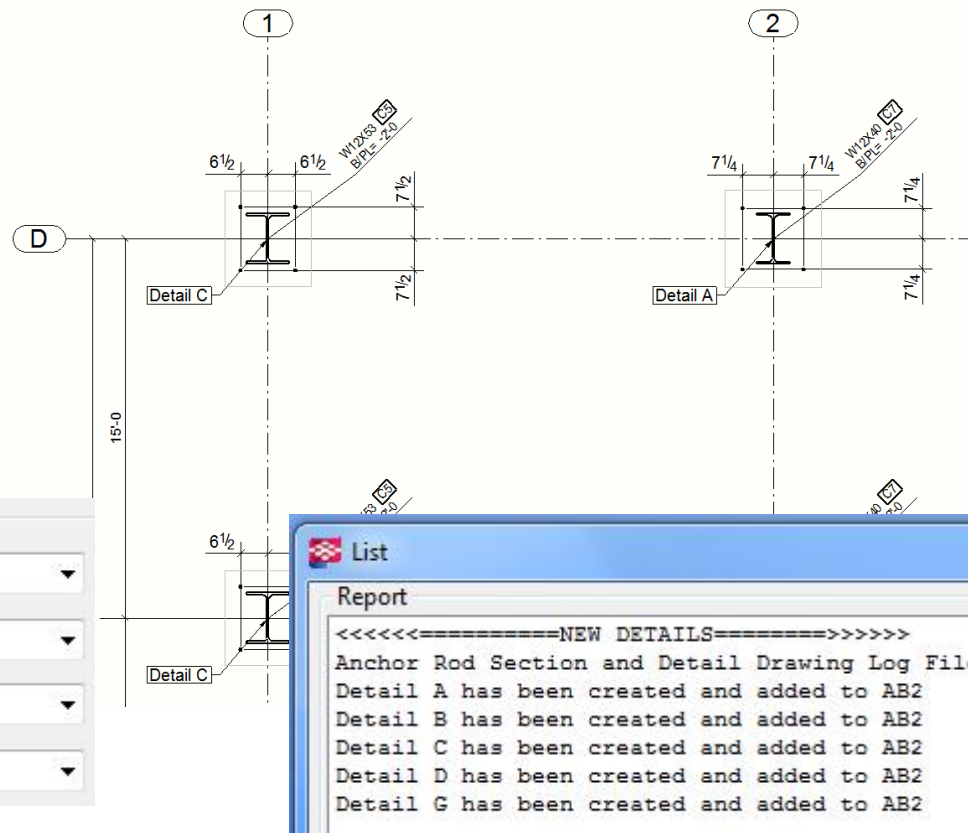
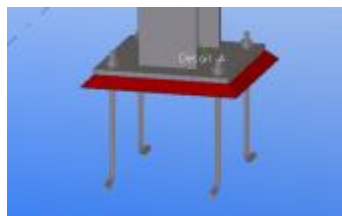


- § Executable programs that run outside Tekla Structures and communicate back and forth between the model and drawings.
- § Slower than plugins, run asynchronously with Tekla
- § Ideal for larger and more complex applications that need real user interfaces
- § Unlimited in use cases and possibilities
- § Can call 'helper' plugins to extract data from the drawing or do smaller tasks

# Anchor Rod Tools



## § Connection Code



### Dimension Settings

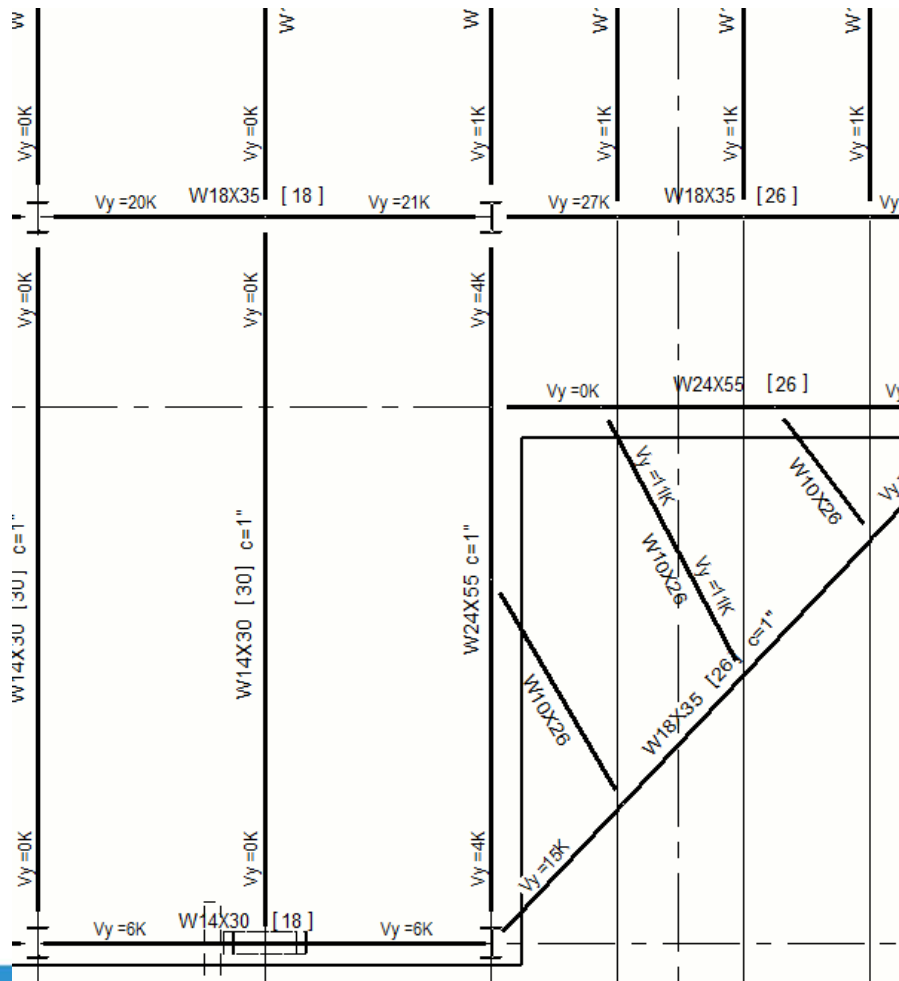
Plan	standard
Projection	AB_Proj
Grout	AB_Grout
Embedment	AB_Embed

### List

#### Report

```
<<<<<=====NEW DETAILS=====>>>>>
Anchor Rod Section and Detail Drawing Log File. Date:8/8/2011 Time:1:26 AM
Detail A has been created and added to AB2
Detail B has been created and added to AB2
Detail C has been created and added to AB2
Detail D has been created and added to AB2
Detail G has been created and added to AB2
```

# Beam Mark



Beam Marks

Save Load shear1 Save as Help...

