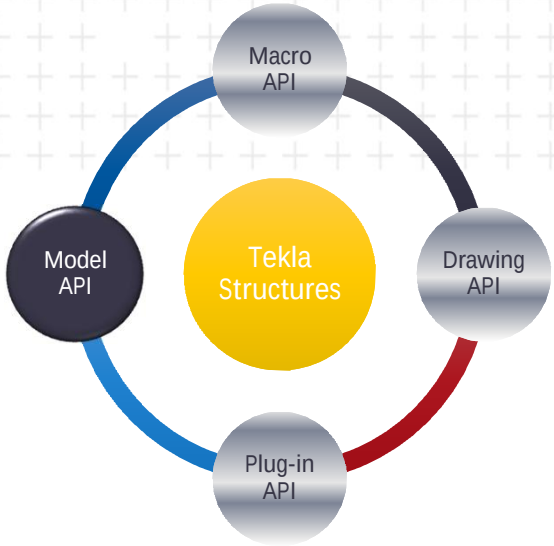
Tekla Open API: Modeling Tools & Plug-Ins

# Model API

# Tekla Open API - Model




- Model API
  - Connect to a running Tekla Structures model
  - Create, modify, and delete model objects
    - Read and write object attributes
    - Read and write user defined attributes
    - Get report properties for objects
  - Interact with the user
    - Get currently selected objects
    - Prompt user to pick objects and locations
    - Select and highlight objects for the user
  - Access catalogs (material, bolt, profile, etc.)
  - Create and manipulate model views

# Structure of the Model API

## Model

Model Info, GetObjects, Project Info, Workplane, GetPhases, CommitChanges

## Grouped Objects

Phases, Hierarchic Objects, Tasks

## Model Objects

Assemblies, Parts, Bolts, Rebar, Welds, Grids, Reference Models and Objects, Surfacing, Cuts, Loads

Connections, Details, Seams, Custom Parts, Plugins

## Operations

Run Macros, Move/Copy, Split, Combine, CNC, MIS, Web Model, Report, Check Numbering, Status Message

## UI (User Interface)

Picker, Highlighter, GetSelected, Views, Clipping Planes, Object Rep, Graphics Drawer (text, lines)

## Geometry

Planes, Intersections, Points, Coordinate Systems, Matrix

## Solid

Part Faces, Edges

## History

## Events

## Clash Checking

# Tekla.Structures.Geometry3D

- Geometric calculations and tests
  - Distance
    - Point to Point, Point to Line, Point to Plane
  - Parallel
    - Line to Line, Line to Plane, Plane to Plane
- Geometric constructions
  - Intersection
    - Line to Line, Line to Plane
  - Projection
    - Point to Line, Point to Plane, Line to Plane
- Matrix
    - Transformations from one coordinate system to another
    - MatrixFactory for easy creation of matrixes

