**Steel Connections COM API**

**Developer Training Guide**

**Contents**

[Chapter 1 Steel Connections Development possibilities 5](#_Toc117525956)

[1.1 Introduction to the Steel Connections API 6](#_Toc117525957)

[1.2 What can you do with the Steel Connections API? 6](#_Toc117525958)

[1.3 Requirements 6](#_Toc117525959)

[1.4 Installation 6](#_Toc117525960)

[Chapter 2 Steel Connections Modeling 7](#_Toc117525961)

[2.1 What are joints? 8](#_Toc117525962)

[2.2 Joint description 8](#_Toc117525963)

[2.3 How joints work 8](#_Toc117525964)

[Chapter 3 Steel Connections Modeling API Presentation 11](#_Toc117525965)

[3.1 Geometry API 12](#_Toc117525966)

[3.1.1 IPoint3d 12](#_Toc117525967)

[3.1.2 IVector3d 13](#_Toc117525968)

[3.1.3 ILine3d 14](#_Toc117525969)

[3.1.4 IPlane 15](#_Toc117525970)

[3.1.5 ICS3d 16](#_Toc117525971)

[3.1.6 IAugPolygon3d 17](#_Toc117525972)

[3.2 Modeling API 19](#_Toc117525973)

[3.2.1 Default attribute state for objects controlled by connections 19](#_Toc117525974)

[3.2.2 Beams 20](#_Toc117525975)

[3.2.3 Plates 26](#_Toc117525976)

[3.2.4 Features 33](#_Toc117525977)

[3.2.5 Special parts 38](#_Toc117525978)

[3.2.6 Bolts, Welds and Anchors 39](#_Toc117525979)

[Chapter 4 Joints API 45](#_Toc117525980)

[4.1 User interface 46](#_Toc117525981)

[4.2 Query method 46](#_Toc117525982)

[4.3 CreateObjects method 46](#_Toc117525983)

[4.4 InField/OutField methods 46](#_Toc117525984)

[4.5 GetUserPages method 46](#_Toc117525985)

[4.6 GetTableName method 47](#_Toc117525986)

[4.7 GetExportData method 47](#_Toc117525987)

[4.8 GetFeatureName method 47](#_Toc117525988)

[4.9 FreeUserPages method 47](#_Toc117525989)

[4.10 Libraries 47](#_Toc117525990)

[4.11 Attributes and categories 47](#_Toc117525991)

[4.12 JointTransfer 48](#_Toc117525992)

[4.13 Database Structure 48](#_Toc117525993)

[4.14 Joint table 49](#_Toc117525994)

[4.15 Developing a joint 50](#_Toc117525995)

[4.15.1 Visual Studio solution 50](#_Toc117525996)

[4.15.2 Create GUID 53](#_Toc117525997)

[4.15.3 Build the program 54](#_Toc117525998)

[4.15.4 Add records in database 54](#_Toc117525999)

[4.15.5 User interface 55](#_Toc117526000)

[4.15.6 Create first page 55](#_Toc117526001)

[4.15.7 Add an ActiveX control 56](#_Toc117526002)

[4.15.8 Control Properties 57](#_Toc117526003)

[4.15.9 Run joint 58](#_Toc117526004)

[4.15.10 Making the joint available in Revit 61](#_Toc117526005)

[Chapter 5 Joints Design API 64](#_Toc117526006)

[5.1.1 Implementation of a joint calculation module 65](#_Toc117526007)

[5.1.2 How it works 65](#_Toc117526008)

[5.1.3 Communication interface 65](#_Toc117526009)

[5.1.4 Description of interface methods 66](#_Toc117526010)

[5.1.5 Attach design module to a joint 67](#_Toc117526011)

[5.2 Developing a joint design 68](#_Toc117526012)

# Chapter 1 Steel Connections Development possibilities

## Introduction to the Steel Connections API

The Steel Connections API allows you to program with any .NET compliant language including C#, and C++. Before using the API, learn to use Steel Connections and its features so that you can better understand this API.

## What can you do with the Steel Connections API?

* Create joints

## Requirements

* Understanding Steel Connections
* Programming languages like C++, C#.
* Visual Studio Community Edition (if applicable) 2022 with “.NET Desktop development” and/or “Desktop development with C++” workloads
* Revit 2025

## Installation

The Steel Connections API is installed by Revit.

Microsoft Visual Studio Community Edition 2022 can be downloaded from <https://visualstudio.microsoft.com/downloads/>

# Chapter 2 Steel Connections Modeling

## What are joints?

Joints are a complex group of basic elements that depend on the main input element(s) and are controlled by a construction rule.

All objects created by the connection can be made visible by setting the Detail level to Fine. On the other detail levels the connection objects are not visible and instead a connection handler object appears, informing that a connectionis there.

## Joint description

A joint is created through an object which implements the *IRule* interface and uses the *IJoint* interface to create Steel Connection elements.

Joint workflow:

* Input definition (Query).
* Create objects (CreateObjects).
* Display dialog (GetUserPages).

Through a joint, the created objects depend on the selected input objects, as long as the joint object exists in the model. The created objects or their properties are modifiable at any time using the joint dialog box. Previously created objects can be updated.

The *IRule* interface *Query* method is called when a joint should be created. In this method you can select the input objects for your joint and initialize default parameters.

The *IRule* interface *CreateObjects* contains the joint functionality. It uses global variables declared in the declaration section and does the main work.

The *IRule* interface *GetUserPages* is called when the “Steel Connections” joint properties dialog is displayed.

The *IJointInfo* interface properties are interrogated to get information about the developer. You need to add a record in **AstorRules.***HRLDefinition* and **AstorRules.***RulesDllSigned* tables after developing a joint to be able to execute it in Revit.

## How joints work

*Steel Connections* creates and modifies joints by using *rules*. The *IRule* interface defines several methods/properties, each being responsible for a certain task.

The *Joint* provides the link of *IRule* with the underlying Joint object of Steel Connections. The Joint object is set by Steel Connections before calling any method of the Rule.

The *Query* method describes the input parameters of the joint. Its implementation should ask the user for input with the help of *IAstUI* and add the necessary entities to *InputObjects* of the joint.

The *CreateObjects* method creates Steel Connection objects and adds them to the *CreatedObjects* of the joint. Rules can be written by anybody. The Steel Connections framework can execute them at runtime. Thus, several services are provided - creation, modification, explode, deletion of a joint object:

* A joint object is created along with its driven objects. A dialog box appears, displaying all the parameters which apply to that joint.
* Exploding an object means deleting the “logical joint unit” while maintaining the driven objects. These objects will then live explicitly and will no longer have any link to the joint they were part of before the joint was exploded.
* Deleting an object means deleting the logical unit and the driven objects as well. This includes features on driving objects (e.g., beam notches) etc.

# Chapter 3 Steel Connections Modeling API Presentation

## Geometry API

These are the basic geometric Steel Connections objects. In any joint that will be developed you will have to use at least a few of these objects (IPoint3d, IVector3d, IPlane, etc.).

### IPoint3d

#### Properties

* X (property) - Gets or sets the x coordinate of the point.
* Y (property) - Gets or sets the y coordinate of the point.
* Z (property) - Gets or sets the z coordinate of the point.

#### Methods

* Add(IVector3d VectorToAdd) - Adds the specified vector to this point.
* Create(double dfX, double dfY, double dfZ) - Creates the point with specified coordinates.
* DistanceTo(IPoint3d target) - Returns the distance between points.
* Project(IPlane targetPlane, IVector3d projDir) - Projects the point on the specified plane, along the direction given by the vector.
* Subtract(IPoint3d Subtracter) - Returns a vector, oriented from subtracter to this point: p1.Subtract(p2) – the vector is oriented from p2 to p1.
* TransformBy(IMatrix3d TransformMatrix) - Applies the transformation specified by the matrix to this point.

#### Example: Create 2 points, calculate the distance between them and create a vector from the second point to the first.

|  |
| --- |
| private void Points()  {  //create a point with specified coordinates  IPoint3d point = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  point.Create(10, 10, 0);  //move point along Z axis with 100 mm  IVector3d vector = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  vector.Create(0, 0, 1);  vector.Multiply(100); //vector has a new length (and new orientation if the value is negative)  point.Add(vector);  //print point coordinates  Debug.WriteLine("x: " + point.x.ToString() + "y: " + point.y.ToString() + "z: " + point.z.ToString());  //Output: x: 10 y: 10 z:100  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(0, 0, 0);  p2.Create(50, 0, 0);  double distance = p1.DistanceTo(p2);  //print distance between points  Debug.WriteLine("Distance between points is: " + distance.ToString());  //Output: Distance between points is: 50.0  //return a vector oriented from p2 to p1  vector = p1.Subtract(p2);  } |

### IVector3d

#### Properties

* X (property) - Gets or sets the x dimension of the vector.
* Y (property) - Gets or sets the y dimension of the vector.
* Z (property) - Gets or sets the z dimension of the vector.

#### Methods

* Create(double dfX, double dfY, double dfZ) - Creates a vector with the specified coordinates.
* CrossProduct(IVector3d pV) - Returns a vector - the result of the cross product of this vector and specified vector. Useful when creating a CS (Coordinate system). See [Figure 1](#_Example).
* DotProduct(IVector3d ProductWithVect) - Returns the result of the scalar product of this vector and the specified vector.
* GetAngle(IVector3d inVect) - Returns the angle between the vectors (in radians).
* GetAngleWithReference(IVector3d inVect, IVector3d referenceVect) - Returns the angle between the vectors using a reference vector.
* IsPerpendicularTo(IVector3d inVect) - Returns true if this vector is perpendicular to the specified vector, otherwise it returns false.
* Multiply(double dfValue) - Multiplies the vector with the specified value – the vector has a new length (and a new orientation if the value is negative).
* Normalize() - Normalize the vector (length = 1). Does not change the orientation in any way .
* RotateBy(IVector3d inVect, double AngleVal) - Rotates the vector around the specified vector with the specified angle (in radians).

a x b

b

a

b x a

#### Example: Create 2 vectors, find the dot product, rotate the vectors and find the angles between them, check if they are perpendicular, determine their lengths.

|  |
| --- |
| private void Vectors()  {  IVector3d xAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  IVector3d yAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  xAxis.Create(1, 0, 0);  yAxis.Create(0, 1, 0);  //cross product  IVector3d zAxis = xAxis.CrossProduct(yAxis);  //dot product is: 1 if angle = 0, > 0 if angle < 90, 0 if angle = 90, < 0 if angle > 90  double dotProd = xAxis.DotProduct(yAxis);  Debug.WriteLine("Scalar product is " + dotProd.ToString());  //rotate vector by zAxis with 30 degrees  xAxis.RotateBy(zAxis, Math.PI / 6);  //Output: Scalar product is 0  //Returns the lower angle between vectors. Doesn't matter if we call function yAxis.GetAngle(xAxis), function will return same result  double dAngle = xAxis.GetAngle(yAxis);  Debug.WriteLine("Angle from x to y is " + ((dAngle \* 180) / Math.PI).ToString() + " degrees");  //Output: Angle from x to y is 60 degrees  //If we want angle on other side from xAxis to yAxis we need to use GetAngleWithReference function  zAxis.Multiply(-1); //Change zAxis direction  double dAngle2 = xAxis.GetAngleWithReference(yAxis, zAxis);  Debug.WriteLine("Angle from x to y is " + ((dAngle2 \* 180) / Math.PI).ToString() + " degrees");  //Output: Angle from x to y is 300 degrees  //Angle from yAxis to xAxis  dAngle2 = yAxis.GetAngleWithReference(xAxis, zAxis);  Debug.WriteLine("Angle from y to x is " + ((dAngle2 \* 180) / Math.PI).ToString() + " degrees");  //Output: Angle from y to x is 60 degrees  if(zAxis.IsPerpendicularTo(xAxis))  {  Debug.WriteLine("Vectors are perpendicular!");  }  //Output: Vectors are perpendicular!  zAxis.Multiply(57);  Debug.WriteLine("Vector length is " + zAxis.Length.ToString());  //Output: Vector length is 57  zAxis.Normalize(); //Normalize vector  Debug.WriteLine("Vector length is " + zAxis.Length.ToString());  //Output: Vector length is 1  } |

### ILine3d

#### Properties/Methods

* Origin - Returns the origin point of this line.
* Direction - Returns the direction vector of this line.
* CreateFromVectorAndPoint(IPoint3d inPoint, IVector3d inVect) - Creates a line with the specified origin and direction.