Position

Horizontal Offset 0

Vertical Offset 0

Settings

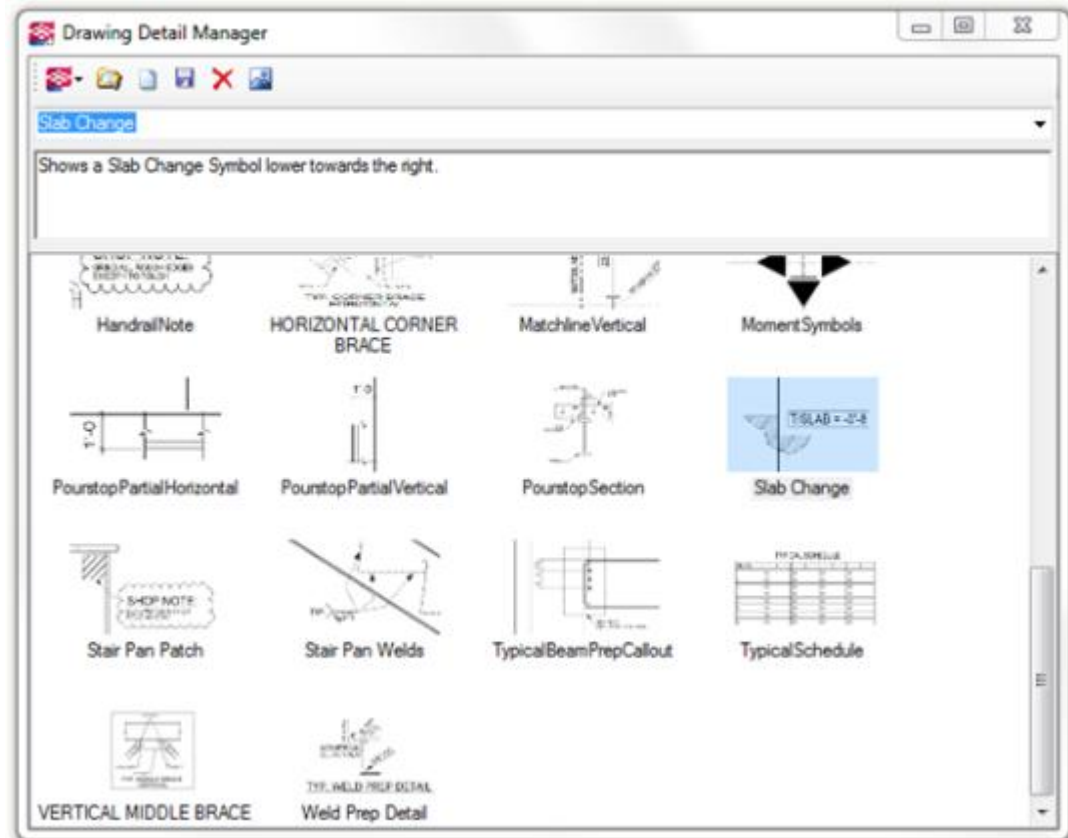
Mark Settings shear\_start

Filter Settings Steel\_Beam

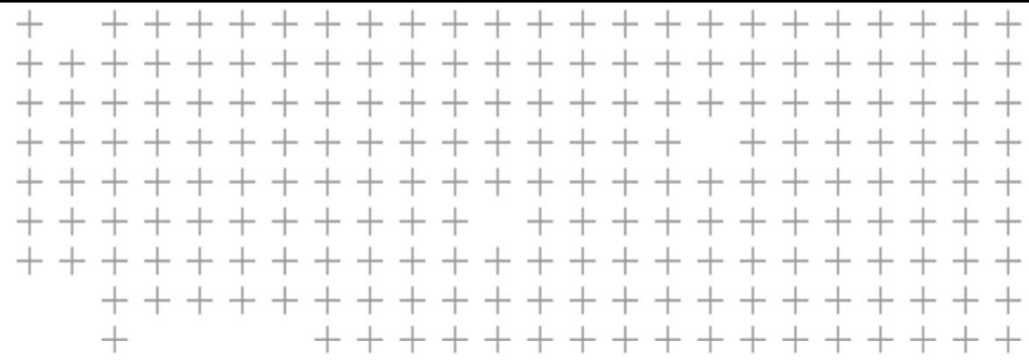
Create Cancel

# Detail Drawing Manager

- § Originated from the request for standard details need.
- § Similar to blocks, but this tool adds real native Tekla lines, hatching, and other 2d objects on top of model graphics
- § Intelligence to model objects is retained

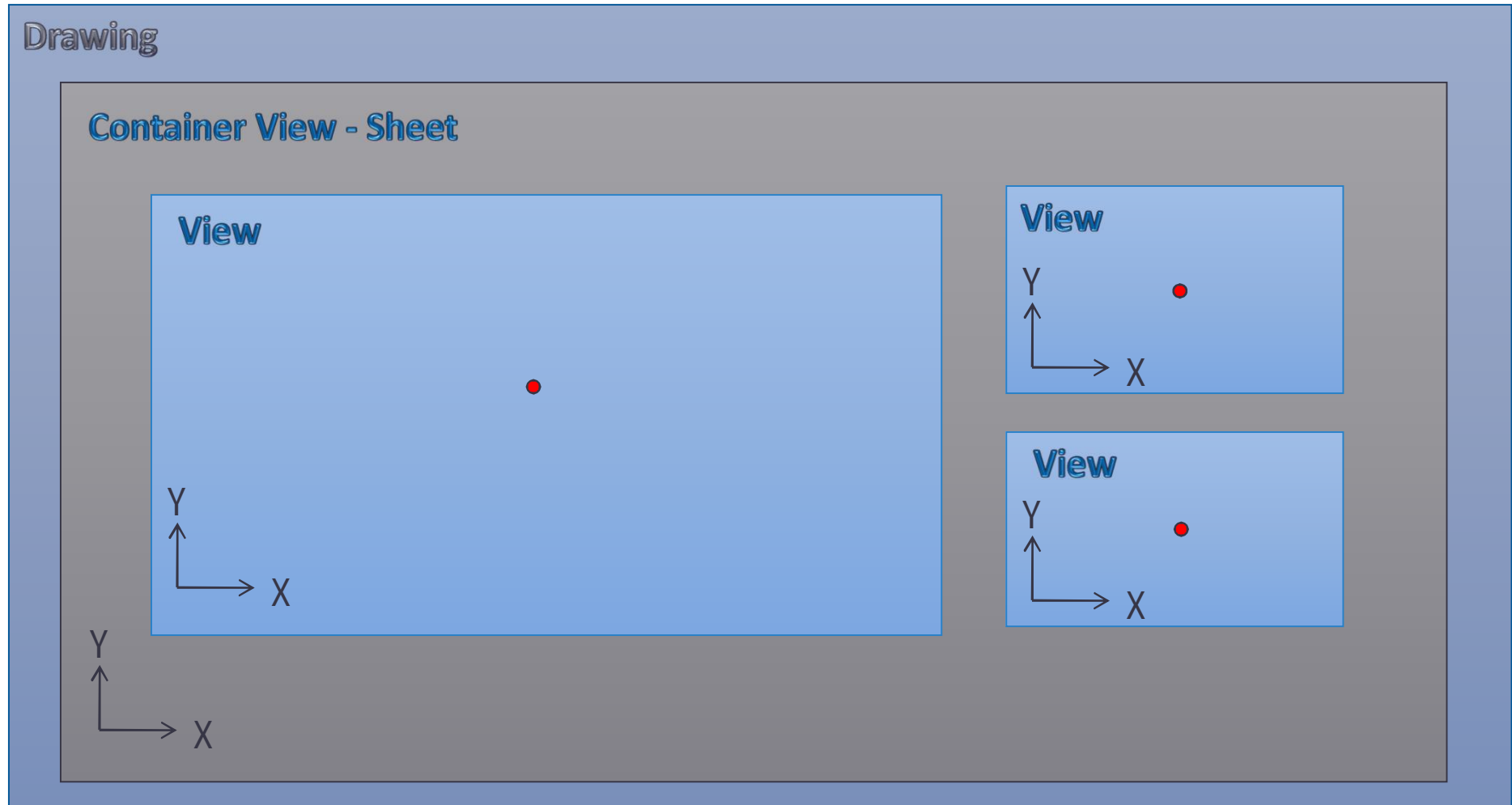






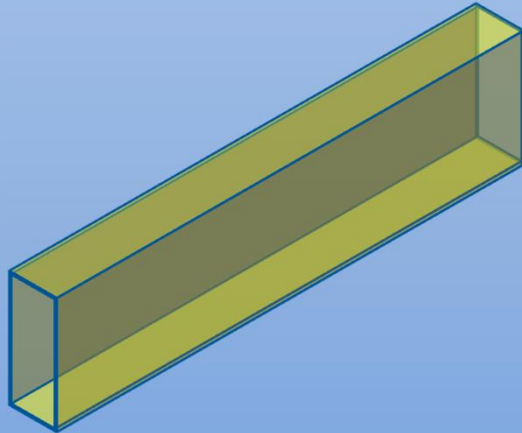
# Key Concepts

# Drawing Architecture (Typical GA Drawing)



# View Architecture

View



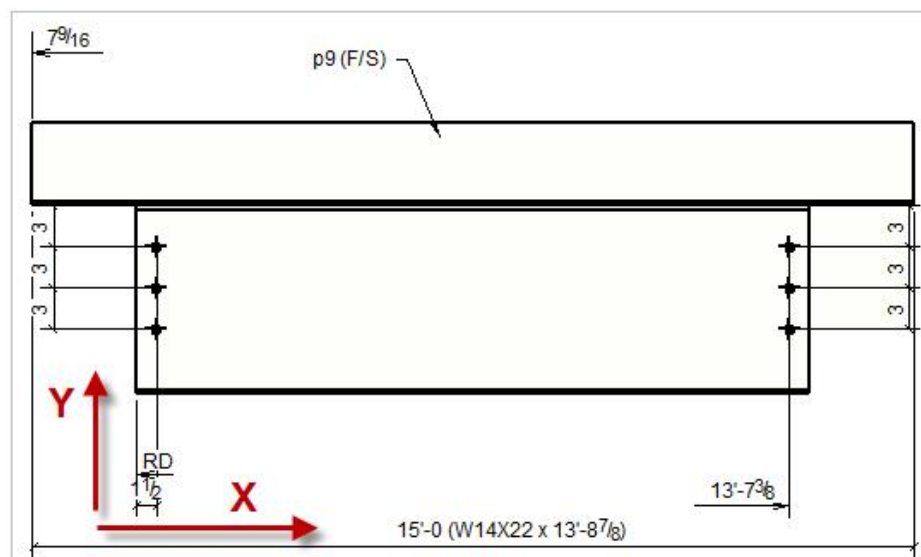
Main defining Geometry

- § View Coordinate System
- § Display Coordinate System
- § Origin
- § Restriction Box (How much of the model you see)