# API Use Types

# Types of API Projects

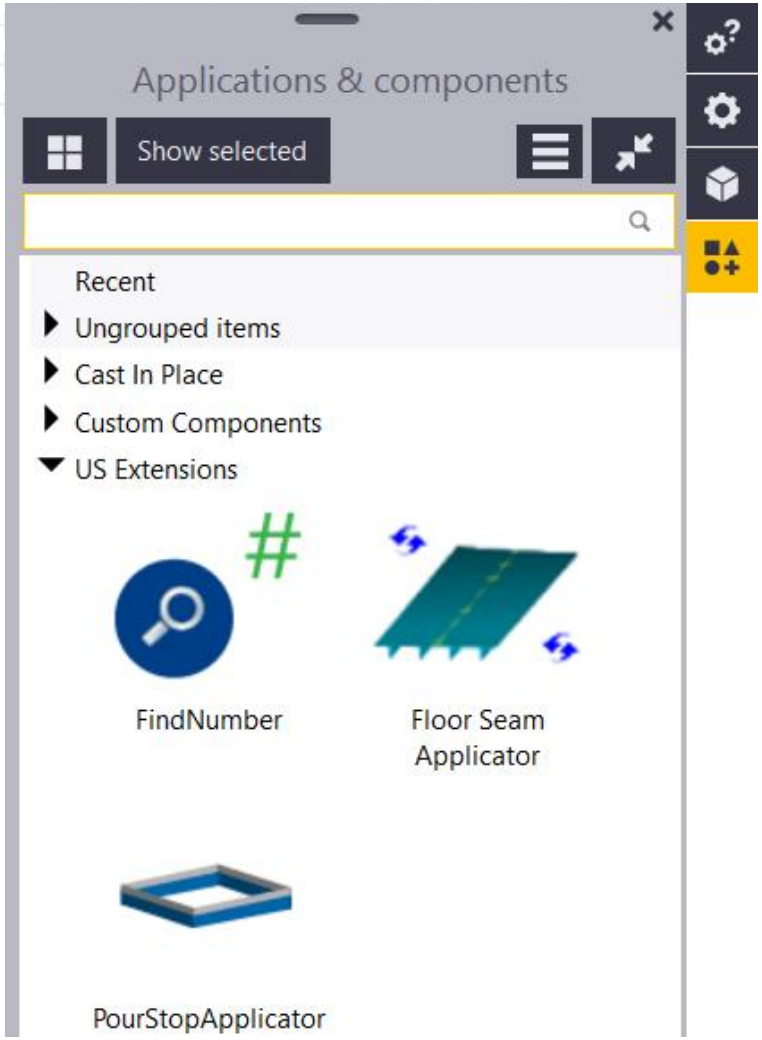
- Macros
  - Recorded scripts
  - Version independent
- Plug-ins
  - Internal to Tekla Structures
  - Update automatically to changes
- Applications
  - External to Tekla Structures
  - Increased flexibility

# Characteristics of Tekla plugins

- A custom entity in the model (component)
- Has a dialog of its own
- Can create new model objects (even other plugins)
- Can be a connection, detail or generic component

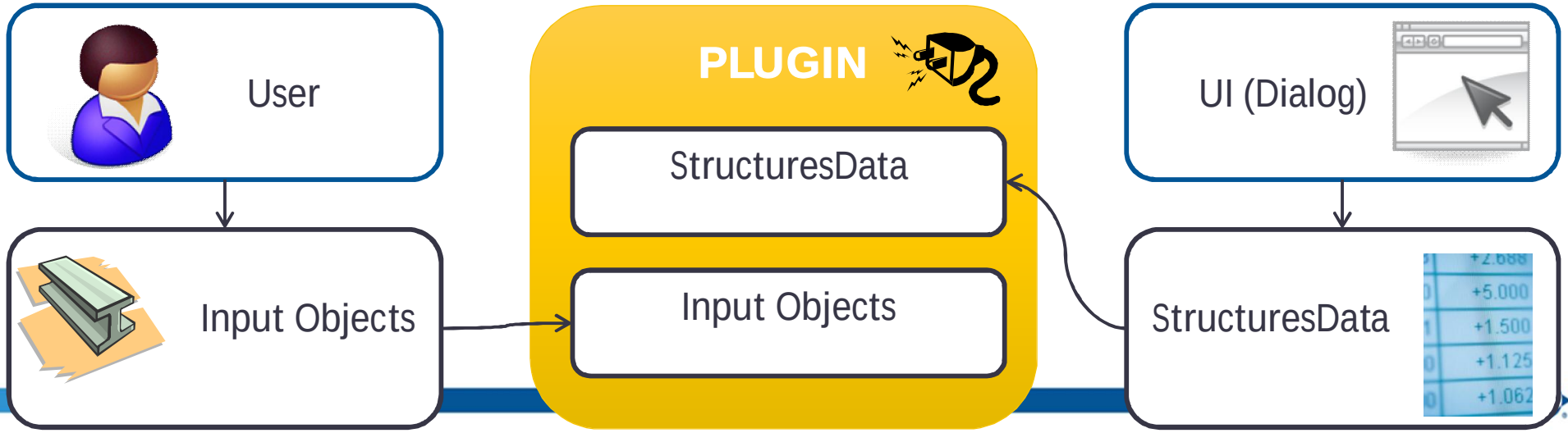
# Accessing plugins

- Component Catalog
- Installed
- Downloaded



# Inserting new plugin into model

- New Plugin is started
  - Constructor method runs
  - Input prompted from user
    - Applied values taken from dialog
    - Plugin Run() when input complete
  - Both the StructuresData and the Input are stored to the Tekla model database