#### Example: Create a line.

|  |
| --- |
| private void Line()  {  IPoint3d origin = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  origin.Create(0, 0, 0);  IVector3d xAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  xAxis.Create(1, 0, 0);  ILine3d line = (ILine3d)(new DSCGEOMCOMLib.Line3d());  line.CreateFromVectorAndPoint(origin, xAxis);  } |

### IPlane

#### Properties/Methods

* PointOnPlane - Returns the origin of this plane.
* Normal - Returns the normal vector of the plane.
* CreateFromPointAndNormal(IPoint3d inPoint, IVector3d inVect) - Creates a plane with the specified origin and normal.
* get\_DistanceTo(IPoint3d ptIn) - Returns the distance from this plane to the input point.
* intersectWithLine(ILine3d inLine, out IPoint3d ptIntersection) - Calculates the intersection point of this plane with a line. Returns True or False depending on whether the objects intersect.

#### Example: Create a plane and a line; find the intersection point where the line meets the plane.

|  |
| --- |
| private void Planes()  {  IPoint3d origin = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  origin.Create(100, 100, 100);  IVector3d normal = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  normal.Create(0, 0, 1);  //create plane from point and normal  IPlane plane = (IPlane)(new DSCGEOMCOMLib.plane());  plane.CreateFromPointAndNormal(origin, normal);  IPoint3d pt = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  pt.Create(20, 30, -45);  //get perpendicular distance from point to plane  double distance = plane.get\_DistanceTo(pt);  Debug.WriteLine("Distance from point to plane is " + distance);  //Output: Distance from point to plane is 145  ILine3d line = (ILine3d)(new DSCGEOMCOMLib.Line3d());  line.CreateFromVectorAndPoint(pt, normal);  //intersect line with plane  IPoint3d ptInt;  if(plane.intersectWithLine(line, out ptInt))  {  Debug.WriteLine("Intersection point is " + ptInt.x + ", " + ptInt.y + ", " + ptInt.z);  }  //Output: Intersection point is (20, 30, 100)  } |

### ICS3d

There are two coordinate systems: a fixed system called World Coordinate System (WCS), and a movable system called User Coordinate System (UCS). When creating a beam, for example, you need to specify a coordinate system where the beam will be created. Coordinate systems can make your workeasier when creating joints.

A coordinate system is defined by three axes (vectors) and an origin point. The axes define three coordinate planes. The (XY) plane contains the x - axis and the y- axis. The (YZ) contains the y - axis and z - axis. The (XZ) plane contains x - axis and z- axis. The Origin point is at the intersection of these planes.

#### Properties

* Origin - Gets or sets the CS origin (as a Point3d).
* XAxis - Gets or sets the CS X axis (as a Vector3d).
* YAxis - Gets or sets the CS Y axis (as a Vector3d).
* ZAxis - Gets or sets the CS Z axis (as a Vector3d).

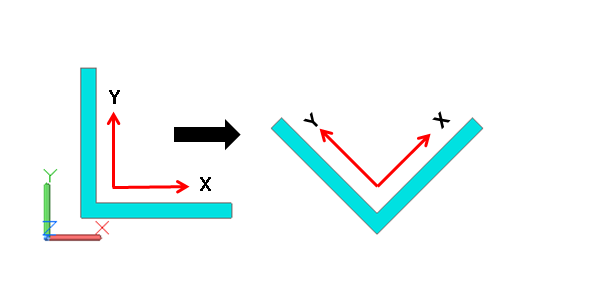
#### Methods

* RotateCSAroundX(double AngleVal) - Rotates the CS around the X axis with the specified angle value (in radians).
* RotateCSAroundY(double AngleVal) - Rotates the CS around the Y axis with the specified angle value (in radians).
* RotateCSAroundZ(double AngleVal) - Rotates the CS around the Z axis with the specified angle value (in radians).
* TransformBy(IMatrix3d TransforMatrix) - Applies the transformation specified by the matrix to this CS.
* TranslateCS(IVector3d TranslationVect) - Translates the CS origin with the specified vector. The vector is interpreted as being in this CS, not in WCS. For example, totranslate this CS along its own X-axis, the vector (1, 0, 0) multiplied by the desired value should be passed to this method.
* SetToAlignCS(ICS3d pToCS) - Returns the Matrix3d that transforms objects from this coordinate system to the “pToCS” coordinate system (the matrix can be used in an object.TransformBy(matrix) method call).

#### Example: Create a coordinate system and rotate it by 45 degrees

|  |
| --- |
| private void CreateCS()  {  IPoint3d startPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  startPoint.Create(10, 10, 0);  IPoint3d endPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  endPoint.Create(20, 20, 20);  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zAxis.Create(0, 1, 0);  ICS3d inputCS = (ICS3d)(new DSCGEOMCOMLib.CS3d());  inputCS.XAxis = startPoint.Subtract(endPoint);  inputCS.ZAxis = zAxis;  //rotate CS around X axis with 45 degrees  inputCS.RotateCSAroundX(Math.PI / 4);  } |

The output will be as follows :



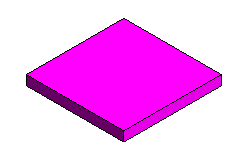
### IAugPolygon3d

* AppendVertex(IPoint3d ptToAppend) - Adds the specified point to the polygon vertices.

#### Example: Create a polygon with 4 points.

|  |
| --- |
| private void Polygon()  {  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(50, 50, 0);  p2.Create(-50, 50, 0);  p3.Create(-50, -50, 0);  p4.Create(50, -50, 0);  //create polygon  IAugPolygon3d polygon = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon.AppendVertex(p1);  polygon.AppendVertex(p2);  polygon.AppendVertex(p3);  polygon.AppendVertex(p4);  } |

Creating a plate using this polygon the output will be:



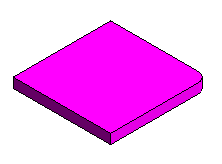
Polygon with arcs. You can do that using the AppendNewVertex function.

* AppendNewVertex(IPoint3d newVertex, IVertexInfo newVertexInfo, bool bCheckValidity)- Appends a new vertex to the polygon. If bCheckValidy is set to True, then it also reinitializes the polygon.

#### Example: Create a polygon with an arc

|  |
| --- |
| private void PolygonWithArc()  {  IVertexInfo vertexInfo = (IVertexInfo)(new DSCGEOMCOMLib.vertexInfo());  IPoint3d center = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  center.Create(40, 40, 0);  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zAxis.Create(0, 0, 1);  //create vertex info  double radius = 10;  vertexInfo.CreateFromCenterAndNormal(radius, center, zAxis);  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p5 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(50, 40, 0);  p2.Create(40, 50, 0);  p3.Create(-50, 50, 0);  p4.Create(-50, -50, 0);  p5.Create(50, -50, 0);  //create polygon  IAugPolygon3d polygon = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon.AppendNewVertex(p1, vertexInfo, true);  polygon.AppendVertex(p2);  polygon.AppendVertex(p3);  polygon.AppendVertex(p4);  polygon.AppendVertex(p5);  } |

Creating a plate using this polygon, the output will be:



## Modeling API

In this chapter you will learn how to create objects using the Steel Connections API. The Steel Connections model is built from elements such as beams, plates, structural elements, bolts, welds, features, and joints.

### Default attribute state for objects controlled by connections

Starting with Steel Connections 2021, all attributes of objects controlled by connections are accessible by default.

Steel Connections developers must now explicitly specify the attributes that the connection controls.

Using the COM API, this can be done by setting the *Attribute* property for specific attributes with the value 0 (see sample below):

|  |
| --- |
| set\_Attribute(eAttributeCodes.<kAttribute>, 0); |

Using the .NET API, setting the Attribute property is done simply by enumerating them as before in the Attributes member of the corresponding *CreatedObjectInformation* structure.

These attributes will become inaccessible from the object GUIs and will not be transferred on connection updates.

Connection developers can switch back to the old behaviour, where the attributes of controlled objects were not accessible by default. This can be accomplished by setting the IJoint.DefaultAttributeStateAccessible property to the false value. If this is done, developers must then specify the accessible attributes from within the connections, i.e., the attributes that the connection does not control, as before.

To preserve the old behaviour for existing connections, the *Version* field was introduced in the **AstorRules.mdf** database, *HRLDefinition* table. To get the old attributes state, set this value to 24 or earlier (the old behaviour means that attributes are not accessible by default, only the specified ones are accessible).

New connections should have the *Version* field set as the current internal Steel Connections version. This ensures that new attributes, not known prior to a certain version, are correctly excepted from the above rules. The version can be found by looking at the major file version (or product version) in the “Details” tab, from the “Properties” window, of any .dll file from the currently installed Steel Connections binary folder (e.g., 29 for Steel Connections 2025).

### Beams

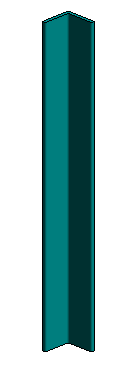
* CreateStraightBeam(string sectClass, string sectName, Role pRole, IPoint3d pt0, IPoint3d pt1, ICS3d InputCS) - Returns a straight beam object. The beam’s className and sectionName are the internal names.

section.className = value from **AstorProfiles**.*ProfileMasterTable*, column TypeNameText

section.Name = value of column SectionName , from the definition table for className,

#### Example: Create a straight beam

|  |
| --- |
| private void CreateStraightBeam(ref AstObjectsArr createdObjectsArr)  {  //get default profile  string angleSection, angleSize;  getDefaultProfile(0, "HyperSectionW", out angleSection, out angleSize);  //create role object  IRole beamRole = m\_Joint.CreateRole("Beam");  //create joint transfer  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Beam");  //set some attributes  setJointTransferForBeam(ref jointTransfer, eClassType.kBeamStraightClass);  IPoint3d startPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d endPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  startPoint.Create(0, 0, 0);  endPoint.Create(0, 0, 500);  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zAxis.Create(0, 1, 0);  ICS3d inputCS = (ICS3d)(new DSCGEOMCOMLib.CS3d());  inputCS.XAxis = startPoint.Subtract(endPoint);  inputCS.ZAxis = zAxis;  //create a straight beam  //Function will use z axis from input CS and input points to create beam CS  //xAxis = vector from input points (startPoint.Subtract(endPoint))  //yAxis = zAxis.CrossProduct(xAxis) (zAxis from CS)  //zAxis = xAxis.CrossProduct(yAxis)  IStraightBeam straightBeam = m\_Joint.CreateStraightBeam(angleSection, angleSize, (AstSTEELAUTOMATIONLib.Role)beamRole, startPoint, endPoint, inputCS);    //Add beam to created object array  if(straightBeam != null)  {  straightBeam.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  createdObjectsArr.Add(straightBeam);  }  }  //Get the default profile from the database  private void getDefaultProfile(int defaultClass, string className, out string sectionClass, out string sectionSize)  {  sectionClass = "";  sectionSize = "";  IOdbcUtils tableUtils = (IOdbcUtils)(new DSCODBCCOMLib.OdbcUtils());  string sSectionProf = tableUtils.GetDefaultString(defaultClass, className);  string separator = "#@" + '§' + "@#";  string[] section = sSectionProf.Split(new string[] { separator }, StringSplitOptions.None);  if (section.Length == 2)  {  sectionClass = section[0];  sectionSize = section[1];  }  }  //Create the JointTransfer for a beam  private void setJointTransferForBeam(ref IJointTransfer jointTrans, eClassType classType)  {  jointTrans.ClassType = classType;  //set here all the properties which can be modified outside the joint  jointTrans.set\_Attribute(eAttributeCodes.kBeamDenotation, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMaterial, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamCoating, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSinglePartNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMainPartNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSinglePartPrefix, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMainPartPrefix, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamAssembly, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamItemNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSinglePartDetailStyle, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMainPartDetailStyle, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamNote, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSPUsedForCollisionCheck, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSPUsedForNumbering, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSPDisplayRestriction, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamSPExplicitQuantity, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMPUsedForCollisionCheck, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBeamMPUsedForNumbering, 1);  } |



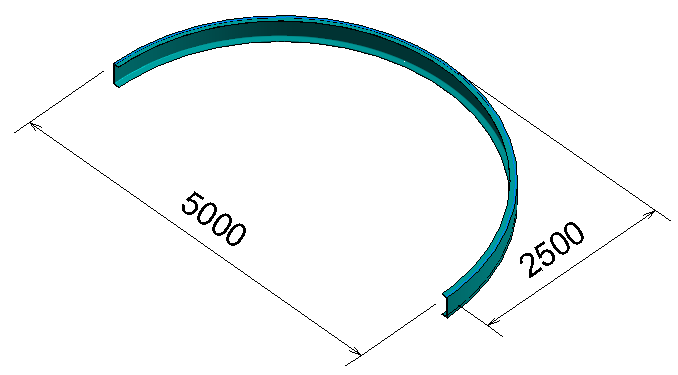
* CreateBentBeam(string sectClass, string Name, Role pRole, IPoint3d pt0, IPoint3d pt1, IPoint3d anyArcPoint, double rotAngle) - Returns a Bent Beam object. The className and sectionName of the beam are the internal names for the section.