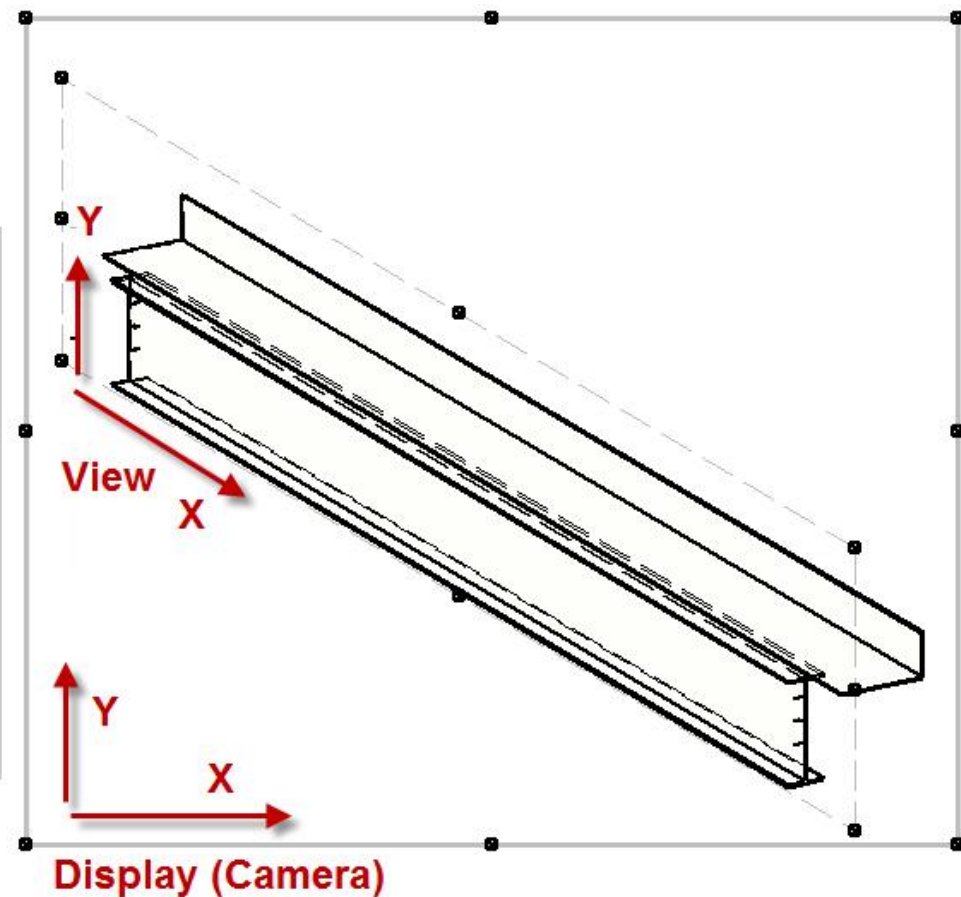
## View Prop. & Methods

- ⇒ GetModelObjects()
- ⇒ GetVisibleAreaRestrictionBoxes()
- ⇒ IsEqual(object)
- ⇒ View(Tekla.Structures.Drawing.Conta
- ⇒ View(Tekla.Structures.Drawing.Conta
- ⇒ View(Tekla.Structures.Drawing.Conta
- ⇒ View(Tekla.Structures.Drawing.Conta
- ⇒ Attributes
- ⇒ DisplayCoordinateSystem
- ⇒ Name
- ⇒ RestrictionBox
- ⇒ ViewCoordinateSystem

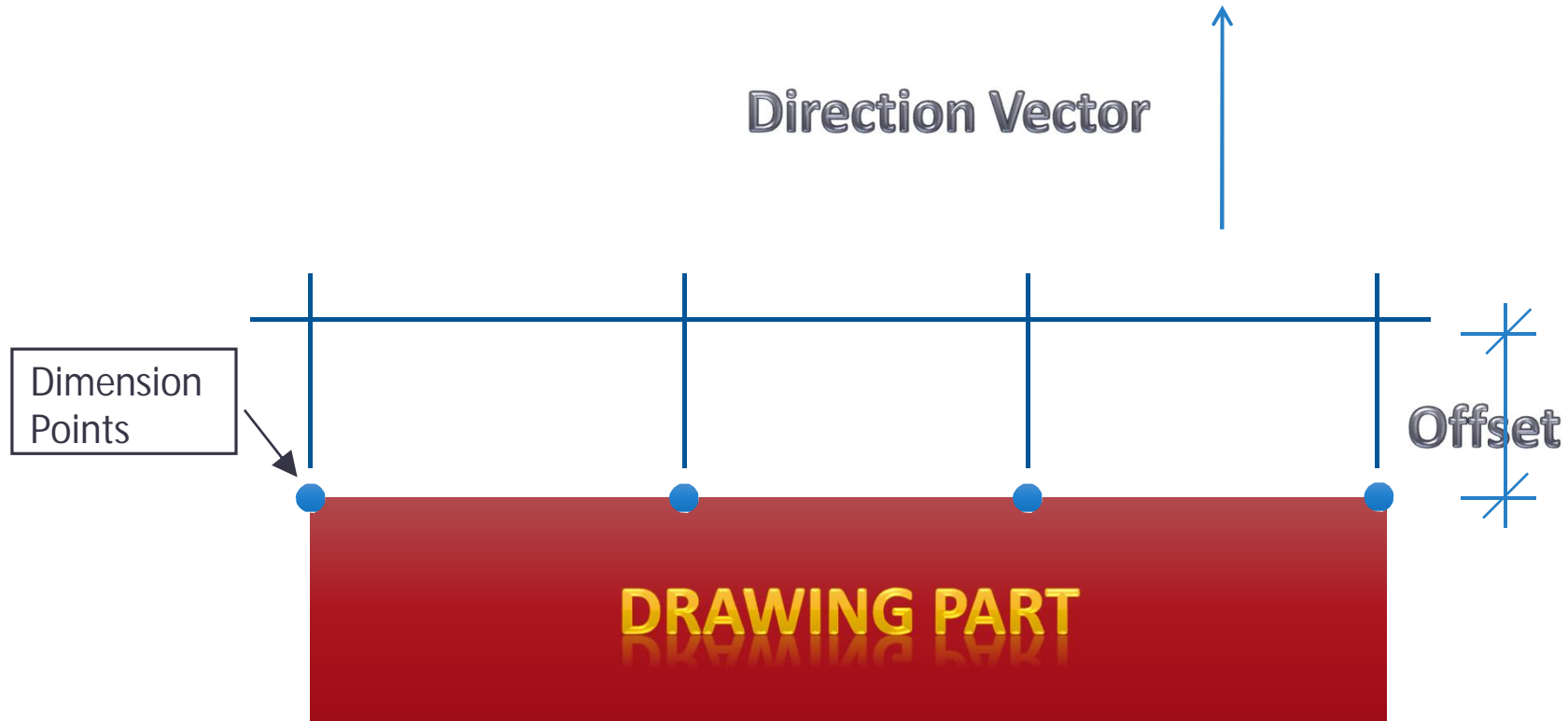
# Examples



**View and Display (Camera) are the same**



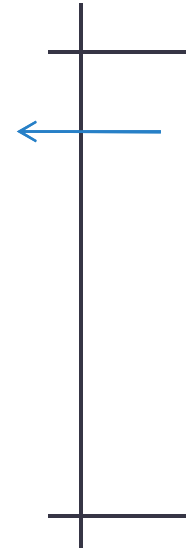
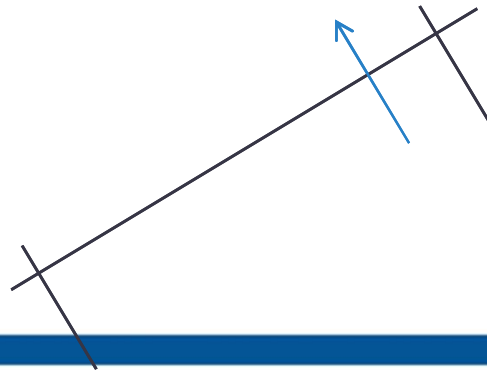
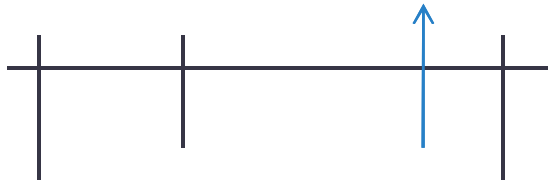
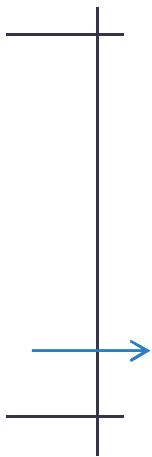
# Dimension Architecture



# Dimension Direction Vectors

§ Sets the up position relative to the direction that is to be dimensioned, always \*90 from dimension line

- Horizontal, Vertical, Skew dimension
- Sets the direction to where dimension is placed
  - § Left, Right, Up, Down from object it is dimensioning
- Set using a Vector (x value, y value, z value)
  - § Relative to DisplayCoordinateSystem



# Drawing objects

## § Are not the same as model objects

- Presentation object that represents the model object in the drawing interface. E.g. Color, accuracy, symbol