# Plug-in dependency

- Plug-ins cannot modify their inputs.
- Plug-ins dependency can be set with the attribute InputObjectDependency.
  - Dependent: updated when input changes.
  - Non-Dependent: doesn’t update when input changes.
  - Geometrically-Dependent: Plug-in updates when the input part geometry changes. This Plug-in cannot create any boolean objects to the input part, since it would cause and endless loop.
  - Non-Dependant-Modifiable: No dependency on input but the instance is modifiable in the model. The created objects have a relation to the plug-in. The plug-in dialog can be opened from the created objects.

# Plugins & Applications

## Plugins

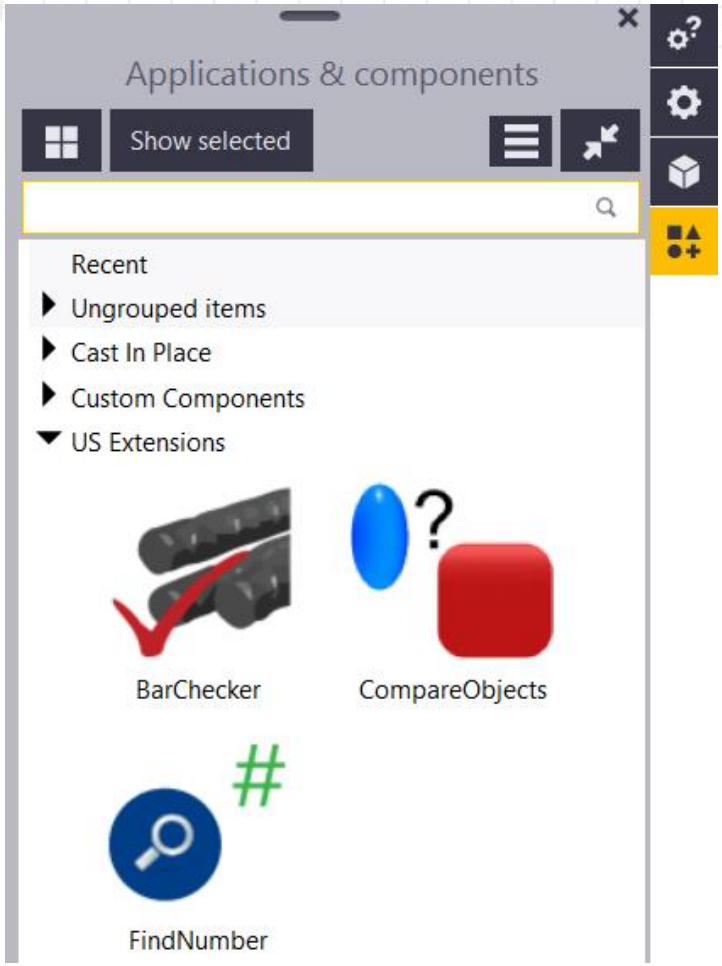
- Great alternative to custom components
- Live in model

## Applications

- Great for doing work in the model
- One shot and done

# Accessing Applications

- Launch Macros
- Direct Launch



# Plugins, Custom Components, & Applications

| | Custom Components | Plugins | Applications |
|---|---|---|---|
| Own 'object' | Yes | Yes | No |
| Auto Updates | Yes | Yes | No |
| Integrated Dialog | Yes | Yes | No |
| Performance | Slower | Faster | Moderate |
| Memory | Heavier | Lighter | n/a |
| Flexibility | Lower | Higher | Moderate |
| Difficulty | Lower | Moderate | Moderate |