section.className = value from **AstorProfiles**.*ProfileMasterTable*, column TypeNameText

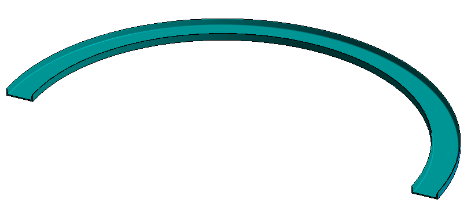
section.Name = from the definition table for className, value of column SectionName

#### Example: Create a curved beam

|  |
| --- |
| private void CreateCurvedBeam(ref AstObjectsArr createdObjectsArr)  {  //create role object  IRole beamRole = m\_Joint.CreateRole("BentBeam");  //create joint transfer  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("BentBeam");  //set joint transfer attributes  setJointTransferForBeam(ref jointTransfer, eClassType.kBeamBentClass);  //get default profile  string sectionClass, sectionSize;  getDefaultProfile(0, "HyperSectionC", out sectionClass, out sectionSize);  IPoint3d startPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d endPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d pointOnArc = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  startPoint.Create(0, 0, 0);  endPoint.Create(500, 0, 0);  pointOnArc.Create(250, 300, 0);  double rotAngle = 0; //rotation angle around the beam axis  //create bent beam  IBentBeam bentBeam = m\_Joint.CreateBentBeam(sectionClass, sectionSize, (AstSTEELAUTOMATIONLib.Role)beamRole, startPoint, endPoint, pointOnArc, rotAngle);  //Add beam to created object array  if(bentBeam != null)  {  bentBeam.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  createdObjectsArr.Add(bentBeam);  }  } |



Rotate the above beam by 90 degrees: rotAngle = 90



* CreatePolyBeam(string sectClass, string Name, Role pRole, IAugPolyline3d line, IVector3d vecRefOrientation, IVector3d zVec) - Creates and returns a poly beam object. The className and sectionName of the beam are the internal names for the section.

section.className = value from **AstorProfiles**.*ProfileMasterTable*, column TypeNameText

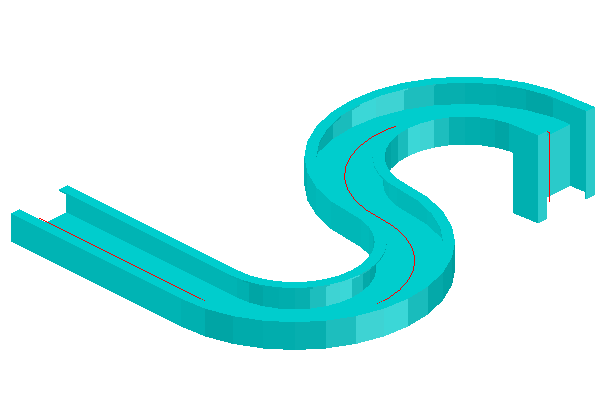
section.Name = from the definition table for className, value of column SectionName

line - defines the new poly-beam’s polyline

zVec - defines the z-direction of the polyline.

#### Example

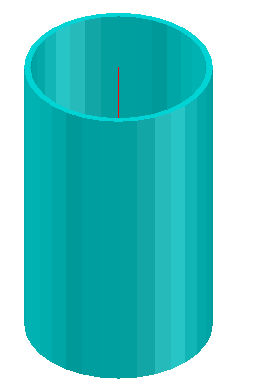
|  |
| --- |
| private void CreatePolyBeam(ref AstObjectsArr createdObjectsArr)  {  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p5 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d c1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d c2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(0, 0, 0);  p2.Create(300, 0, 0);  p3.Create(300, 300, 0);  p4.Create(300, 600, 0);  p5.Create(300, 600, -100);  c1.Create(300, 150, 0);  c2.Create(300, 450, 0);  //radius  double radius = 150;  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zAxis.Create(0, 0, 1);  //create vertex infos  IVertexInfo vertexInfo1 = (IVertexInfo)(new DSCGEOMCOMLib.vertexInfo());  IVertexInfo vertexInfo2 = (IVertexInfo)(new DSCGEOMCOMLib.vertexInfo());  vertexInfo1.CreateFromCenterAndNormal(radius, c1, zAxis);  zAxis.Multiply(-1);  vertexInfo2.CreateFromCenterAndNormal(radius, c2, zAxis);  //build polyline  IAugPolyline3d polyline = (IAugPolyline3d)(new DSCGEOMCOMLib.AugPolyline3d());  polyline.AppendVertex(p1);  polyline.AppendNewVertex(p2, vertexInfo1, true);  polyline.AppendNewVertex(p3, vertexInfo2, true);  polyline.AppendVertex(p4);  polyline.AppendVertex(p5);  //create role object  IRole beamRole = m\_Joint.CreateRole("Beam");  //create joint transfer  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Beam");  //set joint transfer attributes  setJointTransferForBeam(ref jointTransfer, eClassType.kBeamPolyClass);  //get default profile  string sectionClass, sectionSize;  getDefaultProfile(0, "HyperSectionC", out sectionClass, out sectionSize);  IVector3d zVec = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zVec.Create(0, 0, 1);  IVector3d vecRefOrientation = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  vecRefOrientation.Create(0, 1, 0);  IPolyBeam polyBeam = m\_Joint.CreatePolyBeam(sectionClass, sectionSize, (AstSTEELAUTOMATIONLib.Role)beamRole, polyline, vecRefOrientation, zVec);  //Add beam to created object array  if (polyBeam != null)  {  polyBeam.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  createdObjectsArr.Add(polyBeam);  }  } |



* CreateUnfoldedBeamWCS(IAugPolyline3d polyline, ICS3d polyLineCS, Role pRole, IPoint3d pt0, IPoint3d pt1, ICS3d InputCS) - Returns an unfolded beam object. The cross-section of the beam is defined by the polyline together with its coordinate system and the systemline of the beam passes is defined by pt0 and pt1 points.

#### Example: Create an unfolded beam

|  |
| --- |
| private void CreateUnfoldedBeam(ref AstObjectsArr createdObjectsArr)  {  //create vectors  IVector3d xAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  IVector3d yAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  xAxis.Create(1, 0, 0);  yAxis.Create(0, 1, 0);  zAxis.Create(0, 0, 1);  double diameter = 300, height = 500;  //start point of beam  IPoint3d startPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  startPoint.Create(0, 0, 0);  IPoint3d pt = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  pt.setFrom(startPoint);  xAxis.Multiply(diameter / 2);  pt.Add(xAxis);  xAxis.Normalize();  IVertexInfo vertexInfo = (IVertexInfo)(new DSCGEOMCOMLib.vertexInfo());  vertexInfo.CreateFromCenterAndNormal(diameter / 2, startPoint, zAxis);  //build polyline  IAugPolyline3d polyline = (IAugPolyline3d)(new DSCGEOMCOMLib.AugPolyline3d());  polyline.AppendNewVertex(pt, vertexInfo, false);  polyline.Reinitialize();  //end point of beam  IPoint3d endPoint = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  endPoint.setFrom(startPoint);  zAxis.Multiply(height);  endPoint.Add(zAxis);  zAxis.Normalize();  //polyline CS  ICS3d polylineCS = (ICS3d)(new DSCGEOMCOMLib.CS3d());  polylineCS.XAxis = xAxis;  polylineCS.YAxis = yAxis;  polylineCS.ZAxis = zAxis;  polylineCS.Origin = startPoint;  //create role object  IRole beamRole = m\_Joint.CreateRole("Beam");  //create joint transfer  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Beam");  //set joint transfer attributes  setJointTransferForBeam(ref jointTransfer, eClassType.kBeamUnfoldedClass);  IUnfoldedBeam unfoldedBeam = m\_Joint.CreateUnfoldedBeamWCS(polyline, polylineCS, (AstSTEELAUTOMATIONLib.Role)beamRole, startPoint, endPoint, null);  //Add beam to created object array  if(unfoldedBeam != null)  {  unfoldedBeam.Thickness = 10; //thickness  unfoldedBeam.Portioning = 1;  createdObjectsArr.Add(unfoldedBeam);  }  } |



Try creating an unfolded beam with the polyline used in creating a polybeam (Example 3.2.1.3).

### Plates

* CreatePlatePoly(Role pRole, IAugPolygon3d poly, double Thickness) - Returns a Plate object.

#### Example: Create a polygonal plate

|  |
| --- |
| private void CreatePolyPlate(ref AstObjectsArr createdObjectsArr)  {  IVertexInfo vertexInfo = (IVertexInfo)(new DSCGEOMCOMLib.vertexInfo());  IPoint3d center = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  center.Create(40, 40, 0);  IVector3d zAxis = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  zAxis.Create(0, 0, 1);  //create vertex info  double radius = 10;  vertexInfo.CreateFromCenterAndNormal(radius, center, zAxis);  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p5 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(50, 40, 0);  p2.Create(40, 50, 0);  p3.Create(-50, 50, 0);  p4.Create(-50, -50, 0);  p5.Create(50, -50, 0);  //create polygon  IAugPolygon3d polygon = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon.AppendNewVertex(p1, vertexInfo, true);  polygon.AppendVertex(p2);  polygon.AppendVertex(p3);  polygon.AppendVertex(p4);  polygon.AppendVertex(p5);  //create plate role and joint transfer  IRole plateRole = m\_Joint.CreateRole("Plate");  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Plate");  setJointTransferForPlate(ref jointTransfer);  //plate thickness  double plateThickness = 10;  //create plate  IPlate platePoly = m\_Joint.CreatePlatePoly((AstSTEELAUTOMATIONLib.Role)plateRole, polygon, plateThickness);  //Add plate to created object array  if (platePoly != null)  {  //set joint transfer  platePoly.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  platePoly.Portioning = 0;  createdObjectsArr.Add(platePoly);  }  }  private void setJointTransferForPlate(ref IJointTransfer jointTrans)  {  jointTrans.ClassType = eClassType.kPlateClass;  //set here all the properties which can be modified outside the joint  jointTrans.set\_Attribute(eAttributeCodes.kPlateDenotation, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMaterial, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateCoating, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSinglePartNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMainPartNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSinglePartPrefix, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMainPartPrefix, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateAssembly, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateItemNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSinglePartDetailStyle, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMainPartDetailStyle, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateNote, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSPUsedForCollisionCheck, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSPUsedForNumbering, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSPDisplayRestriction, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateSPExplicitQuantity, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMPUsedForCollisionCheck, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMPUsedForNumbering, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMPDisplayRestriction, 1);  jointTrans.set\_Attribute(eAttributeCodes.kPlateMPExplicitQuantity, 1);  } |

* CreatePlateRectangular(Role pRole, double dLength, double dWidth, double dThickness, ICS3d cs) - Returns a plate object with the specified length, width and thickness in the specified coordinate system. The center of the plate will be in the origin of the given coordinate system.

#### Example: Create a rectangular plate.

|  |
| --- |
| Private void CreateRectangularPlate(ref AstObjectsArr createdObjectsArr)  {  Ipoint3d origin = (Ipoint3d)(new DSCGEOMCOMLib.Point3d());  origin.Create(0, 0, 0);  Ivector3d xAxis = (Ivector3d)(new DSCGEOMCOMLib.Vector3d());  Ivector3d yAxis = (Ivector3d)(new DSCGEOMCOMLib.Vector3d());  Ivector3d zAxis = (Ivector3d)(new DSCGEOMCOMLib.Vector3d());  xAxis.Create(1, 0, 0);  yAxis.Create(0, 1, 0);  zAxis.Create(0, 0, 1);  ICS3d csPlate = (ICS3d)(new DSCGEOMCOMLib.CS3d());  csPlate.Origin = origin;  csPlate.Xaxis = xAxis;  csPlate.Yaxis = yAxis;  csPlate.Zaxis = zAxis;  //create plate role and joint transfer  Irole plateRole = m\_Joint.CreateRole(“Plate”);  IjointTransfer jointTransfer = m\_Joint.CreateJointTransfer(“Plate”);  setJointTransferForPlate(ref jointTransfer);  double dPlateLength = 400, dPlateWidth = 200, dPlateThickness = 10;  //create rectangular plate  Iplate rectangularPlate = m\_Joint.CreatePlateRectangular((AstSTEELAUTOMATIONLib.Role)plateRole, dPlateLength, dPlateWidth, dPlateThickness, csPlate);    //Add plate to created object array  if (rectangularPlate != null)  {  //set joint transfer  rectangularPlate.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  rectangularPlate.Portioning = 0;  createdObjectsArr.Add(rectangularPlate);  }  } |