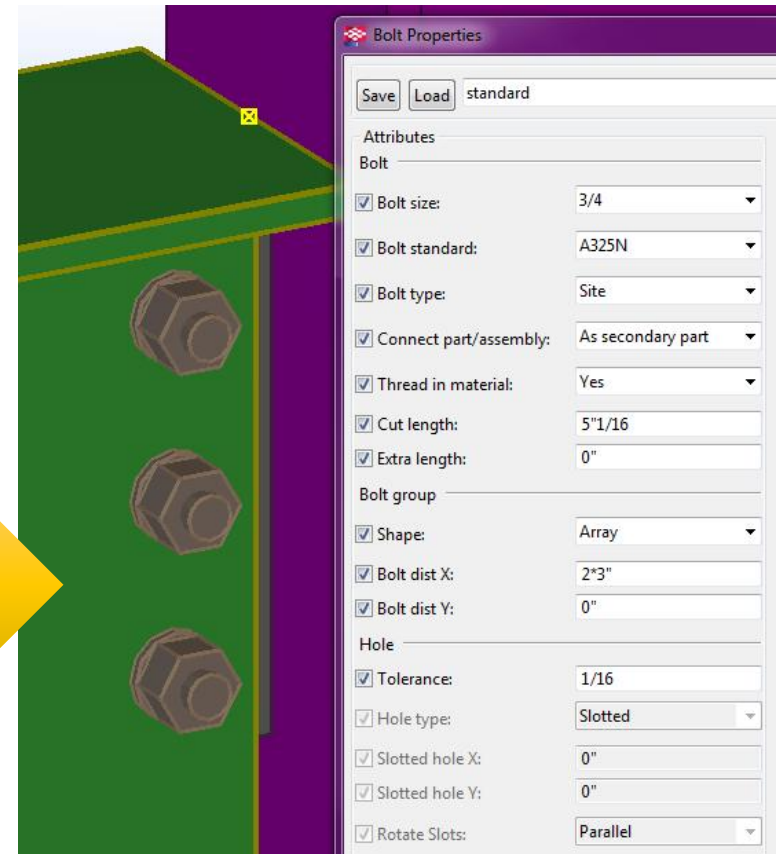
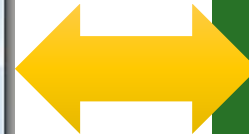
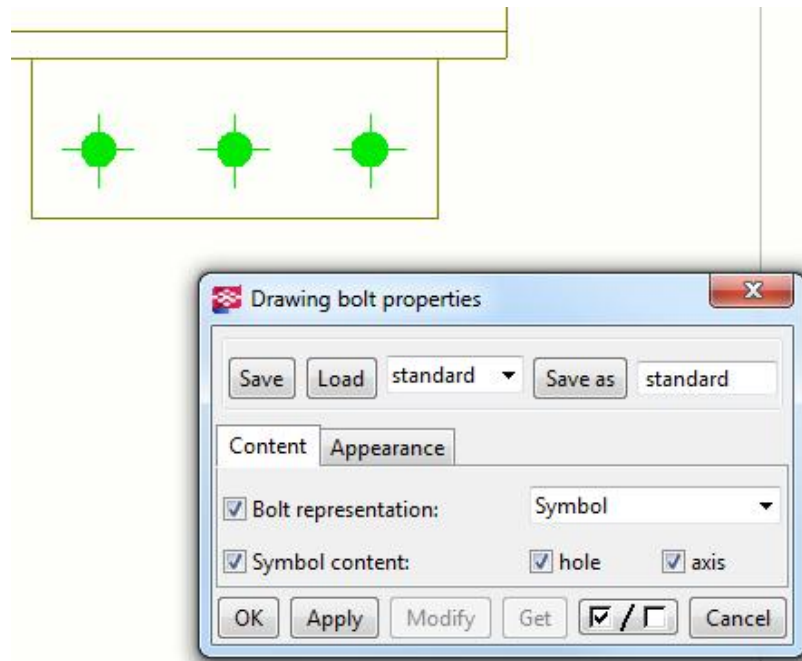
## § Can be many different types of objects including text, lines, marks

## § Can get model object identifier for the model part that the drawing part represents

- You can not change the Model object when a drawing is open, but you can get information from it

# Model API versus the Drawings

- § Drawings are just presentations of the model objects
- § Drawing part settings
- § Model object



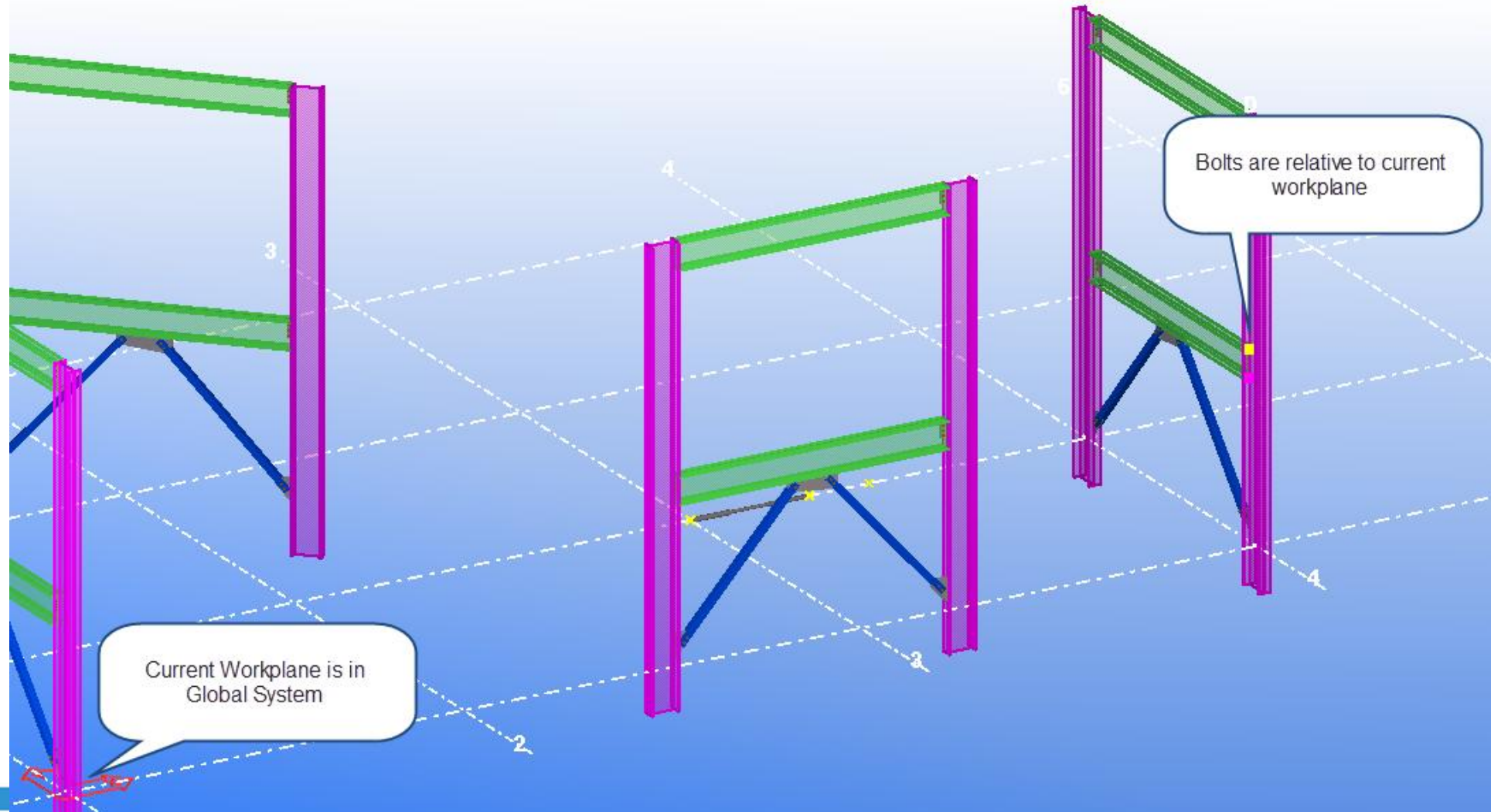


# Get model object from drawing object

- § Drawing Parts have a method to get the model object identifier
- § Make sure custom line types are NOT used

```
var modelBoltGroup = new Model().  
    SelectModelObject(drawingPart.ModelIdentifier) as BoltGroup;  
if (modelBoltGroup == null) return;
```

# Transforming Points to View Local



# Show from coordinate system to local

## Global System

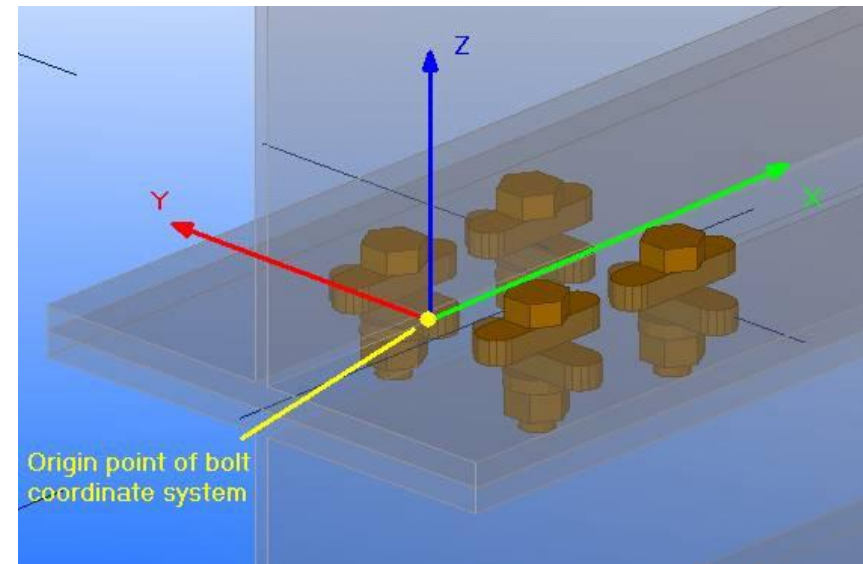
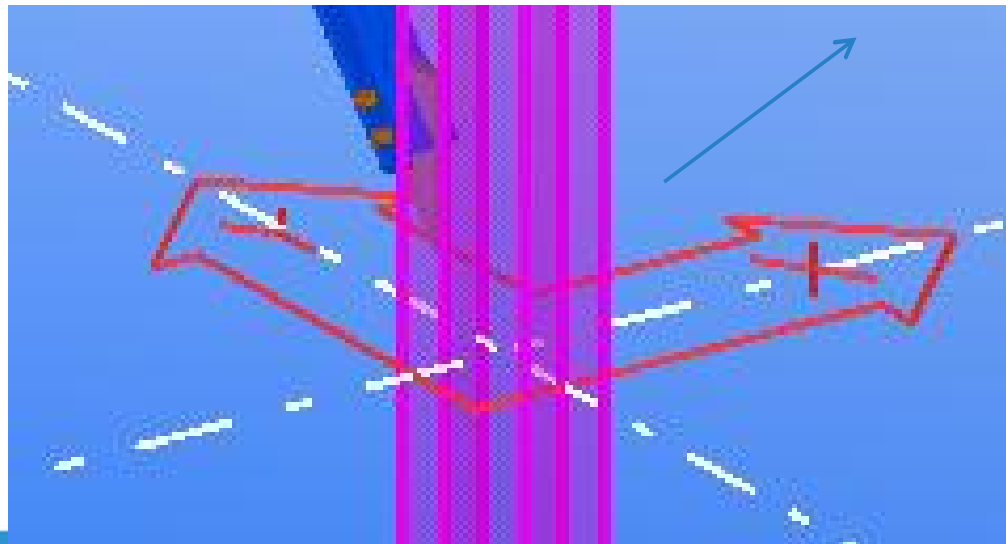
### § Origin