# Model API Tips

# Things to Know

- GetCurrentUser()
- OBB Class

| | |
|---|---|
| IntersectionPointsWith(Line) | Calculates the intersection points between OBB and given Line. |
| IntersectionPointsWith(LineSegment) | Calculates the intersection points between OBB and given LineSegment. |
| IntersectionWith(Line) | Creates an intersection between the OBB and the given Line. |
| IntersectionWith(LineSegment) | Creates an intersection between the OBB and the given LineSegment. |
| Intersects(GeometricPlane) | Tests if current OBB intersects with the given GeometricPlane |
| Intersects(Line) | Tests if current OBB intersects with the given Line |
| Intersects(Li | |
| Intersects(O | |
| ClosestPointTo(Line) | Calculates the closest point in OBB to given Line. |
| ClosestPointTo(LineSegment) | Calculates the closest point in OBB to given LineSegment. |
| ClosestPointTo(Point) | Calculates the closest point in OBB to given point. |
| ComputeVertices | Calculates the corner points of the OBB. |
| DistanceTo(Line) | Calculates the distance from OBB to given Line. |
| DistanceTo(LineSegment) | Calculates the distance from OBB to given LineSegment. |
| DistanceTo(Point) | Calculates the distance from OBB to given point. |

# Things to Know

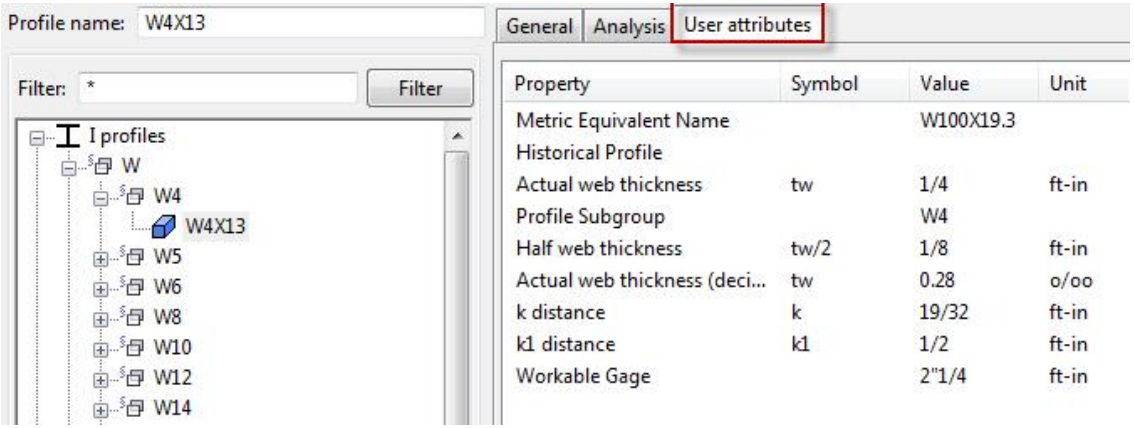
- Solid.IntersectAllFaces() allows use of high-accuracy solids.
  - Part.GetSolid(),
  - Solid.Intersect(LineSegment),
  - Solid.Intersect(Point, Point).
- Rebar geometries fetched in deformed form by default.

# Things to Know: Catalogs

- Rebar catalog: Item export and import methods
- Mesh catalog: Item export and import methods
- Shape catalog: Export() method in ShapeItem class
- ImportShapes() method in CatalogHandler class
- Import from folder for:
  - LibraryProfileItems,
  - ParametricProfileItems,
  - MaterialItems,
  - Custom components.

# Getting profile item and user attributes

- items = new CatalogHandler().GetLibraryProfileItems();
- aProfileItemUserParameters give list of user attributes
- aProfileItemParameters
- aProfileItemAnalysisParameters
- Type, name, Sketch & sketch info
- Export



# Solid Class

- Contains GetAllIntersectionPoints method that gets all the intersection points between the solid and a plane. Compared to the IntersectAllFaces method, it does not arrange the points into polygons, and is thus a lot faster.
- Solid.SolidCreationTypeEnum.NORMAL_WITHOUT_WELDPREPS has been added to the part.GetSolid() method.