#### Example: Create a folded plate using 6 plates.

|  |
| --- |
| private void CreateFoldedPlates (ref AstObjectsArr createdObjectsArr)  {  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1.Create(50, 50, 0);  p2.Create(-50, 50, 0);  p3.Create(-50, -50, 0);  p4.Create(50, -50, 0);  //create polygon 1  IAugPolygon3d polygon = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon.AppendVertex(p1);  polygon.AppendVertex(p2);  polygon.AppendVertex(p3);  polygon.AppendVertex(p4);  //create plate role and joint transfer  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("FoldedPlate");  setJointTransferForPlate(ref jointTransfer);  IJointTransfer jointTransfer2 = m\_Joint.CreateJointTransfer("FoldedPlateRelation");  jointTransfer.ClassType = eClassType.kPlateFoldedClass;  jointTransfer2.ClassType = eClassType.kPlateFoldRelationClass;  //plate thickness  double plateThickness = 10;  //create first plate of folded plate  IPlateFold platePolyFold = m\_Joint.CreatePolyPlateFold(polygon, plateThickness);  IPoint3d p1b = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2b = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3b = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4b = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1b.Create(100, -45, 75);  p2b.Create(100, 45, 75);  p3b.Create(50, 45, 0);  p4b.Create(50, -45, 0);  //create polygon 2  IAugPolygon3d polygon2 = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon2.AppendVertex(p1b);  polygon2.AppendVertex(p2b);  polygon2.AppendVertex(p3b);  polygon2.AppendVertex(p4b);  //create second plate of folded plate  IPlateFold platePolyFold2 = m\_Joint.CreatePolyPlateFold(polygon2, plateThickness);  IPoint3d p1c = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2c = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3c = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4c = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1c.Create(-50, -45, 0);  p2c.Create(-50, 45, 0);  p3c.Create(-100, 45, 75);  p4c.Create(-100, -45, 75);  //create polygon 3  IAugPolygon3d polygon3 = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon3.AppendVertex(p1c);  polygon3.AppendVertex(p2c);  polygon3.AppendVertex(p3c);  polygon3.AppendVertex(p4c);  IPlateFold platePolyFold3 = m\_Joint.CreatePolyPlateFold(polygon3, plateThickness);  IPoint3d p1d = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2d = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3d = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4d = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1d.Create(-45, 50, 0);  p2d.Create(45, 50, 0);  p3d.Create(45, 100, 75);  p4d.Create(-45, 100, 75);  //create polygon 4  IAugPolygon3d polygon4 = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon4.AppendVertex(p1d);  polygon4.AppendVertex(p2d);  polygon4.AppendVertex(p3d);  polygon4.AppendVertex(p4d);  IPlateFold platePolyFold4 = m\_Joint.CreatePolyPlateFold(polygon4, plateThickness);  IPoint3d p1e = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2e = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3e = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4e = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1e.Create(-45, -100, 75);  p2e.Create(45, -100, 75);  p3e.Create(45, -50, 0);  p4e.Create(-45, -50, 0);  //create polygon 5  IAugPolygon3d polygon5 = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon5.AppendVertex(p1e);  polygon5.AppendVertex(p2e);  polygon5.AppendVertex(p3e);  polygon5.AppendVertex(p4e);  IPlateFold platePolyFold5 = m\_Joint.CreatePolyPlateFold(polygon5, plateThickness);  IPoint3d p1f = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2f = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3f = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4f = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  p1f.Create(100, -45, 75);  p2f.Create(100, 45, 75);  p3f.Create(200, 45, 50);  p4f.Create(200, -45, 50);  //create polygon 6  IAugPolygon3d polygon6 = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  polygon6.AppendVertex(p1f);  polygon6.AppendVertex(p2f);  polygon6.AppendVertex(p3f);  polygon6.AppendVertex(p4f);  IPlateFold platePolyFold6 = m\_Joint.CreatePolyPlateFold(polygon6, plateThickness);  if ((platePolyFold != null) && (platePolyFold2 != null))  {  IRole foldedPlateRole = m\_Joint.CreateRole("FoldedPlate");  string relationRole = "Relation";  int stringerID;  //create folded plate using first plate created  IPlateFolded foldedPlate = m\_Joint.CreateFoldedPlate((AstSTEELAUTOMATIONLib.Role)foldedPlateRole, (AstSTEELAUTOMATIONLib.PlateFold)platePolyFold, out stringerID);  if (foldedPlate != null)  {  createdObjectsArr.Add(foldedPlate);  int stringerID2;  PlateFoldRelation plateFoldRelation;  FeaturesArray plateFoldFeatures;  bool ok;  //extend the folded plate to the second created plate and get the Relation between the 2 plates  ok = foldedPlate.ExtendBy((AstSTEELAUTOMATIONLib.PlateFold)platePolyFold2, 1, stringerID, p1, p3b, relationRole, out stringerID2, out plateFoldRelation, out plateFoldFeatures);  foldedPlate.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;    int stringerIDFold2 = stringerID2; //remember this to connect plate #6  if (plateFoldRelation != null)  {  plateFoldRelation.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer2;  createdObjectsArr.Add(plateFoldRelation);  plateFoldRelation.Radius = 10; //set the radius  }  if (plateFoldFeatures != null)  {  int noFeats = (int)plateFoldFeatures.Count;  IAstFeatObject currFeat;  for (int i = 0; i < noFeats; i++)  {  currFeat = plateFoldFeatures[i];  createdObjectsArr.Add(currFeat);  }  }  if (platePolyFold3 != null)  {  ok = foldedPlate.ExtendBy((AstSTEELAUTOMATIONLib.PlateFold)platePolyFold3, 1, stringerID, p2, p2c, relationRole, out stringerID2, out plateFoldRelation, out plateFoldFeatures);  if (plateFoldRelation != null)  {  plateFoldRelation.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer2;  createdObjectsArr.Add(plateFoldRelation);  plateFoldRelation.Radius = 10; //set the radius  }  if (plateFoldFeatures != null)  {  int noFeats = (int)plateFoldFeatures.Count;  IAstFeatObject currFeat;  for (int i = 0; i < noFeats; i++)  {  currFeat = plateFoldFeatures[i];  createdObjectsArr.Add(currFeat);  }  }  }  if (platePolyFold4 != null)  {  ok = foldedPlate.ExtendBy((AstSTEELAUTOMATIONLib.PlateFold)platePolyFold4, 1, stringerID, p1, p2d, relationRole, out stringerID2, out plateFoldRelation, out plateFoldFeatures);  if (plateFoldRelation != null)  {  plateFoldRelation.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer2;  createdObjectsArr.Add(plateFoldRelation);  plateFoldRelation.Radius = 10; //set the radius  }  if (plateFoldFeatures != null)  {  int noFeats = (int)plateFoldFeatures.Count;  IAstFeatObject currFeat;  for (int i = 0; i < noFeats; i++)  {  currFeat = plateFoldFeatures[i];  createdObjectsArr.Add(currFeat);  }  }  }  if (platePolyFold5 != null)  {  ok = foldedPlate.ExtendBy((AstSTEELAUTOMATIONLib.PlateFold)platePolyFold5, 1, stringerID, p3, p4e, relationRole, out stringerID2, out plateFoldRelation, out plateFoldFeatures);  if (plateFoldRelation != null)  {  plateFoldRelation.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer2;  createdObjectsArr.Add(plateFoldRelation);  plateFoldRelation.Radius = 10; //set the radius  }  if (plateFoldFeatures != null)  {  int noFeats = (int)plateFoldFeatures.Count;  IAstFeatObject currFeat;  for (int i = 0; i < noFeats; i++)  {  currFeat = plateFoldFeatures[i];  createdObjectsArr.Add(currFeat);  }  }  }  if (platePolyFold6 != null)  {  ok = foldedPlate.ExtendBy((AstSTEELAUTOMATIONLib.PlateFold)platePolyFold6, 1, stringerIDFold2, p1b, p1f, relationRole, out stringerID2, out plateFoldRelation, out plateFoldFeatures);  if (plateFoldRelation != null)  {  plateFoldRelation.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer2;  createdObjectsArr.Add(plateFoldRelation);  plateFoldRelation.Radius = 10; //set the radius  }  if (plateFoldFeatures != null)  {  int noFeats = (int)plateFoldFeatures.Count;  IAstFeatObject currFeat;  for (int i = 0; i < noFeats; i++)  {  currFeat = plateFoldFeatures[i];  createdObjectsArr.Add(currFeat);  }  }  }  }  }  } |

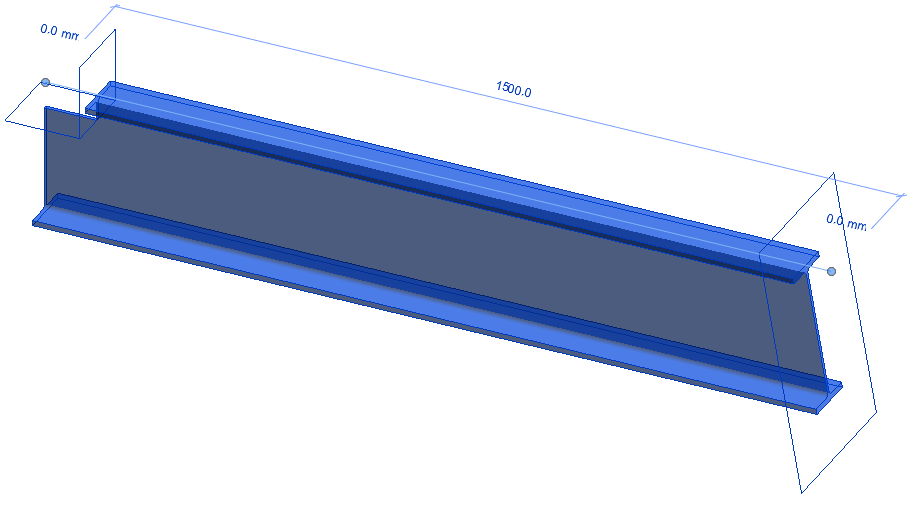
### Features

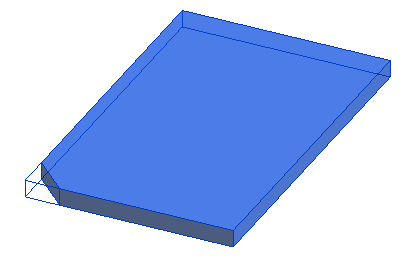
The basic objects (i.e., beams and plates) can have processing features. The most important processing features types are:

Beam processing features: coping, miter cuts, rectangular and circular contour cuts, or any type of contour.

Plate processing features: corner finishes, chamfers, outer plate contours and inner contours, etc.

The processing objects can be selected using the Tab key. The processing objects cannot exist alone and are part of a basic element (i.e., beam or plate). The object processing are edited as individual objects. Ex: see beam with Shorten and Cope skewed processing features – and plate with corner cut processing feature.





The variety of processing options in Steel Connections allows you to create almost any beam and plate contour.

If a basic element is deleted all processing objects will also be deleted.

* addBeamShortening(eBeamEnd endOfBeam, IPlane shorteningPlane) - Shortens the beam with the specified plane at the given beam end. Returns and creates a beam shortening object. For information about possible values of eBeamEnd, see the Appendix (“Steel Connections Developer Guide”).

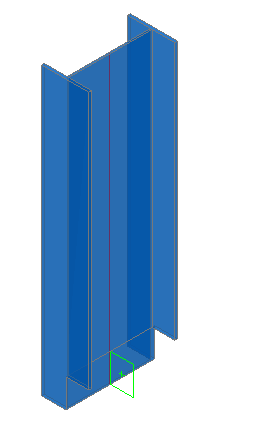
#### Example: Create a shortening on a beam.

|  |
| --- |
| private void AddBeamShortening (ref AstObjectsArr createdObjectsArr, IBeam inputBeam)  {  //get beam cs at start  ICS3d csBeam = inputBeam.get\_PhysicalCSAt(eBeamEnd.kBeamStart);  IPoint3d origin = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  origin.setFrom(csBeam.Origin);  IVector3d vDir = (IVector3d)(new DSCGEOMCOMLib.Vector3d());  vDir.setFrom(csBeam.XAxis);  vDir.RotateBy(csBeam.YAxis, Math.PI / 6);  vDir.Normalize();  origin.Add(vDir);  //add beam Shortening  IPlane plShortening = (IPlane)(new DSCGEOMCOMLib.plane());  plShortening.CreateFromPointAndNormal(origin, vDir);  IBeamShortening shortening = inputBeam.addBeamShortening(eBeamEnd.kBeamStart, plShortening);  if (shortening != null)  createdObjectsArr.Add(shortening);  } |

* addBeamMultiContourNotch(Role notchRole, eBeamEnd endOfBeam, IAugPolygon3d notchPts) - Creates a notch at the specified beam end with a given role and contour. For information about possible values of eBeamEnd, see the Appendix (“Steel Connections Developer Guide”).