— (640540.24,554707.56,  
5687.32)

## Local System

### § Origin

— (0,0,0)



[Date]

# Drawing Handler

`new DrawingHandler();`

§ Check if a drawing is currently open

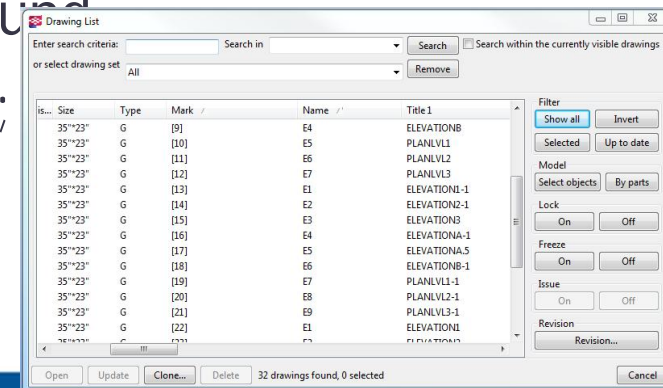
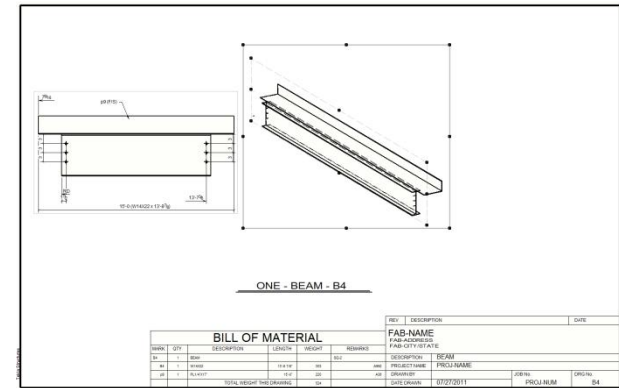
- `GetActiveDrawing()` check if null or Drawing

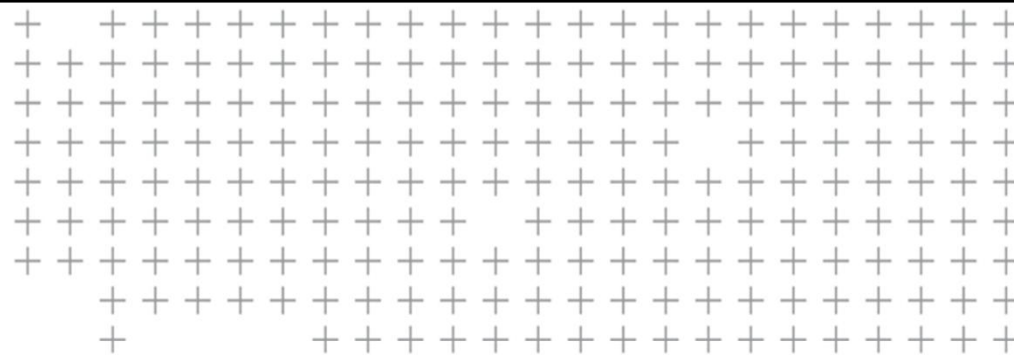
§ Gets Picker for getting input from user

§ Closes, Saves, and Opens drawings

- Can open drawing hidden in background
- `SetActiveDrawing(myDrawing, false);`

§ Get drawings from drawing list





# Drawing API Tips

# Things to Know

- § The UpdateDrawing command is in the DrawingHandler class. This command allows the non-up-to-date drawings in the drawing list to be updated as if used from the user interface.
- § ExcludePartsAccordingToFilter
  - You can set the filter to be used for StraightDimensionSet tags.
  - If an invalid ExcludePartsFilter is given, it will become None.
  - StraightDimensionSet.GetAllExcludePartsAccordingToFilter() queries the available ExcludePartsAccordingToFilters.

# Things to Know

- § AssemblyDrawing, CastUnitDrawing (both by id and by position) and SinglePartDrawing can be created and inserted. Insert always inserts the drawing using the view creation rules. Views are added as defined by the rules.
- § The GetDrawing method in the DrawingObject, lets all objects now know to which drawing they belong.
- § The GetOriginalDrawing method in the ViewBase class lets views now know if they are linked to another drawing or not.



# Things to Know

§ The following drawing objects can be hidden:

- ModelObject (all objects inherited from the ModelObject class)
- WeldMark (the drawing weld mark)
- Text
- Symbol
- MarkBase (all marks)
- GraphicObject (all graphical objects)
- DimensionBase (all single dimensions)
- DimensionSetBase (all dimension sets)



# Things to Know






## § Create preset views on Assembly type Drawings

### View Methods

[View Class](#) [See Also](#) [Send Feedback](#)

The [View](#) type exposes the following members.

#### Methods

	Name	Description
	<a href="#">Create3dView</a>	Generates a new 3d view on the sheet of the given drawing.
	<a href="#">CreateBackView</a>	Generates a new back view on the sheet of the given drawing.
	<a href="#">CreateBottomView</a>	Generates a new bottom view on the sheet of the given drawing.
	<a href="#">CreateFrontView</a>	Generates a new front view on the sheet of the given drawing.
	<a href="#">CreateTopView</a>	Generates a new top view on the sheet of the given drawing.

# Things to Know

- § LoadAttributes sets the FixedViewPlacing property.
- § LoadAttributes also loads the layout of a drawing just as if done from the Tekla Structures user interface.
- § When modifying a view, the drawing plug-ins inside the view are also updated.

# Hideable Feature

§ The Hideable member has the following information:

- `bool IsHidden`, which gives the information if the object is hidden or visible.
- `HideFromDrawingView()` function hides the object in the current view it exists in.
- `HideFromDrawing()` function hides the object in all the views it exists in.
- `ShowInDrawingView()` function shows the object in the current view it exists in.
- `ShowInDrawing()` function shows the object in all the views it exists in.



# Create New Assembly Drawing

§ Create assembly drawings through the API with views and settings for part objects

## AssemblyDrawing Members

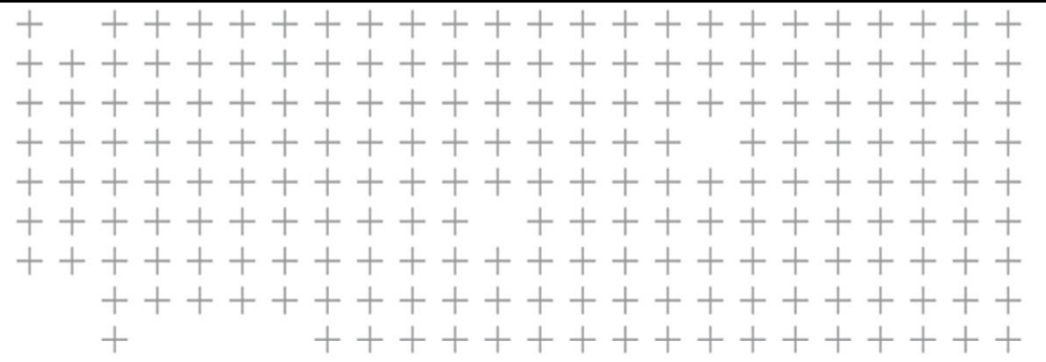
[AssemblyDrawing Class](#) [Constructors](#) [Methods](#) [Properties](#) [See Also](#)

[This is preliminary documentation and is subject to change]

The [AssemblyDrawing](#) type exposes the following members.