# ModelHistory Class

- Works with Tekla Model Sharing
- The following methods have been added:
  - GetNotSharedObjects() - returns a list of objects created or modified since the last Sharing WriteOut or multi-user save.
  - TakeModifications() - returns the modifications since the previous call and resets the modification stamp.
  - GetModifications() - returns the modifications since the previous call of
  - TakeModifications() without resetting the modification stamp.
- The following methods are obsolete:
  - GetModifiedObjects(), GetModifiedObjectsWithType(),
  - GetDeletedObjects(), GetDeletedObjectsWithType(),
  - GetCurrentModificationStamp()

# AutoFetch Property

- New static AutoFetch property in ModelObjectEnumerator and DrawingObjectEnumerator which increases the enumeration speed significantly.

```
ModelObjectEnumerator.AutoFetch = true;
ModelObjectEnumerator myEnum = myModel.GetModelObjectSelector().GetAllObjects();
ArrayList allObjects = new ArrayList();
while (myEnum.MoveNext())
{
    try
    {
        allObjects.Add(myEnum.Current);
    }
    catch { }
}
```

# Bolt Catalog Items

## Constructors

| | Name | Description |
|---|---|---|
| | BoltItem | Creates a new bolt item instance. |

## Methods

| | Name | Description |
|---|---|---|
| | ExportBoltStandard | Exports the bolt item standard + nee given bolt standard + items are exp |

## Properties

| | Name | Description |
|---|---|---|
| | Size | The bolt item's size. |
| | Standard | The bolt item's grade. |
| | Type | The bolt item's type. |

# Profile Catalog Items

## ProfileItem Members

ProfileItem Class Methods Properties See Also Send Feedback

**[This is preliminary documentation and is subject to change.]**

The ProfileItem type exposes the following members.

### Methods

| | Name | Description |
|---|---|---|
| | Export | Exports the profile item in the profile database to given file name. Currently profiles are supported. Library profiles are exported to *.lis format. Sketch p profiles are exported to *.clb format. If path is not given profile is exported name or prefix is used as filename. |
| | IsProfileUserDefined | Whether the profile is a fixed user-defined profile. |
| | IsProfileUserParametric | Whether the profile is a parametric user-defined profile. If so, the prefix can |
| | Select | Selects the profile item in the profile database. |

### Properties

| | Name | Description |
|---|---|---|
| | aProfileItemParameters | An array list with the profile item parameters. |
| | IsMultiCrossSectionUserParametric | Whether the profile is a parametric user-defined multi cross section profile. |
| | IsSketchedUserParametric | Whether the profile is a parametric user-defined sketched profile. |
| | NumberOfCrossSections | The number of cross sections in the profile item. |
| | ParameterString | The profile item parameter string. |
| | ProfileItemSubType | The profile item subtype. |
| | ProfileItemType | The profile item type. |

# Component Catalog Items

## Constructors

| | Name | Description |
|---|---|---|
| | ComponentItem | Creates a new component item instance |

## Methods

| | Name | Description |
|---|---|---|
| | Export | Exports the custom component item in * exported to model folder. If filename is |
| | GetVersion | Gets the version number of custom com |
| | Select | Selects the component item from the co |

## Properties

| | Name | Description |
|---|---|---|
| | Name | The component item's internal name wh |
| | Number | The component item's internal number |
| | Type | The component item's type. |
| | UIName | The component item's name which is vi |

# Frequently Asked Questions

# Construction Circle

```
using Tekla.Structures.Model;
using Tekla.Structures.Geometry3d;

public class Example
{
        public void Example1()
        {
            Point p1 = new Point(4500, 4500, 0);
            Point p2 = new Point(4500, 9000, 0);
            Point p3 = new Point(0, 0, 0);
            ControlCircle controlCircle = new ControlCirle(p1, p2, p3);

            bool Result = false;
            Result = ControlCircle.Insert();
        }
}
```