#### Example: Create a contour notch on a beam.

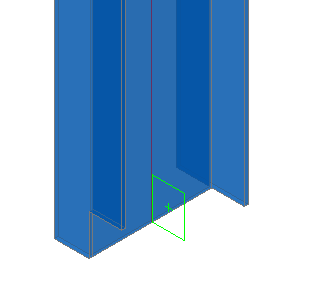
|  |
| --- |
| using JointDesignUtils;  /\* … \*/  private void AddBeamContourNotch (ref AstObjectsArr createdObjectsArr, IBeam inputBeam)  {  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d startPoint = inputBeam.PhysicalCSStart.Origin;  IPoint3d endPoint = inputBeam.PhysicalCSEnd.Origin;  IVector3d xAxis = startPoint.Subtract(endPoint);  //retrieve beam geometrical data  IProfType profType = inputBeam.getProfType();  double dWidth = profType.getGeometricalData(eProfCommonData.kWidth); //profile width  double dHeight = profType.getGeometricalData(eProfCommonData. kProfHeight); //profile height  double dWeb = profType.getGeometricalData(eProfCommonData.kWeb); //profile web  p1.setFrom(startPoint);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.ZAxis, dHeight / 2, out p1);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.YAxis, dWeb / 2, out p1);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.YAxis, (dWidth - dWeb) / 2, out p2);  Utils.movePoint(p2, xAxis, -100, out p3);  Utils.movePoint(p1, xAxis, -100, out p4);  //create role object  IRole notchRole = m\_Joint.CreateRole("Feature");  notchRole.ClassType = eClassType.kBeamMultiContourNotch;  //compute contour  IAugPolygon3d contourNotch = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  contourNotch.AppendVertex(p1);  contourNotch.AppendVertex(p2);  contourNotch.AppendVertex(p3);  contourNotch.AppendVertex(p4);  //add beam notch  IBeamMultiContourNotch multiContourNotch = inputBeam.addBeamMultiContourNotch((AstSTEELAUTOMATIONLib.Role)notchRole, eBeamEnd.kBeamStart, contourNotch);  //Add notch to created object array  if (multiContourNotch != null)  {  createdObjectsArr.Add(multiContourNotch);  }  } |



* addBeamMultiContourNotchClip(Role notchRole, eBeamEnd endOfBeam, IAugPolygon3d notchPts, IPoint3d pt0, IPoint3d pt1) - Creates a notch at the specified beam end with given role and points that define it. The notch has z clipping with the clipping heights defined by pt0 and pt1. Returns and creates a BeamMultiContourNotch object. For information about possible values of eBeamEnd, see the Appendix (“Steel Connections Developer Guide”).

#### Example: Create a clipped contour notch on a beam.

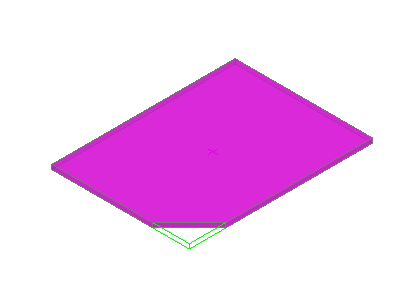
|  |
| --- |
| using JointDesignUtils;  /\* … \*/  private void AddBeamContourNotchClip (ref AstObjectsArr createdObjectsArr, IBeam inputBeam)  {  IPoint3d p1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p2 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p3 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d p4 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d startPoint = inputBeam.PhysicalCSStart.Origin;  IPoint3d endPoint = inputBeam.PhysicalCSEnd.Origin;  IVector3d xAxis = startPoint.Subtract(endPoint);  //retrieve beam geometrical data  IProfType profType = inputBeam.getProfType();  double dWidth = profType.getGeometricalData(eProfCommonData.kWidth); //profile width  double dHeight = profType.getGeometricalData(eProfCommonData.kProfHeight); //profile height  double dWeb = profType.getGeometricalData(eProfCommonData.kWeb); //profile web  p1.setFrom(startPoint);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.ZAxis, dHeight / 2, out p1);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.YAxis, dWeb / 2, out p1);  Utils.movePoint(p1, inputBeam.PhysicalCSStart.YAxis, (dWidth - dWeb) / 2, out p2);  Utils.movePoint(p2, xAxis, -100, out p3);  Utils.movePoint(p1, xAxis, -100, out p4);  //create role object  IRole notchRole = m\_Joint.CreateRole("Feature");  notchRole.ClassType = eClassType.kBeamMultiContourNotch;  //compute contour  IAugPolygon3d contourNotch = (IAugPolygon3d)(new DSCGEOMCOMLib.AugPolygon3d());  contourNotch.AppendVertex(p1);  contourNotch.AppendVertex(p2);  contourNotch.AppendVertex(p3);  contourNotch.AppendVertex(p4);  IPoint3d u0 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  IPoint3d u1 = (IPoint3d)(new DSCGEOMCOMLib.Point3d());  u0.setFrom(p1);  double dFlange = profType.getGeometricalData(eProfCommonData.kFlange);  Utils.movePoint(u0, inputBeam.PhysicalCSStart.ZAxis, -dFlange - 1e-3, out u1);  IBeamMultiContourNotch multiContourNotch = inputBeam.addBeamMultiContourNotchClip((AstSTEELAUTOMATIONLib.Role)notchRole, eBeamEnd.kBeamStart, contourNotch, u0, u1);  //Add notch to created object array  if (multiContourNotch != null)  {  createdObjectsArr.Add(multiContourNotch);  }  } |



* addChamfer(double a\_val, double b\_val, int VertexIndex) - Returns and creates a Chamfer object. The plate corner is identified by the vertexIndex.

#### Example: Create a chamfer on a plate.

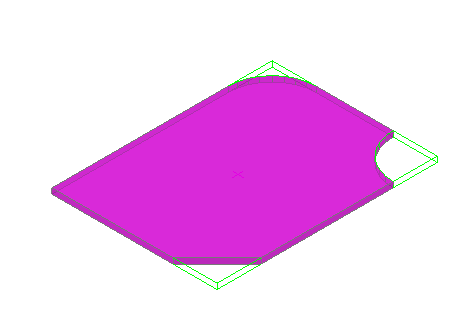
|  |
| --- |
| private void AddPlateChamfer (ref AstObjectsArr createdObjectsArr, IPlate basePlate)  {  double dWidth = 80, dHeight = 80;  VertexFeat chamfer = basePlate.addChamfer(dWidth, dHeight, 0);  if (chamfer != null)  {  createdObjectsArr.Add(chamfer);  }  } |



* addFillet(double rad\_val, eFilletTypes fillet\_type, int VertexIndex) - Returns and creates a Fillet object. The plate corner is identified by the vertexIndex. For information about possible values of eFilletTypes, see the Appendix (“Steel Connections Developer Guide”).

#### Example: Create fillet on a plate.

|  |
| --- |
| private void AddPlateChamfer (ref AstObjectsArr createdObjectsArr, IPlate basePlate)  {  double dWidth = 80;  VertexFeat fillet = basePlate.addFillet(dWidth, eFilletTypes.kFillet\_Concav, 1);  if(fillet != null)  {  createdObjectsArr.Add(fillet);  }  VertexFeat fillet2 = basePlate.addFillet(dWidth, eFilletTypes.kFillet\_Convex, 2);  if (fillet2 != null)  {  createdObjectsArr.Add(fillet2);  }  } |



### Special parts

Objects that are not Steel Connections standard objects can be created as special parts. When Steel Connections creates models with special parts they are handled like standard objects.

* CreateSpecialPart(Role pRole, double Scale, string BlockName, ICS3d pCS) - Creates and returns a SpecialPart object. For insertion point, the inputCS origin must be correctly specified.

#### Example: Create a special part.

|  |
| --- |
| private void AddSpecialPart (ref AstObjectsArr createdObjectsArr)  {  ICS3d cs = (ICS3d)(new DSCGEOMCOMLib.CS3d());  // set the coordinate system  /\* … \*/  double dScale = 1.0;  string blockName = "agcal12";  //create role object  IRole partRole = m\_Joint.CreateRole("block");  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("block");  jointTransfer.ClassType = eClassType.kSpecialPartWithBlock;  //create special part  ISpecialPart specialPart = m\_Joint.CreateSpecialPart((AstSTEELAUTOMATIONLib.Role)partRole, dScale, blockName, cs);  //Add special part to created object array  if(specialPart != null)  {  specialPart.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  createdObjectsArr.Add(specialPart);  }  } |

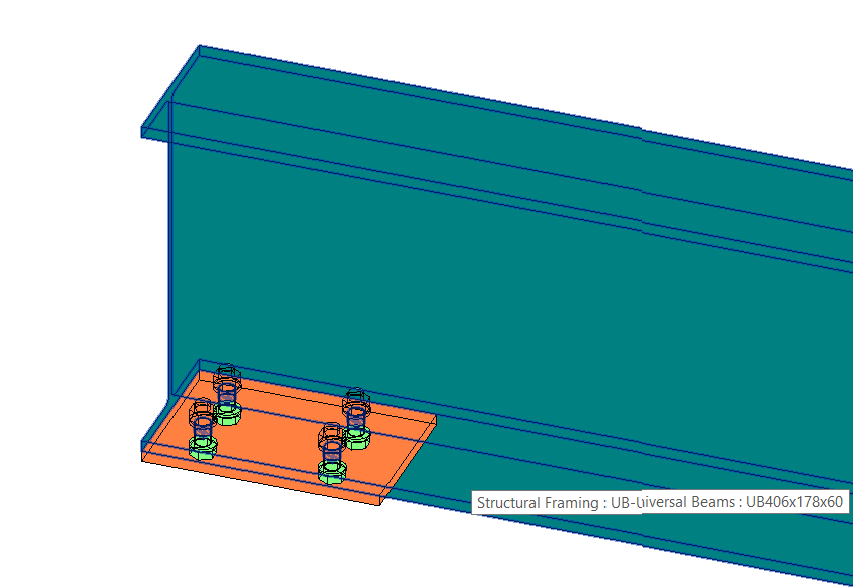
### Bolts, Welds and Anchors

The basic objects (i.e., beams and plates) can be connected using:

* Bolt patterns (or holes only)
* Welds
* Anchors

These objects establish a connection between objects (e.g., beams and plates). This information is stored on the objects (i.e., beam or plate) including any bolt pattern (with its definition) or welds (with their relevant properties). Any individual element in the connection recognizes what holes, bolts, or welds it contains or which element it is connected to.

A bolt pattern can describe one or several bolts, which are automatically created in any plane together with the appropriate holes.



Changes in the bolt pattern automatically update the holes.

The tools for creating bolt patterns are used for bolts in addition to:

* Holes, slotted holes, countersunk holes, blind holes, threaded holes, and punch marks.
* Shear studs.

All the above are created with their respective properties or definitions.

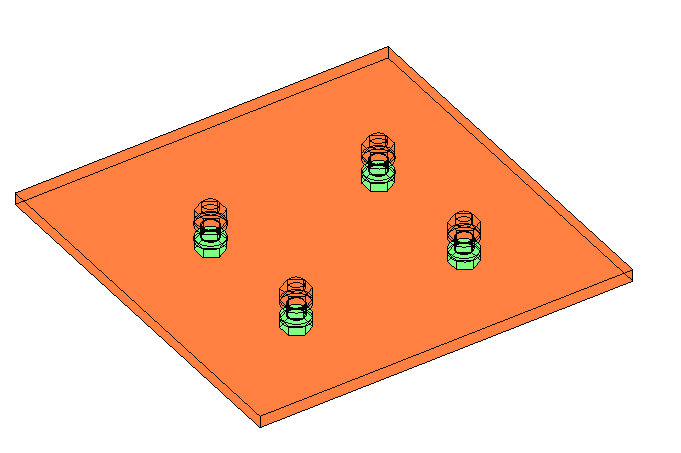
It is also possible to create various hole types as part of a bolt object and a separate hole object.

Weld points are displayed as crosses in the model.

* CreateBoltFinitRect(Role pRole, string Material, string norm, double wx\_dim, double wy\_dim, double dx\_dim, double dy\_dim, double nx\_dim, double ny\_dim, double diameter, ICS3d coordSys) - Returns a Bolt object. Creates a bolt pattern of type finite, rectangular.