### Constructors

Description	Name
Instantiates a new assembly drawing with standard attributes.	<a href="#">AssemblyDrawing(Identifier)</a>
Instantiates a new assembly drawing with standard attributes and with a specified sheet number.	<a href="#">AssemblyDrawing(Identifier, Int32)</a>
Instantiates a new assembly drawing with given attributes.	<a href="#">AssemblyDrawing(Identifier, String)</a>
Instantiates a new assembly drawing with given attributes and with a specified sheet number.	<a href="#">AssemblyDrawing(Identifier, Int32, String)</a>



# Frequently Asked Questions

# Create New GA Drawing

## GADrawing Class

[Members](#) [Example](#) [See Also](#) [Send Feedback](#)

**Namespace:** [Tekla.Structures.Drawing](#)

**Assembly:** Tekla.Structures.Drawing (in Tekla.Structures.Drawing.dll)

### Syntax

#### C#

```
public sealed class GADrawing : Drawing
```

#### Visual Basic

```
Public NotInheritable Class GADrawing  
    Inherits Drawing
```

```
public static void CreateGaDrawing(string name, string settings)  
{  
    //Create new GA Drawing  
    var a3 = new Size(410.0, 287.0);  
    var newDrawing = new GADrawing(settings, a3);  
    newDrawing.Name = name;  
    newDrawing.Insert();  
    ...  
    //Open New Drawing  
    new DrawingHandler().SetActiveDrawing(newDrawing);  
}
```

# Create View

The following example creates and inserts a new view to the active drawing. A drawing must be open when executing the code.

```
using Tekla.Structures.Drawing;
using Tekla.Structures.Geometry3d;

public class Example
{
    public void Example1()
    {
        DrawingHandler MyDrawingHandler = new DrawingHandler();
        if(MyDrawingHandler.GetConnectionStatus())
        {
            Drawing Drawing = MyDrawingHandler.GetActiveDrawing();
            CoordinateSystem CoordinateSystem = new CoordinateSystem(new Point(),
                                                                    new Vector(1, 0, 0),
                                                                    new Vector(0, 1, 0));

            View newView = new View(Drawing.GetSheet(), CoordinateSystem, CoordinateSystem,
                                    new AABB(new Point(0, 0), new Point(1000, 1000, 1000)));

            newView.Insert();
        }
    }
}
```



# Drawing.UI Picker

## How to get a drawing object from the user

```
//Setup variables to pass to drawing methods
```

```
Tekla.Structures.Drawing.ViewBase pickedViewBase;
```

```
Tekla.Structures.Geometry3d.Point pickedPoint;
```

```
Tekla.Structures.Drawing.DrawingObject pickedObject;
```

```
...
```

```
//Get input from user
```

```
Tekla.Structures.Drawing.UI.Picker picker = dh1.GetPicker();
```

```
picker.PickPoint("Pick point", out pickedPoint, out pickedViewBase);
```

```
picker.PickObject("Pick bolt group", out pickedObject, out pickedViewBase);
```



## Example: Open a drawing in background

§ It is possible to open drawing hidden to user in order to add views or change objects in the views if needed.

```
DrawingHandler dh1 = new DrawingHandler();  
:  
//Opens visually to user  
dh1.SetActiveDrawing(myDrawing, true);  
:  
//Opens hidden to user  
dh1.SetActiveDrawing(myDrawing, false);
```

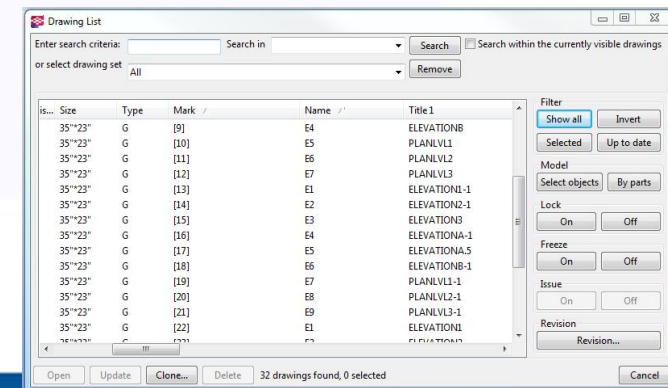
## Example: Check if a drawing is open

```
DrawingHandler dh1 = new DrawingHandler();
Drawing openDrawing = dh1.GetActiveDrawing();
if (openDrawing == null)
{
    //No drawing is open
}
else
{
    //Drawing is open
    string drawingName = openDrawing.Name;
}
```

# Example: Get certain drawing

§ The DrawingHandler is also useful for getting drawings

```
DrawingHandler dh1 = new DrawingHandler();
DrawingEnumerator drawings = dh1.GetDrawings();
while(drawings.MoveNext())
{
    if(drawings.Current.Name==desiredName)
    {
        return drawings.Current;
    }
}
```



# Get model object from drawing object

§ Drawing Parts have a method to get the model object identifier

```
//Get model objects from drawing objects
BoltGroup tBoltGroup = new Model().SelectModelObject(
    drawingBolt.ModelIdentifier)
    as BoltGroup;

//Make sure boltgroup is not null, Select
if (tBoltGroup == null) return;
tBoltGroup.Select();
```

# StraightDimensionSetHandler

```
new StraightDimensionSetHandler();
```

- § Easiest way to create dimension 'set'
- § Set is multiple dimensions grouped together
- § CreateDimensionSet(View, Points, Direction, Offset);

## Example: Create dimension to bolts

```
//Set distance past object to insert dimension line in mm
const double distancePast = 400.0;
:
//Create both horizontal and vertical dimension set of same points to object
StraightDimensionSetHandler newDimSet = new
:   Tekla.Structures.Drawing.StraightDimensionSetHandler();
:
//X direction dimension
newDimSet.CreateDimensionSet(pickedView, dimensionPoints, dimVector, distancePast);
:
//Y direction dimension
newDimSet.CreateDimensionSet(pickedView, dimensionPoints, perpVector, distancePast);
```



# Inserting Text into Drawing

```
Tekla.Structures.Drawing.Text COGText = new Tekla.Structures.Drawing.Text(  
    View1, InsertionPoint, "Test Text");  
COGText.Attributes.Font.Color = DrawingColors.Black;  
COGText.Attributes.Font.Height = 0.125 * 25.4;  
COGText.Attributes.Angle = 0;  
COGText.Insert();
```

# Inserting Symbols into Drawing

```
/// <summary> ...
```

```
private static void InsertSymbol(ViewBase thisViewBase, Point insertionPoint)
{
    var cogSybmol = new Symbol(thisViewBase, insertionPoint);
    var cogSymbolInfo = new SymbolInfo(SymbolFileName, SymbolFileNumber);
    cogSybmol.SymbolInfo = cogSymbolInfo;
    cogSybmol.Attributes.Color = DrawingColors.Black;
    cogSybmol.Attributes.Height = 0.125 * 25.4;
    cogSybmol.Attributes.Angle = 0;
    cogSybmol.Insert();
    cogSybmol.SetUserProperty("CREATED_BY", "STARTPOINT_CHECKER");
}
```



# Create new Blank GA drawing

```
var myDrawing = new GADrawing("standard",  
    new TSD.Size(36.0 * 25.4, 24.0 * 25.4));  
:  
:  
myDrawing.Name = "User Meeting 2012";  
myDrawing.Title1 = "Drawing API Session";  
myDrawing.Insert();  
:  
// This opens that drawing and shows on the screen  
_drawinghandler.SetActiveDrawing(myDrawing, true);
```



# Marks



- § Generic mark base
- § 3 Types of placement
- § Associative Note UI shown to user
  
- § Getting existing marks
- § Getting drawing part from mark

# Insert Mark into Drawing

```
//Create new mark
var beamMark = new Tekla.Structures.Drawing.Mark(dwgPart)
{
    Placing = new PointPlacing(),
    InsertionPoint = partCenterLocal
};
beamMark.Attributes.LoadAttributes(DrawingAttributeSaveAsDefaultName);
beamMark.Attributes.PlacingAttributes.IsFixed = true;

:
beamMark.MoveObjectRelative(new Vector(10.0, 0, 0));
:
```

# Get Information from Selected

```
public static void GetPartInfoForDwgObj()  
{  
    var selected = Handler.GetDrawingObjectSelector().GetSelected();  
    while (selected.MoveNext())  
    {  
        var dwgPart = selected.Current as Part;  
        if (dwgPart == null) continue;  
        var mdlPart = new Model().SelectModelObject(dwgPart.ModelIdentifier) as Tekla.Structures.Model.Part;  
        if (mdlPart == null) continue;  
        Trace.Write(string.Format("PartName: {0}", mdlPart.Name) + Environment.NewLine);  
        Trace.Write(string.Format("PartMark: {0}", mdlPart.GetPartMark()) + Environment.NewLine);  
        Trace.Write(string.Format("PartId: {0}", mdlPart.Identifier) + Environment.NewLine);  
    }  
}
```

# Insert Drawing Plug-in

```
private static void TestInsertPlugin()
{
    var allViews = ActiveDrawing.GetSheet().GetAllViews();
    while (allViews.MoveNext())
    {
        var tView = allViews.Current as View;
        if (tView == null) continue;
        var parts = tView.GetAllObjects(new[] {typeof (Tekla.Structures.Drawing.Part)});
        while (parts.MoveNext())
        {
            var tPart = parts.Current as Tekla.Structures.Drawing.Part;
            if (tPart == null) continue;
            var mPart = new Model().SelectModelObject(tPart.ModelIdentifier);
            if (mPart == null || !(mPart is Beam)) continue;
            var tBeam = mPart as Beam;

            //Set input
            var tranMatrix = MatrixFactory.ByCoordinateSystems(tBeam.GetCoordinateSystem(), tView.DisplayCoordinateSystem);
            var p1 = tranMatrix.Transform(tBeam.GetCoordinateSystem().Origin);
            var input = new PluginPickerInput();
            input.Add(new PickerInputPoint(tView, p1));
            input.Add(new PickerInputPoint(tView, new Point(p1.X + 250, p1.Y, p1.Z)));

            //Define and insert plugin
            var pg = new Plugin(tView, "COGDimensioning");
            pg.SetPickerInput(input);
            if (!pg.Insert()) Trace.WriteLine("Unable to insert plugin");
        }
    }
    ActiveDrawing.CommitChanges();
}
```

# Auto Drawings from File

## DrawingCreator Class

[Members](#) [See Also](#) [Send Feedback](#)

The DrawingCreator class is for handling the creation of drawings using the built-in logic of Tekla Structures.