# Construction Point

```
using Tekla.Structures.Model;

public class Example
{
        public void Example1()
        {
            Point point = new Point(6000,6000,0);
            ControlPoint controlpoint = new ControlPoint(point);

            bool Result = false;
            Result = controlpoint.Insert();
        }
}
```

# Part Cut

```
public static void RemovePart(this Beam existingBeam, Part partToCutWith)
{
    //Create part cut object from temporary beam
    partToCutWith.Class = BooleanPart.BooleanOperativeClassName;
    var cutObject = new BooleanPart {Father = existingBeam};
    cutObject.SetOperativePart(partToCutWith);
    cutObject.Type = BooleanPart.BooleanTypeEnum.BOOLEAN_CUT;
    if (!cutObject.Insert())
    {
        Trace.WriteLine("Unable to insert cut");
        return;
    }

    //Delete temporary part
    partToCutWith.Delete();
    new Model().CommitChanges();
}
```

# Part Add

```
public static void AddPart(this Beam existingBeam, Part partToAdd)
{
    //Create part cut object from temporary beam
    partToAdd.Class = BooleanPart.BooleanOperativeClassName;
    var cutObject = new BooleanPart { Father = existingBeam };
    cutObject.SetOperativePart(partToAdd);
    cutObject.Type = BooleanPart.BooleanTypeEnum.BOOLEAN_ADD;
    if (!cutObject.Insert())
    {
        Trace.WriteLine("Unable to add part");
        return;
    }

    //Delete temporary part
    partToAdd.Delete();
    new Model().CommitChanges();
}
```

# Unit Conversion

- Tekla.Structures.Datatype.dll
- Form control binding

| Tekla Structures | |
|---|---|
| AttributeName | **SpacingList** |
| AttributeTypeName | **DistanceList** |
| BindPropertyName | **Text** |
| IsFilter | False |

```
public static List<Distance> GetDistancesFromString(string combinedStringList)
{
    var distances = DistanceList.Parse(combinedStringList, null, Distance.UnitType.Millimeter);
    return distances.ToList();
}
```

# Unit Conversion

- Converting single strings to distance objects

```
Distance.CurrentUnitType = Distance.UnitType.Millimeter;
return Distance.Parse(singleDistance, CultureInfo.InvariantCulture);
```

```
return Distance.Parse(singleDistance, CultureInfo.InvariantCulture,
    Distance.UnitType.Millimeter);
```

# Finding Connected objects

```
public static List<Part> GetConnectedObjects(this Part fatherPart)
{
    if (fatherPart == null) return null;
    var connectedParts = new List<Part>();

    //Enumerate child components
    var children = fatherPart.GetComponents();
    while (children.MoveNext())
    {
        //Filter out unwanted types
        if (!(children.Current is Connection)) continue;
        var connection = children.Current as Connection;

        //Get primary and secondaries
        var primary = connection.GetPrimaryObject() as Part;
        if (primary != null) connectedParts.Add(primary);
        var secondaries = connection.GetSecondaryObjects();
        connectedParts.AddRange(secondaries.OfType<Part>().Select(secondary => secondary));
    }
    return connectedParts;
}
```

# Creating Assemblies

§ Assembly / CastUnit

```
//Add plate to assembly
if (data.CreateAssembly && _createdContourPlates.Any())
{
    Assembly masterAssembly = null;
    foreach (var plate in _createdContourPlates)
    {
        if (masterAssembly == null) masterAssembly = plate.GetAssembly();
        else masterAssembly.Add(plate);
    }
    if (masterAssembly != null) masterAssembly.Modify();
}
```

# Traversing Assemblies

```
//Get objects sorted by x direction of main part/assembly
var secondaryParts = mainAssembly.GetSecondaries();
secondaryParts.Add(mainAssembly.GetMainPart())
var moObjects = secondaryParts.Cast<ModelObjec
var sortedObjects = (from modelObject in moObj
                     orderby modelObject.GetCo
                     select modelObject).Last(
```

### What is LINQ?

stackoverflow

From MSDN:

> The LINQ Project is a codename for a set of extensions to the .NET Framework that encompass language-integrated query, set, and transform operations. It extends C# and Visual Basic with native language syntax for queries and provides class libraries to take advantage of these capabilities.