#### Example: Add bolts to a plate.

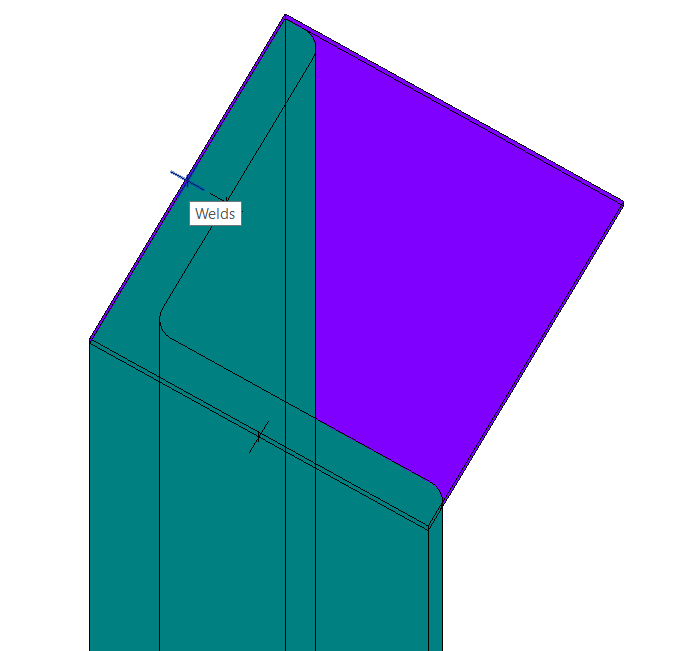
|  |
| --- |
| private void AddBolts (ref AstObjectsArr createdObjectsArr, IPlate basePlate)  {  //read defaults from database  IOdbcUtils tableUtils = (IOdbcUtils)(new DSCODBCCOMLib.OdbcUtils());  string sBoltType = tableUtils.GetDefaultString(401, "Norm");  string sBoltGrade = tableUtils.GetDefaultString(401, "Material");  string sBoltAssembly = tableUtils.GetDefaultString(401, "Garnitur");  double dBoltDiameter = tableUtils.GetDefaultDouble(401, "Diameter");  //Create bolts  IRole boltRole = m\_Joint.CreateRole("Bolt"); //role object  //create joint transfer object  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Bolt#1");  setJointTransferForBolt(ref jointTransfer);  ICS3d csPlate = (ICS3d)(new DSCGEOMCOMLib.CS3d());  // set the coordinate system  /\* … \*/  //create bolt pattern  IBolt bolt = m\_Joint.CreateBoltFinitRect((AstSTEELAUTOMATIONLib.Role)boltRole, sBoltGrade, sBoltType, 0, 0, 100, 100, 2, 2, dBoltDiameter, csPlate);  //Add bolt to created object array  if (bolt != null)  {  //set joint transfer  bolt.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTransfer;  bolt.SetHoleTolerance(2, true);  bolt.BoltSet = sBoltAssembly;  //connect objects  AstObjectsArr conObj = m\_Joint.CreateObjectsArray();  conObj.Add(basePlate);  bolt.Connect(conObj, eAssembleLocation.kOnSite);  createdObjectsArr.Add(bolt);  }  }  private void setJointTransferForBolt(ref IJointTransfer jointTrans)  {  jointTrans.ClassType = eClassType.kFinitrectScrewBoltPattern;  //set here all the properties which can be modified outside the joint  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonGripLengthAddition, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonHoleTolerance, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonCoating, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonDenotation, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonAssembly, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonItemNumber, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonInvertAble, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonNote, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonIgnoreMaxGap, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonSPUsedForCollisionCheck, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonSPUsedForNumbering, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonSPUsedForBillOfMaterial, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonSPExplicitQuantity, 1);  jointTrans.set\_Attribute(eAttributeCodes.kBoltPatternCommonRole, 1);  } |



* CreateWeld(Role pRole, eWeldType Type, double Thickness, IPoint3d pLocation, ICS3d coordSys) - Returns a weld object with the specified type and thickness.

#### Example: Add a weld between a beam and a plate.

|  |
| --- |
| private void AddWeld (ref AstObjectsArr createdObjectsArr, IPlate basePlate, IBeam inputBeam)  {  //create new role object  IRole weldRole = m\_Joint.CreateRole("Weld");  //create new joint transfer object  IJointTransfer jointTrans = m\_Joint.CreateJointTransfer("Weld#1");  //set some attributes  jointTrans.set\_Attribute(eAttributeCodes.kWeldPatternThickness, 1);  jointTrans.set\_Attribute(eAttributeCodes.kWeldPatternAssembleLocation, 0);  jointTrans.set\_Attribute(eAttributeCodes.kWeldPatternSeamType, 1);  IWeld weld = m\_Joint.CreateWeld((AstSTEELAUTOMATIONLib.Role)weldRole, eWeldType.kTWeld, 4, point, csAtPoint);  //Add weld to created object array  if (weld != null)  {  //set joint transfer  weld.JointTransfer = (AstSTEELAUTOMATIONLib.JointTransfer)jointTrans;  createdObjectsArr.Add(weld);  //connect objects  AstObjectsArr conObj = m\_Joint.CreateObjectsArray();  conObj.Add(inputBeam);  conObj.Add(basePlate);  weld.Connect(conObj, eAssembleLocation.kInShop);  }  } |



#### Example: Add a weld preparation on a plate.

|  |
| --- |
| private void AddWeldPreparation (ref AstObjectsArr createdObjectsArr, IPlate basePlate)  {  //create weld preparation  IEdgeFeat weldBevel;  //create weld role  IRole weldRole = m\_Joint.CreateRole("Feature");  int edgeIndex = 1; //which edge of plate  double xDist = 10; //x dist of weld prep  double yDist = 5; // ydist of weld prep  int plateSide = 0; // side of plate  weldBevel = platePoly.addPlateWeldingBevel((AstSTEELAUTOMATIONLib.Role)weldRole, edgeIndex, xDist, yDist, plateSide);  if (weldBevel != null)  createdObjectsArr.Add(weldBevel);  } |

# Chapter 4 Joints API

## User interface

All the attributes of a joint can be seen and modified from a property sheet dialog box. A joint can be modified by instance (using the Modify Parameters option from the ribbon or using the Detailed parameters button from the Properties dialog), or by type (from Edit Type, Modify parameters button).

The rule designer must design all the other pages. This is done by implementing the *GetUserPages* method of the *IRule*.

## Query method

The *Query* method implementation uses the provided *AstUI* object and gets user input regarding necessary parameters or objects. The objects must be added to the *Joint.InputObjects*. The selected objects will become “driver objects” for the joint. Modification of the “driver objects” will prompt an update of the Joint.

## CreateObjects method

*CreateObjects* method implementation creates Steel Connections objects and adds them to the *CreatedObjects* of the *Joint*. The objects can be created by using specific methods of the *Joint* object. Once the objects are created, they are automatically added to the model. The connection between them and the Joint object is done after they are added to the *Joint*.*CreatedObjects*. Although it is possible to create objects without adding them to *Joint*.*CreatedObjects*, doing this is wrong because the connection is not created, and those objects willnot be part of the joint. They will not be erased when updating the joint and others will be created.

## InField/OutField methods

Steel Connections calls the *OutField* method every time it seems necessary to get the values and / or names of the *Rule* attributes. Similarly, the *InField* method is called every time it is necessary for the *Rule* to change its current attribute values. For example, calls to these methods will happen when a model containing a joint created with this Rule is opened/saved, when the user changes the joint parameters in the joint properties dialog box, etc.

## GetUserPages method

By implementing this method, you can specify the GUI of this Rule. This method should return handles to the created windows. The windows will become property pages for the Rule dialog box. Also, here you should assign the corresponding prompts for the Rule dialog box title and for the PropertyPages titles.

#### Example

|  |
| --- |
| //Set Title(From AstCrtlDb.ErrorMessages)  pPropSheetData.SheetPrompt = 81309;  //First Page bitmap index(From AstorBitmaps)  pPropSheetData.FirstPageBitmapIndex = 60782;  pPropSheetData.ResizeOption = eGUIDimension.kStandard;  //Property Sheet 1  RulePage rulePage1 = m\_Joint.CreateRulePage();  rulePage1.title = 88438; //Base plate layout  m\_Page1 = new Page1(this);  rulePage1.hWnd = m\_Page1.Handle.ToInt64();  pagesRet.Add(rulePage1); |

## GetTableName method

Returns the name of the table used by the rule to run.

## GetExportData method

This is an optional method to use. By implementing this method, the *Rule* can specify the necessary external data it needs to run. Based on this information a tool can import / export data from / to Steel Connections databases.

## GetFeatureName method

This method can be used if you want to license your joint to a specific license feature. If you intend to develop a joint which will work only if the license file has that feature, return from this method the license feature name. Return VARIANT\_FALSE if you do not want to use a specific license feature.

## FreeUserPages method

Releases user pages.

## Libraries

Steel Connections provides several libraries that are intended to be used when implementing a Rule.

* **AstSteelAutomation** library – provide access to all Steel Connections model objects.
* **DSCGeomCom** library – Geometry library, useful for geometric calculation (vectors, coordinates, points, parametric curves).
* **DSCProfilesAccesCom** library – Provides access to the defined profiles used by Steel Connections.
* **DSCUtilFacetCom** library - Provides access to the geometric body of a Steel Connections object.
* **DSCOdbcCom** library - Provides access to data used by Steel Connections and stored in external databases.
* **AstControls** library – Provides several controls intended to be used for the GUI of the joints. Mainly,this library has regular controls but uses database prompts (language dependent) instead of hardcoded prompts. Also, this library has a Bitmap control that uses bitmaps stored in **AstorBitmaps** database or in .dll files.

## Attributes and categories

Whenever you need to create a Steel Connections object, you should define a new IRole object. This object is used to identify objects from the model. To create a rule, you need to assign a name from **AstorBase.mdf** database, *ModelRole* table, corresponding to your object.

|  |
| --- |
| // create role object  IRole role = m\_Joint.CreateRole("Haunch"); // name from AstorBase.ModelRole table |

To set the user attributes of an object:

|  |
| --- |
| role.set\_Attribute(eAttributeCodes.kUserAttr1, 0);  string userAttribute = beam.get\_UserAttribute(0);  beam.set\_UserAttribute(0, "value"); |

## JointTransfer

This object will be passed after it is created to almost all the methods that create Steel Connections objects. The joint transfer has mainly two “visible” applications: to identify objects with identical geometry and properties and to set which of the properties of joint created objects can be set by the user outside of the joint.

To define a complete joint transfer means to assign a name, the class type of the objects it will be applied to, and to define which of the properties of the object will be editable outside the joint.

After setting the object class type, you must specify which of the attributes will be editable. In Visual C# syntax, this is what you must do:

|  |
| --- |
| // create the joint transfer object  IJointTransfer jointTransfer = m\_Joint.CreateJointTransfer("Beam");  // this joint transfer will be used when creating a straight beam  jointTransfer.ClassType = eClassType.kBeamStraightClass;  //set here all the properties which can be modified outside the joint  jointTransfer.set\_Attribute(eAttributeCodes.kBeamMaterial, 1); //means the beam material can be set outside the joint  jointTransfer.set\_Attribute(eAttributeCodes.kBeamCoating, 1); //other editable properties of object with this joint transfer |

If a joint transfer attribute should not be modified outside the joint (this is the default state of attributes – read only), you could use:

jointTransfer.set\_Attribute(eAttributeCodes.kBeamMaterial, 0); //means the beam material cannot be set outside the joint

## Database Structure

Usually, the following databases are used for joint development.

**AstCrtlDb.mdf** - used to store the localized texts the joint uses to display in Steel Connections dialog box pages.

**AstorRules.mdf** - used to store the joint definition records (*HRLDefinition* and *RulesDllSigned* tables) and to store any table the joint uses.

The definition record in *HRLDefinition* table requires the specification of the GUID value on the ClassID column.

#### Relationships

AstorRules.HRLDefintion

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Key** | **RuleRunName** | **InternalName** | **Category** | **Dll** | **SubNameInDll** | **ClassID** | **…** |
| 200000 | Create Plate | CreatePlate | CreatePlate | 1000 | CreatePlate | {DFB7A86F-7E86-40F2-8E58-C16C776F7464} |  |

|  |
| --- |
|  |

AstorRules.RulesDllSigned table

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **FileName** | **Tech** | **Signature** |
| 1000 | SampleJoint.dll | 1 | NULL |

For joints created in pure .Net (without COM technology), the SubNameInDll field should contain the fully qualified name of the class. E.g.: SampleJoint.CreatePlate (where SampleJoint is the namespace and CreatePlate is the class name). Also, for joints created in pure .Net, the ClassID field should be a new generated GUID.

## Joint table

Saving and getting joint table records from the joint table is not available in the Revit UI.

Each joint can store its parameters to **AstorRules.mdf** database in its own table. You have the flexibility to save different joint configurations, and based on the user input, to select one of them when the joint is created. You will have to create your joint table in the **AstorRules.mdf** database on your own, if you would like to use it.

Joint table columns must be in order and with the same name as the members from (InField/OutField).