**Namespace:** [Tekla.Structures.Drawing.Automation](#)

**Assembly:** Tekla.Structures.Drawing (in Tekla.Structures.Drawing.dll)

### Syntax

C#

```
[SerializableAttribute]  
public static class DrawingCreator
```

### Inheritance Hierarchy

[System.Object](#)

**Tekla.Structures.Drawing.Automation.DrawingCreator**

### See Also

[DrawingCreator Members](#)

[Tekla.Structures.Drawing.Automation Namespace](#)

```
public static bool CreateAutoDrawings(string fileName, Identifier partIdentifier)  
{  
    var rule = new AutoDrawingRule(fileName);  
    return DrawingCreator.CreateDrawings(rule, partIdentifier);  
}
```

# Detail View

## § New static method on View class

```
//Create settings, can not be null
var viewSettings = new View.ViewAttributes(SettingsManager.ViewSettings);
var markSettings = new DetailMark.DetailMarkAttributes(SettingsManager.DetailSettings);

//Create new detail view
View newView;
DetailMark newDetailMark;
if (!View.CreateDetailView(pickedView, point1, point2, point3, point4, viewSettings, markSettings,
                           out newView, out newDetailMark))
{
    Trace.WriteLine("Unable to create new detail view");
    return;
}

//Set save point
ActiveDrawing.CommitChanges();
```



# Section View

## § New static method on View class

```
//Create settings, can not be null
var viewSettings = new View.ViewAttributes(SettingsManager.ViewSettings);
var markSettings = new SectionMarkBase.SectionMarkAttributes(SettingsManager.SectionSettings);

//Calculate some values from input
var distanceBefore = Distance.PointToLineSegment(pickedPoints[2], new LineSegment(pickedPoints[0], pickedPoints[1]));
if (distanceBefore < 0) distanceBefore = ExtremaPast;
var distanceAfter = Distance.PointToLineSegment(pickedPoints[3], new LineSegment(pickedPoints[0], pickedPoints[1]));
if (distanceAfter < 0) distanceAfter = ExtremaPast;

//Create new section view
View newView;
SectionMark newSectionMark;
if (!View.CreateSectionView(pickedView, pickedPoints[0], pickedPoints[1], insertPoint, distanceBefore,
                           distanceAfter, viewSettings, markSettings, out newView, out newSectionMark))
{
    Trace.WriteLine("Unable to create new section view");
    return;
}

//Set save point
ActiveDrawing.CommitChanges();
```



# Curved Section View

## § New static method on View class

```
private static void CreateCurvedSectionBase(View pickedView, Point startPt, Point middlePt, Point endPt, Point insertPt)
{
    //Create settings, can not be null
    var viewSettings = new View.ViewAttributes(SettingsManager.ViewSettings);
    var markSettings = new SectionMarkBase.SectionMarkAttributes(SettingsManager.SectionSettings);

    //Create new section view
    const int extremaOffset = 500;
    View newView;
    CurvedSectionMark newSectionMark;
    if (!View.CreateCurvedSectionView(pickedView, startPt, middlePt, endPt, insertPt, extremaOffset, extremaOffset, viewSettings,
                                     markSettings, out newView, out newSectionMark))
    {
        Trace.WriteLine("Unable to create new section view");
        return;
    }

    //Set save point
    ActiveDrawing.CommitChanges();
}
```

# Detail Mark Symbol

```
//Set some attributes
var markAttributes = new DetailMark.DetailMarkAttributes(SettingsManager.DetailSettings);
var blank = new ViewMarkBasicTagAttributes();
var a1 = new ViewMarkBasicTagAttributes
{
    TagContent = new ContainerElement
    {
        new PropertyElement(PropertyElement.PropertyType.SectionMarkPropertyElementTypes.SectionName())
    },
    Location = TagLocation.AboveSymbolCenterLine
};

var a2 = new ViewMarkBasicTagAttributes
{
    TagContent = new ContainerElement { new TextElement("test") },
    Location = TagLocation.BelowSymbolCenterLine
};
markAttributes.TagsAttributes = new DetailMarkTagsAttributes(a1, a2, blank, blank, blank);
markAttributes.BoundaryShape = DetailMark.DetailMarkAttributes.DetailBoundaryShape.Rectangular;

//Insert and tag
var detailMark = new DetailMark(pickedView, point1, point2, new Point(point2.X + 200, point2.Y + 200, point2.Z),
                                markAttributes);

if (!detailMark.Insert())
{
    Trace.WriteLine(string.Format("Unable to insert detail mark {0}", detailMark));
    return;
}

detailMark.SetUserProperty(UdaName, UdaValue);
```

# Section Mark Symbol

```
//Set some attributes
var markAttributes = new SectionMarkBase.SectionMarkAttributes(SettingsManager.SectionSettings);
var blank = new SectionMarkBase.SectionMarkTagAttributes();
var a1 = new SectionMarkBase.SectionMarkTagAttributes
{
    TagContent = new ContainerElement
    {
        new PropertyElement(PropertyElement.PropertyType.SectionMarkPropertyElementTypes.SectionName())
    },
    Location = TagLocation.AboveSymbolCenterLine
};
var a2 = new SectionMarkBase.SectionMarkTagAttributes
{
    TagContent = new ContainerElement { new TextElement("test") },
    Location = TagLocation.BelowSymbolCenterLine
};
markAttributes.TagsAttributes = new SectionMarkBase.SectionMarkTagsAttributes(a1, a2, blank, blank, blank);

//Insert and tag
var detailMark = new SectionMark(pickedView, point1, point2, markAttributes);
if (!detailMark.Insert())
{
    Trace.WriteLine(string.Format("Unable to insert section mark {0}", detailMark));
    return;
}
detailMark.SetUserProperty(UdaName, UdaValue);

//Set save point
ActiveDrawing.CommitChanges();
```

# Get Marks from Drawing Part

## § GetRelatedObjects()

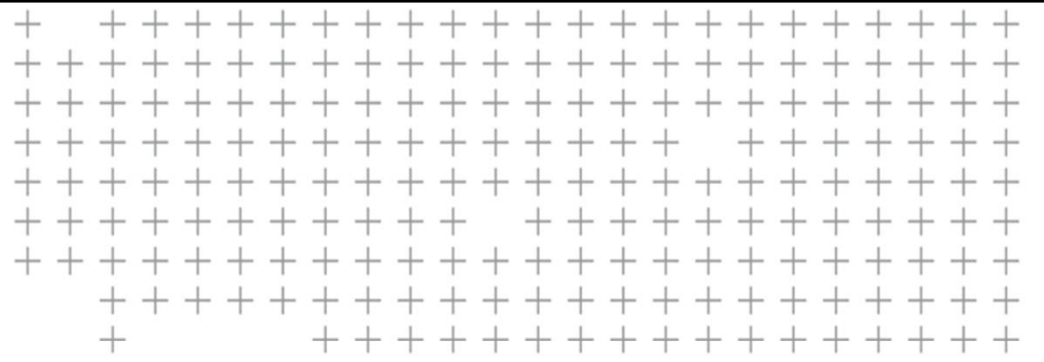
```
var existingMarks = DrawingPart.GetRelatedObjects(new[] { typeof(Mark) });
foreach (Mark mk in existingMarks)
{
    if (GetRelativePositionTag(mk) != RelativePosition) continue;
    return false;
}
```

# Drawing Non-Dependent Plugins

- § Part of drawing plugin framework
- § Plugin Definition Class Setting
- § The plug-in is never updated
- § The Created objects do not have any relation to the plug-in anymore
- § The plug-in dialog cannot be opened from the created objects.

```
[Plugin("My Drawing Plugin")]  
[PluginUserInterface("MyDrawingPlugin.View.MainForm")]  
[UpdateMode(UpdateMode.CREATE_ONLY)]  
1 reference | 0 changes | 0 authors, 0 changes  
public class MyDrawingPlugin : DrawingPluginBase  
{  
    3 references | 0 changes | 0 authors, 0 changes  
    private MyPluginData Data { get; set; }  
  
    0 references | 0 changes | 0 authors, 0 changes  
    public MyDrawingPlugin(MyPluginData data)  
    {  
        Data = data;  
    }  
}
```





# Plan Level Mark Example

# Plan Level Mark Example

- § Drawing plug-in
- § Dependent, updates with drawing
- § Creates 'Mark' objects
- § Merges content with associative note settings