What this means is that LINQ provides a standard way to query a variety of datasources using a common syntax.

### What flavours of LINQ are there?

Currently there are a few different LINQ providers provided by Microsoft:

- Linq to Objects which allows you to execute queries on any IEnumerable object.
- Linq to SQL which allows you to execute queries against a database in an object oriented manner.
- Linq to XML which allows you to query, load, validate, serialize and manipulate XML documents.
- Linq to Entities as suggested by Andrei
- Linq to Dataset

There are quite a few others, many of which are listed here.

### What are the benefits?

- Standardized way to query multiple datasources
- Compile time safety of queries
- Optimized way to perform set based operations on in memory objects
- Ability to debug queries

# Rotating Objects

- MatrixFactory.Rotate(double value, Vector direction)

```
public static CoordinateSystem GetRotatedCoordinateSystem(this CoordinateSystem inputCs,
                                                          double xValue, double yValue, double zValue)
    {
        //Get rotation from current based on input
        var rotationX = MatrixFactory.Rotate(xValue * Math.PI / 180, inputCs.AxisX);
        var rotationY = MatrixFactory.Rotate(yValue * Math.PI / 180, inputCs.AxisY);
        var rotationZ = MatrixFactory.Rotate(zValue * Math.PI / 180, Vector.Cross(inputCs.AxisX, inputCs.AxisY));
        var rotation3D = rotationX * rotationY * rotationZ;

        //Get rotated system from original
        return new CoordinateSystem
        {
            Origin = inputCs.Origin,
            AxisX = new Vector(rotation3D.Transform(new Point(inputCs.AxisX))),
            AxisY = new Vector(rotation3D.Transform(new Point(inputCs.AxisY)))
        };
    }
}
```

# Getting Desired Part Faces

§ Contour plate, local top face

```
public static List<Face> GetPartTopFaces(this Part part)
{
    if (part == null) return null;
    var cs = part.GetCoordinateSystem();
    var localUpDir = Vector.Cross(cs.AxisX, cs.AxisY);

    var faceEnum = part.GetSolid().GetFaceEnumerator();
    var candidates = new List<Face>();
    while (faceEnum.MoveNext())
    {
        var face = faceEnum.Current as Face;
        if (face == null) continue;
        if (face.Normal.IsSameDirection(localUpDir)) candidates.Add(face);
    }
    if (candidates.Count < 1) return null;

    var topPointZ = candidates.Max(f => f.GetFaceOrigin().Z);
    var orderedFaces = (from fc in candidates
                        where (Math.Abs(fc.GetFaceOrigin().Z - topPointZ) < GeometryConstants.DISTANCE_EPSILON)
                        select fc).ToList();
    return orderedFaces;
}
```

# Insert LayoutPoint plug-in

- LayoutPoint is a plug-in that comes with Tekla Structures installation.
- You can insert new instances of any plug-in through API

```
public static void InsertLayoutPoint(this Point pt, string label = null)
{
    var ci = new ComponentInput();
    ci.AddOneInputPosition(pt);
    var cp = new Component(ci) { Number = -100000, Name = "LayoutPointPlugin" };
    cp.SetAttribute("PointLabel", label);
    if (!cp.Insert()) Trace.WriteLine("Unable to insert layout point");
}
```

# Getting Component Children

- You can get objects created by components in the model

```
public static void GrabWeldInfo()
{
    var counter = 0;
    var fieldWelds = new List<BaseWeld>();
    var selectedObjects = new ModelObjectSelector().GetSelectedObjects();
    while (selectedObjects.MoveNext())
    {
        if (selectedObjects.Current is Connection)
        {
            var cnx = selectedObjects.Current as Connection;
            var children = cnx.GetChildren();
            while (children.MoveNext())
            {
                var weld = children.Current as BaseWeld;
                if (weld == null || weld.ShopWeld) continue;
                fieldWelds.Add(weld);
            }
        }
    }
    Trace.WriteLine(counter + " welds modified.");
    new Model().CommitChanges();
}
```