In the Query method, after adding the input objects, you can set parameters for the joint (load from joint table or initialize with some values).

|  |
| --- |
| IOdbcTable odbcTable = (IOdbcTable)(new DSCODBCCOMLib.OdbcTable());  odbcTable.SetCurrent("RULE\_CreatePlate"); // table name  odbcTable.AddSearchCriteria(1, "Default");  //Search if joint table contain a record with "Default" name. If succeeded load values from that record.  int key = (int)odbcTable.Search();  if (-1 != key)  {  int idx = 2;  m\_dPlateThickness = (double)odbcTable.GetAt(idx++);  //…  }  else  {  m\_dPlateThickness = 10;  } |

## Developing a joint

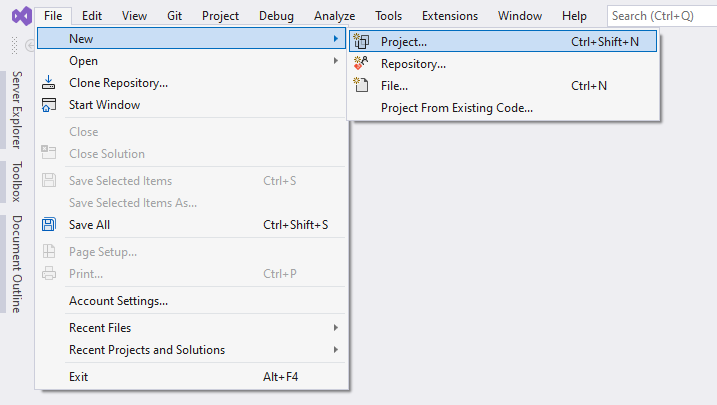
First, please make sure you at least have **.NET desktop development** and **Desktop development with C++** workloads installed..

Graphical user interface, text, application, email

Description automatically generated

### Visual Studio solution

After starting Visual Studio, you need to create a new project (**File / New / Project**).



The dialog below will appear. Choose **“Class Library (A project for creating a class library that targets .NET or .NET Standard)”**

A screenshot of a computer program

Description automatically generated

Project configuration should target .NET 8.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Make sure to change the newly created project configuration to target x64.

A screenshot of a computer program

Description automatically generated

From the Project menu choose Add Reference. In the Add Reference dialog box, click the Browse tab. Locate the folder where Steel Connections is installed (***C:\Program Files\Autodesk\Revit 2025\AddIns\SteelConnections*** or the custom path where you installed Revit).

Select the following files:

* Interop.DSCGEOMCOMLib.dll
* Interop.DSCODBCCOMLib.dll
* Interop.DSCPROFILESACCESSCOMLib.dll
* Interop.AstSTEELAUTOMATIONLib5.dll
* Interop.DSCUTILFACETCOMLib.dll

Each of these libraries has a well-defined role in the joint development process and will be described in the following brief descriptions.

**AstSteelAutomation** handles all the “core” Steel Connections functionalities. It exposes interfaces to objects that can be created with Steel Connections and also handles joint, GUI and other functionalities.

**AstControls** is used mainly for GUI development of joints. Several ActiveX controls are provided with this library. For example, the joint developer can link one static prompt to a CrtlDb message through a StaticDbTextControl.

**DSCUtilFacetCom** mainly handles the body interface. It provides access to body intersection with lines and other basic geometry interfaces.

**DSCGeomCom** handles basic geometry classes. This library can be used without Steel Connections. It exposes interfaces like point, vector, plane etc.

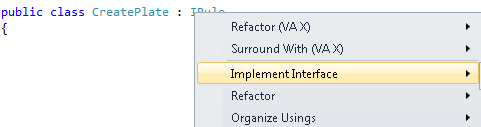
**DSCOdbcCom** library provides useful and easy access to Steel Connections databases. You can get prompts, defaults or easily access tables through interfaces provided by this library.

**DSCProfilesAccesCom** library is used to handle Proftype objects (beam profiles, user

defined and general use profiles).

At this stage, the project should have only one class (default named “Class1”). This can be renamed to fit the name of the joint. Your class should inherit from the IRule interface and IJointInfo.

You need to implement several methods/properties. To do that, right click on IRule and select Implement Interface.



**Query method**

Is the function that is passed only once when the joint is run the first time. Its role is to ask the user for input objects and set defaults.

|  |
| --- |
| public void Query(AstUI pAstUI)  {  //Filter to select only straight beam  IClassFilter classFilter;  classFilter = pAstUI.GetClassFilter();  classFilter.AppendAcceptedClass(eClassType.kBeamStraightClass);  //Declare the input objects  AstObjectsArr inputObjectsArr = m\_Joint.CreateObjectsArray();  eUIErrorCodes errCode;  IAstObject selectedObject = pAstUI.AcquireSingleObject(163, out errCode);  //selection incorrect  if (errCode == eUIErrorCodes.kUIError)  return;  //user abort the selection  if (errCode == eUIErrorCodes.kUICancel)  return;  //add selected object to the input objects array  if (selectedObject != null)  inputObjectsArr.Add(selectedObject);  //add all the objects selected by the user (input objects)  m\_Joint.InputObjects = inputObjectsArr;  //load default parameters for joint  loadDefaultValues();  } |

**CreateObjects** contains the joint functionality. It uses the global variables declared in the declaration section and does the main work. The normal behavior when creating a new joint is to create objects in the "CreateObjects()" method and show the property pages. Remember - when you show property pages you should not force the update of the joint.

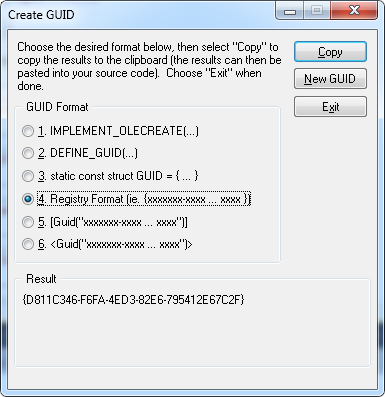
For the correct functionality of a joint when a problem has appeared and the box should be red, the joint should implement the CreateObjects method as follows:

|  |
| --- |
| public void CreateObjects()  {  bool bCreationStatus = true;  AstObjectsArr createdObjectsArr = m\_Joint.CreateObjectsArray();  try  {  //…creation stuff…..  }  catch (COMException ex)  {  System.Diagnostics.Debug.WriteLine(ex.Message);  bCreationStatus = false;  }  catch (System.Exception ex)  {  System.Diagnostics.Debug.WriteLine(ex.Message);  bCreationStatus = false;  }  //Set it to false will result in a joint "red box" situation. The  //objects created by the joint until the error occurred will be deleted and the old  objects will be available.  m\_Joint.CreationStatus = bCreationStatus;  m\_Joint.CreatedObjects = createdObjectsArr;  } |

### Create GUID

From the Visual Studio menu, select (**Tools/Create GUID**). Select Registry Format and click Copy. Then add the following lines to your project before the class name.

|  |
| --- |
| [ComVisible(true)]  [Guid("<paste generated GUID here>")] |



### Build the program

After completing the code, you must build the project. From the Build menu, click Build Solution.

### Add records in database

Go to ***C:\ProgramData\Autodesk\Steel Connections 2025\en-US*** **(or your currently installed country folder)** and open the **AstorRules.mdf** database.

**Note**: The Steel Connections databases are delivered archived when installing Revit. Please make sure to start Revit once and press the Connection Settings button to make sure the databases are unarchived in the installation folder. After this step, all MDF databases should be in that folder. There should also be some LDF files created automatically.

Open the *HRLDefinition* table. Here is the table structure:

* **Key**
* **RuleRunName** - The name that appears on the joint property sheet.
* **InternalName** – The name used to call the joint.
* **Category** - Joint category.
* **Dll** - Key to RulesDllSigned table.
* **ClassID** - Generated GUID.

Add a new record for your joint:



Open *RulesDllSigned* table. Here is the table structure:

* **Key** - number from *HRLDefinition*.Dll
* **FileName** - DLL name.
* **Tech** – Possible values are:
  + 0 – for *C++ COM*
  + 1 – for *COM in .NET*
  + 2 – for *pure .NET*

Add a new record with your joint:



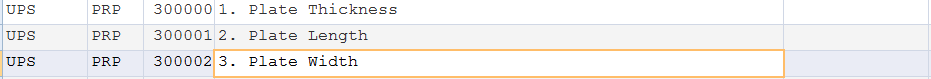
Save, close, and detach the database.

Prompts are stored in **AstCrtlDb.mdf** database.

Go to *PromptSetDefinition* table and set your prompt range first.

Description: C:\Users\Costy\Desktop\Capture.PNG

Go to *ErrorMessages*tableand add your prompts using the keys inside the range added above:



### User interface

The joint with its attributes is controlled by a property sheet dialog box. This means that this dialog box will appear during modification.

The rule designer must design all pages. This is done by implementing the *GetUserPages* method of the *IRule*.

*GetUserPages* provides the link between the C# Library class and the GUI form*s*. The joint developer is responsible for creating GUI forms for the joint. Each form represents one property sheet in the Joint Properties tab.

**InField** should contain persistent data reading from the model. In the InField function there should be calls to *pFiler.readItem* like:

m\_dPlateThickness = (double) pFiler.readItem("PlateThickness");

Please note that the “**PlateThickness**” string is important and should be exactly the same in the reverse operation.

**Outfield**: pFiler.writeItem(m\_dPlateThickness, "PlateThickness")

### Create first page

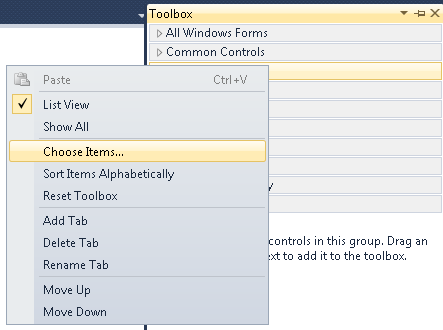
Add a new Windows Form to your class library. After that you need to set some properties to the form:

* **ShowIcon**: *False*
* **ShowInTaskBar**: *False*
* **FormBorderStyle**: *None*
* Override the **CreateParams** property.

|  |
| --- |
| private const int WS\_CHILD = 0x40000000;  protected override CreateParams CreateParams  {  get  {  CreateParams cp = base.CreateParams;  cp.Style = cp.Style | WS\_CHILD;  return cp;  }  } |

### Add an ActiveX control

Right-click on Toolbox and select Choose Items.

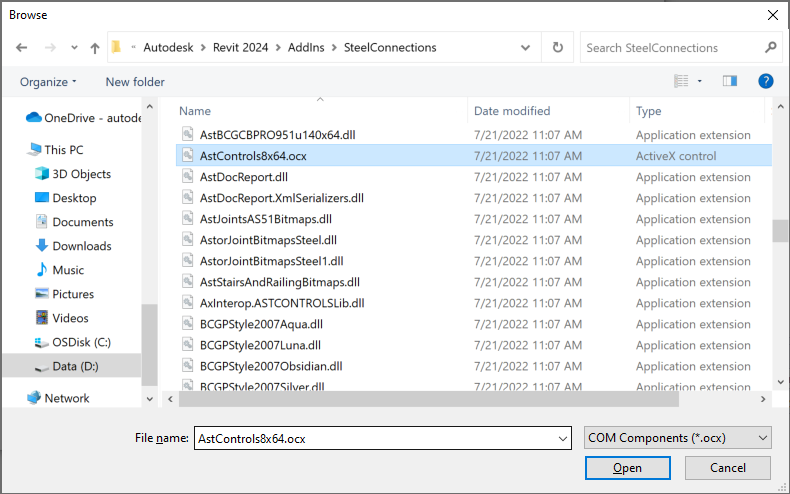


* On the **COM Components** tab, select browse (\*.ocx files).
* Go to your Steel Connections installation folder (default - ***C:\Program Files\Autodesk\Revit 2025\AddIns\SteelConnections***).
* Select AstControls8x64.ocx

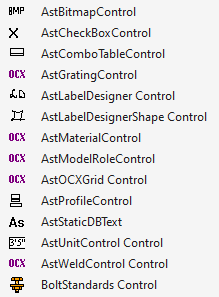
**Note**: Make sure the project’s active solution platform is set to **x64** and **NOT Any CPU**. AstControls8x64 is compiled as a 64-bit ActiveX library and NOT compatible with the **Any CPU** target. The Visual Studio 2022 form designer is also a 64 bit process. Otherwise, the following error will appear when adding Steel Connections controls to your form.

Graphical user interface, text, application

Description automatically generated



* The controls pictured in the next image should be added to the palette:



You should first register the COM manually by opening the command window with administrator rights (it should point to the ***C:\Windows\system32*** path) and call **regsvr32** with the full path to the **AstControls8x64.ocx** from your Steel Connections installation folder.

Adding the OCX control in the Visual Studio toolbox requires elevated permissions (run Visual Studio as administrator) so the COM will be registered in the system.

To add an ActiveX control, simply drag and drop it in the Visual Studio Form Designer.