# Main Definition

```
[Plugin("Plan Level Mark")]
[PluginUserInterface("PlanLevelMark.View.MainForm")]
[UpdateMode(UpdateMode.DRAWING_OPENED)]
public class PlanLevelMark : DrawingPluginBase
{
    private PlanLevelMarkData Data { get; set; }

    public PlanLevelMark(PlanLevelMarkData data) ...

    public override List<InputDefinition> DefineInput() ...

    public override bool Run(List<InputDefinition> inputs) ...

    public PlanLevelMark(PlanLevelMarkData data)
    {
        Data = data;
    }
}
```



# Define Input

```
...  
public override List<InputDefinition> DefineInput()  
{  
    var inputs = new List<InputDefinition>();  
    var drawingHandler = new TSD.DrawingHandler();  
    var picker = drawingHandler.GetPicker();  
    ...  
}
```

```
TSD.ViewBase view = null;  
TSD.DrawingObject pickedobject = null;  
Point firstpoint = null;  
Point secondPoint = null;  
...  
try  
{  
    TsLocalization.Initialize(GlobalConstants.PluginName);  
    picker.PickObject(GlobalConstants.GetLocalized(GlobalConstants.PickObjPrompt), out pickedobject, out view);  
    picker.PickPoint(GlobalConstants.GetLocalized(GlobalConstants.PickLeaderLineLocation), out firstpoint, out view);  
    picker.PickPoint(GlobalConstants.GetLocalized(GlobalConstants.PickTextLocation), out secondPoint, out view);  
}  
catch (TSD.PickerInterruptedException)  
{  
    //expected  
}
```

# Run method

```
public override bool Run(List<InputDefinition> inputs)
{
    //Get inputs from plugin
    var view = InputDefinitionFactory.GetView(inputs[0]);
    var pickedPart = (TSD.ModelObject)InputDefinitionFactory.GetDrawingObject(inputs[0]);
    pickedPart.Select();
    var firstPoint = InputDefinitionFactory.GetPoint(inputs[1]);
    var secondPoint = InputDefinitionFactory.GetPoint(inputs[2]);
    Data.LoadDefaults(this);
    ....
    //Call logic to create new mark for rebar
    return new PlanLevelMarkLogic().CreateMark(pickedPart, view, secondPoint, firstPoint, Data);
}
```

# Plug-in Data

```
public class PlanLevelMarkData
{
    [StructuresField("MarkFormatString")]
    public string MarkFormatString;

    [StructuresField("MarkSettings")]
    public string MarkSettings;

    [StructuresField("LengthUnitType")]
    public int LengthUnitType;

    [StructuresField("LengthFormat")]
    public string LengthFormat;

    [StructuresField("GlobalZOffsetUda")]
    public string GlobalZOffsetUda;

    [StructuresField("UseViewDatum")]
    public int UseViewDatum;

    [StructuresField("AdditionalOffset")]
    public double AdditionalOffset;

    public void LoadDefaults(DrawingPluginBase plugin)...
```

# Main Logic

```
//Create new mark in memory
var mk = new Tekla.Structures.Drawing.Mark(pickedModelObject);
mk.Attributes.LoadAttributes(data.MarkSettings);
mk.InsertionPoint = secondPoint;
.....
//Get point in space to dimension
var levelString = GetElevationString(data);
.....
//Add qty to content, replace existing
var existingContent = mk.GetContent();
mk.Attributes.Content.Clear();
mk.Attributes.Content.Add(new Tekla.Structures.Drawing.TextElement(levelString));
mk.Placing = new Tekla.Structures.Drawing.LeaderLinePlacing(firstPoint);
foreach (var ce in existingContent) mk.Attributes.Content.Add(ce);
.....
//Insert new mark to drawing
if (mk.Insert())
{
    RighthJustifyMark(mk);
    return true;
}
```

# Getting Elevation Point on Solid

```
var modelPart = new Tekla.Structures.Model.Model().SelectModelObject(_pickedModelObject.ModelIdentifier)
    as Tekla.Structures.Model.Part;
var modelSolid = modelPart.GetSolid(new FormingStates(DeformingType.DEFORMED));
...
//Get 'line' to intersect with solid
var dz = GetHeight(modelPart) * 25.0;
if (dz < 500) dz = 5000;
var p1 = new Point(_firstPoint);
var p2 = new Point(_firstPoint);
p1.Translate(0, 0, -dz);
p2.Translate(0, 0, dz);
...
//Get topmost solid intersection
var intersections = modelSolid.Intersect(p1, p2);
if (intersections == null || intersections.Count < 1) return "No intersections found";
var maxPt = intersections.Cast<Point>().OrderBy(f => f.Z).Last();
...
//Transform to global
var transMatrix = new Tekla.Structures.Model.Model().GetWorkPlaneHandler().
    GetCurrentTransformationPlane().TransformationMatrixToGlobal;
maxPt = transMatrix.Transform(maxPt);
```

# Getting Elevation Point on Solid

```
//Add global offset and format string
var tempValue = 0.0;
new Tekla.Structures.Model.Model().GetProjectInfo().GetUserProperty(data.GlobalZOffsetUda, ref tempValue);
if (Math.Abs(tempValue - TsModel.NullDoubleValue) < GeometryConstants.DISTANCE_EPSILON) tempValue = 0;
if (data.UseViewDatum == 1 && _viewBase is Tekla.Structures.Drawing.View) tempValue =
... ((Tekla.Structures.Drawing.View)_viewBase).Attributes.DatumLevel;
return GetFormattedElevation(maxPt.Z + tempValue + data.AdditionalOffset, data, modelPart);
```

# Getting Mark Existing Content

```
public static class TxMark
{
    public static List<ElementBase> GetContent(this Mark mk)
    {
        if (mk == null) throw new ArgumentNullException("mk");
        var result = new List<ElementBase>();
        var contentEnum = mk.Attributes.Content.GetEnumerator();
        while (contentEnum.MoveNext()) result.Add(contentEnum.Current as ElementBase);
        return result;
    }
}

private static void RighthJustifyMark(Tekla.Structures.Drawing.Mark mk)
{
    var width = mk.Attributes.Frame.GetAxisAlignedBoundingBox().Width;
    mk.MoveObjectRelative(new Vector(width / 2, 0, 0));
    mk.Modify();
}
```

# Get Elevation in Right Units, Text Replace

```
private static string GetFormattedElevation(double maxPt, PlanLevelMarkData data, Tekla.Structures.Model.Part pickedPart)
{
    if (data == null) throw new ArgumentNullException("data");
    var symSetup = new QuantityUnitSymbols { SeparatorBetweenFeetAndInches = "-"};
    var lengthString = string.Empty;
    if ((LengthUnit)data.LengthUnitType == LengthUnit.Millimeter)
        lengthString = new Length(maxPt).ToString((LengthUnit)data.LengthUnitType, data.LengthFormat);
    else
        lengthString = new Length(maxPt).ToString((LengthUnit)data.LengthUnitType, data.LengthFormat, null, symSetup);
    var str1 = data.MarkFormatString.Replace("ELEVATION", lengthString);
    var str2 = str1.Replace("PART_MARK", pickedPart.GetPartMark());
    return str2.Replace("ASSEMBLY_MARK", pickedPart.GetAssemblyMark());
}
```

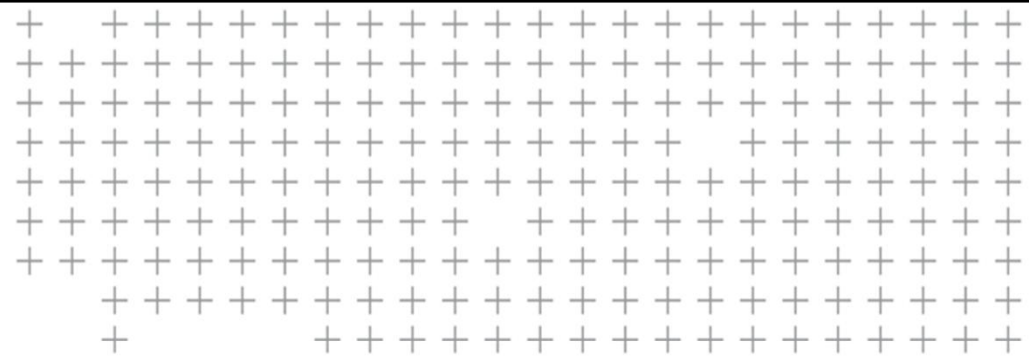


# User Interface

The screenshot shows a software dialog box titled "albl\_Plan\_Level\_Mark". At the top, there are buttons for "albl\_Save", "albl\_Load", a dropdown menu, "albl\_Save\_As", and "albl\_Help\_\_". Below these is a tab labeled "albl\_Parameters". The main area contains a list of settings, each with a checked checkbox and a corresponding input field:

Parameter	Value
<input checked="" type="checkbox"/> albl_Mark_Settings	[Dropdown menu]
<input checked="" type="checkbox"/> albl_Mark_Format_String	[Text field]
<input checked="" type="checkbox"/> albl_Unit_Type	[Dropdown menu]
<input checked="" type="checkbox"/> albl_Unit_Precision	[Dropdown menu]
<input checked="" type="checkbox"/> albl_Global_Z_Offset_Uda_Na	[Text field]
<input checked="" type="checkbox"/> albl_Datum_Offset	[Dropdown menu]
<input checked="" type="checkbox"/> albl_Additional_Offset	[Text field]

At the bottom of the dialog are buttons for "albl\_OK", "albl\_Apply", "albl\_Modify", "albl\_Get", a checkbox with a slash symbol, and "albl\_Cancel".



 Thank You