# Part Cut

```
public static void RemovePart(this Beam existingBeam, Part partToCutWith)
{
    //Create part cut object from temporary beam
    partToCutWith.Class = BooleanPart.BooleanOperativeClassName;
    var cutObject = new BooleanPart {Father = existingBeam};
    cutObject.SetOperativePart(partToCutWith);
    cutObject.Type = BooleanPart.BooleanTypeEnum.BOOLEAN_CUT;
    if (!cutObject.Insert())
    {
        Trace.WriteLine("Unable to insert cut");
        return;
    }

    //Delete temporary part
    partToCutWith.Delete();
    new Model().CommitChanges();
}
```

# Part Add

```
public static void AddPart(this Beam existingBeam, Part partToAdd)
{
    //Create part cut object from temporary beam
    partToAdd.Class = BooleanPart.BooleanOperativeClassName;
    var cutObject = new BooleanPart { Father = existingBeam };
    cutObject.SetOperativePart(partToAdd);
    cutObject.Type = BooleanPart.BooleanTypeEnum.BOOLEAN_ADD;
    if (!cutObject.Insert())
    {
        Trace.WriteLine("Unable to add part");
        return;
    }

    //Delete temporary part
    partToAdd.Delete();
    new Model().CommitChanges();
}
```

# B-rep Parts Through API

- Part Item
- Insert or edit B-rep type parts
  - Insert,
  - Delete,
  - Modify,
  - SetPhase,
  - SetUserProperty,
  - normal base get
- ShapeItem & ShapeItemEnumerator classes
- Enumeration & exporting shape geometry definitions
- Create B-rep objects utilizing shape definitions

# B-rep Parts Through API

```
using Tekla.Structures.Model;
using Tekla.Structures.Geometry3d;

public class Example
{
        public void Example1()
        {
            Point point = new Point(0, 0, 0);
            Point point2 = new Point(1000, 0, 0);
            Brep brep = new Brep();
            brep.StartPoint = point;
            brep.EndPoint = point2;
            brep.Profile = new Profile { ProfileString = "Default" };
            bool result = brep.Insert();
        }
}
```

# Picking Faces

# Picker Class

```
using Tekla.Structures.Model.UI;
using Tekla.Structures.Geometry3d;

using System;
using System.Windows.Forms;

public class Example
{
        public void Example1()
        {
            Picker Picker = new Picker();
            Point p = null;
            try
            {
                p = Picker.PickPoint();
            }
            catch (Exception e)
            {
                MessageBox.Show(e.ToString());
            }
        }
}
```

# Picker Input

## PickInput Members

PickInput Class Methods Properties See Also Send Feedback

**[This is preliminary documentation and is subject to change.]**

The PickInput type exposes the following members.

### Methods

| | Name | Description |
|---|---|---|
| | CopyTo | Copies the elements of the ICollection t |
| | GetEnumerator | Returns an enumerator that iterates thr |

### Properties

| | Name | Description |
|---|---|---|
| | Count | Gets the number of elements contained |
| | IsSynchronized | Gets a value indicating whether access |
| | SyncRoot | Gets an object that can be used to sync |

# Getting Face Input from User

```
public static List<Point> GetPointsOnFace()
{
    PickInput pickInput = null;
    var picker = new Picker();
    try
    {
        pickInput = picker.PickFace("Pick face on part.");
    }
    catch (Exception)
    {
        //User interrupted
    }
    if (pickInput == null) return null;

    //Get results from pick input
    var myEnum = pickInput.GetEnumerator();
    while (myEnum.MoveNext())
    {
        var item = myEnum.Current as InputItem;
        if (item.GetInputType() != InputItem.InputTypeEnum.INPUT_POLYGON) continue;
        var points = item.GetData() as ArrayList;
        return points.Cast<Point>().ToList();
    }
    return null;
}
```

# Thank You