* This will add under the projects dependencies two new COM references (under the COM category – AxInterop.ASTCONTROLSLib & Interop. ASTCONTROLSLib). Before compiling, these need to be removed and AxInterop.ASTCONTROLSLib.dll has to added to the “Dependencies”.
* Adding the AxInterop.ASTCONTROLSLib.dll to the dependencies category will prevent any other new ActiveX control from being dragged and dropped in the Forms Designer. To add new controls, this must be removed again, and before compiling, the step above needs to be performed again.

### Control Properties

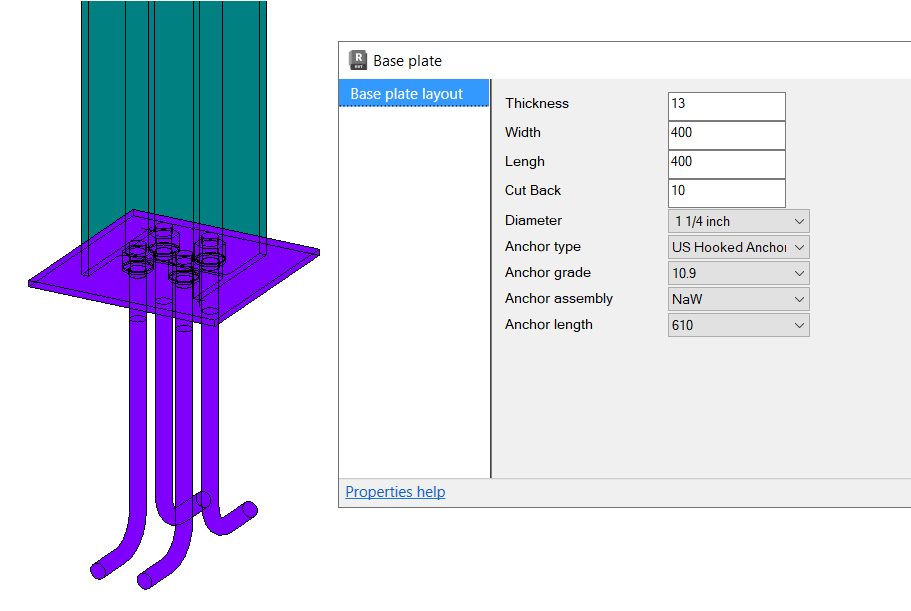
* **AstUnitControl**
* **EditType**: Length, Angle, Area, Weight, Volume, Double Adimensional, Integer Adimensional and String.
* **LabelDbkey**: UPS Prompt number from **AstCrtlDb.mdf** database (*ErrorMessages* table).
* **LabelLength**: Control length of the edit box label.
* **Visible**: True/False.
* **IntegerValue**: Control value (use ONLY for integer type controls, as set from **EditType**).
* **DoubleValue**:Control value (use ONLY for double type controls, as set from **EditType**).
* **AstComboTableControl**
* **LabelKey**: Prompt number from **AstCrtlDb.mdf** database(*ErrorMessages* table).
* **LabelLength**: Control length of the combo box label.
* **TableName**: Link to the table name from **AstorRules.mdf** database.
* **ComboIndex**: Order of runname in list, starting from 0 - not the key of the table record
* **ComboCaption**: Displayed runname. Must be a valid value from the list.
* **StringLey:** Key of record for current runname - use ONLY when the table has a string key.
* **LongKey:** Key of record for current runname - use ONLY when the table has an integer key.
* **AstBitmapControl**
* **Key**: **AstorBitmaps.mdf** database. *BitmapIndex* table.
* **AstCheckBoxControl**
* **CaptionKey:** UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **Value**: Control value. True/False, if the checkbox is checked or not.
* **Enable**: True/False.
* **AstBoltStandards**
* **BoltStandard**: Bolt type, valid key from database table (depends on **ElementType**).
* **BoltMaterial**: Bolt material, valid key from database table (depends on **ElementType**).
* **BoltSet**: Bolt assembly, valid key from database table (depends on **ElementType**).
* **BoltDiameter**: Bolt diameter, valid key from database table (depends on **ElementType**).

Which tables from either **AstorBase.mdf** or **AstorRules.mdf** are used for the Standard, Material, Set and Diameter depend on the connection being worked on.

* **LabelStandard**: UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **LabelMaterial**: UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **LabelBoltSet**: UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **LabelDiameter**: UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **ElementType**: Bolt, Shear stud or Anchor.
* **LabelLength**: Control length of the combo box labels.
* **AstProfileControl**
* **CaptionClass**: UPS Prompt number from **AstCrtlDb.mdf** database(table *ErrorMessages*).
* **CaptionShowHideAllSection**: UPS Prompt number from **AstCrtlDb.mdf** database (*ErrorMessages* table).
* **CaptionTyp**: UPS Prompt number from **AstCrtlDb.mdf** database(ErrorMessages table).
* **CurrentClass**: Value from **AstorProfiles.mdf** database, *ProfileMasterTable* table, TypeNameText column.
* **CurrentSection**: From the definition table for ClassName, value of column SectionName.
* **AstWeldControl**
* **LabelDbKey**: Prompt number from **AstCrtlDb.mdf** database (*ErrorMessages* table).
* **LabelLength**: Control length of the combo box label.
* **SelectItemByKey**: Weld type, selected by key from the weld tables in **AstCrtlDb.mdf** or **AstorRules.mdf** (contextual, depending on the connection).

### Run joint

* Compile your project.
* Copy your output .dll file to ***C:\Program Files\Autodesk\Revit 2025\AddIns\SteelConnections*** or in your custom installation folder under Addins\SteelConnections folder from Revit.
* Run Revit.
* Create a connection and select your new one.

****

### Making the joint available in Revit

To make the user defined connection available in Revit:

* Set the project references to the required Revit installed version (e.g.: %Program Files%\Autodesk\Revit 2025\AddIns\SteelConnections – if the connection should load into Revit 2025).
  + Tip: A “.props” file similar with the StructuralConnectionsSDKSamples.Common.props provided in this documentation can be used for this.
* Compile the project
* Create and add the required configuration file about the new joint.
  + This is a .xml file that should be created under a folder named “ThirdPartySettings”.
  + The “ThirdPartySettingsFolder” can be placed in one of the following locations:
    - in the Steel Connections data path plugin installation, next to the SteelConnectionsData.xml (e.g. %ProgramData%\Autodesk\Revit Steel ConnectionsVersion\lang\ThirdPartySettings\CompanyName\UserCreatedCustomConnection.xml).
    - in the AppData folder for the Steel Connections Add-in (%Appdata%\Autodesk\Revit\Autodesk Revit Version\SteelConnections\ThirdPartySettings\Company\CustomConnection.xml).
  + Multiple .xml files with different names could be added in “ThirdPartySettings” folder
  + Subfolders can be created under “ThirdPartySettings” directory with .xml files in them.
  + If the folder doesn’t exist, it should be created.

**.xml file syntax:**

**Tips:**

* The content structure should be like the SteelConnectionsData.xml from the %ProgramData%\Autodesk\Revit Steel Connections 2025\lang\ path.
* There is a sample SteelConnectionsSampleJoints.xml file provided with this documentation.

<SteelConnectionsPath> the path to the binary and resource files for the user defined connection

- Note: if this field is missing, Revit will search for the binaries only in the “SteelConnections” folder from the installation location.

<ResourceDll> - the name of the resource dll file that contains the images (Both the image and preview image should be contained in resource .dll files). The resources .dll filenames should be separated with a comma (“,”).

Example:



<TypeId> - contains the guid from the “classId” column from the **AstorRules**.*HRLDefinition* table.

<PreviewText> - contains the tooltip for the connection.

<Images> - the name of the image file to be displayed under various dialogs in Revit, found in the .dll file containing the resource image

<PreviewImages> – the name of the preview image for the connection that should be found in the .dll file containing the resource image

Example :



*-* for joints requiring additional input points (e.g., stiffener connection) you must create the

*InputPointsInfo* field inside the joint definition. Under this field you must define an *InputPointInfo* field, which will contain the necessary information for the point definition like:

- *PointSelectionText*  - This string is a Revit tip message shown to the users to guide them through joint creation.

- *InputPointRestriction –* in this fieldyou can define a restriction for the input point. Currently only restrictions of type InputMemberAxis can be defined, which will restrict the movement of your input point only to the xAxis of any of your input members. You can select to which input member to restrict your point by using the *InputMemberIndex* field*.*

Example:

**In Revit, joints have a default symbolic representation in coarse/medium view level of detail.

- to disable the the symbolic representation of a joint in coarse detail level, add the following joint field in xml configuration file of the joint:



- to disable the symbolic representation of a joint in medium detail level, add the following joint field in the xml:



* update the **AstorRules**.*AutoFilteringConfig* table with information about the new joint.

Example:

|  |  |
| --- | --- |
| *Key* | *999999* |
| *Category* | *SampleJoint* |
| *RunName* | *ColOrRaf Any to ColOrRaf Any* |
| *InputSet* | *Any+Any* |
| *InputSetConds* | *No Condition* |
| *RuleInternalName* | *SampleJoint* |
| *ObjectsOrderForJoints* | *2 Beams inversed* |
| *OwnerText* |  |

For more information about configuring the **AstorRules**.*AutoFilteringConfig* please check the following document: "**AstorRules** set up for SteelConnection Project.docx"

* Joint Installation
* The joint dll, .NET resources .dll and all the database content and the .xml files updated for all the languages. Currently the Steel Connections addin is installed in many languages, so the databases and SteelConnectionsData.xml for each country must be updated accordingly.
* The path to the SteelConnections binaries should be composed from the Revit install location read from Registry Keys (e.g. ***HKEY\_LOCAL\_MACHINE\SOFTWARE\Autodesk\Revit  
  \2025\REVIT-05:0409\InstallationLocation***) and the relative path to the Revit addins, “AddIns\SteelConnections”. The result should be something like this: “***C:\Program Files\Autodesk\Revit\2025\AddIns\SteelConnections***”.
* The SteelConnections addin stores its data in a dedicated folder specified by the “DataPath” key from ASSettings\_Advance.xml configuration file. The data, meaning the databases, are located usually under ***C:\ProgramData\Autodesk\Revit Steel Connections 2025\en-US***. The “DataPath” does not include the language subfolder, that is dynamically appended on start up. SteelConnections data (including databases) is archived in a .zip file and it is automatically unzipped on the first usage of the Steel Connections functionality in Revit.
* Database modifications for the new joint could be also added automatically on Steel Connections start-up by using the third-party project database patching mechanism (see DatabasePatching.docx).

**Joint input objects**

The Steel Connections joints currently work on the Revit structural families (framings, columns).

The structural steel columns and framing elements need to meet a series of requirements to be used in the steel fabrication workflow - see [*Supported Structural Steel Shapes and Families for Steel Fabrication*](http://help.autodesk.com/view/RVT/2024/ENU/?guid=GUID-6244E741-5D14-4DAD-AE25-5069F71B69F3).

**Special steps required to add a joint design module in Revit**

The .dll calculation file must be added to the Revit installation, under the AddIns\SteelConnections folder and necessary database configurations need to be done – more details in the [*Attach design module to joint*](#_Attach_design_module)later chapter.

**Steel Connections with Ranges of Applicability**

It is possible to associate steel connections with profile sizes by using ranges of applicability attached to the connection types. This allows you to automate the creation of structural steel connections relying on rule-based solutions.

An example implementation can be found as part of the Revit SDK sample *SampleCommandsSteelElements*. See the *AddRangesToConnectionType.cs* source code file for details.

For additional information on ranges of applicability in Revit, see [*Library-Based Connection Design Automation*](https://help.autodesk.com/view/RVT/2023/ENU/?guid=RevitDynamoSteel_Library_Based_Connection_Design_Automation_html).

# Chapter 5 Joints Design API

### Implementation of a joint calculation module

We will use the following terms:

* joint calculation module - the module which implements the interface for communication between Steel Connections and the program used to calculate the steel structure.
* Calculation engine - the external calculation module.

### How it works

A joint calculation module must be able to perform the following operations:

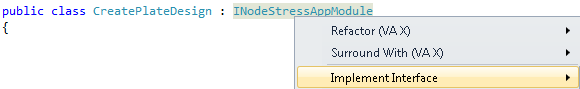
* accept input data.
* process input data.
* display results of input processing.

Description of the above operations:

* The calculation module must be able to read joint data:
* **Joint parameters** - values that define the configuration of the joint and are important to the calculation engine used (e.g., thickness of a plate, number of bolts, position of bolts, dimensions of a plate, etc.)
* **Efforts** - the values should be read from Steel Connections nodeobjects defined in the proximity of the joint, but also the efforts / loadings can be entered and stored as persistent data in the calculation module itself.
* Processing the read data: The data processing (with an external calculation engine) must support the following actions:
* **Check joint** - The external calculation engine must receive all important the information from the joint and then only check if the joint configuration is correct.
* **Presize joint** - The external calculation engine receives the required data from the joint and supplies the “best configuration” for the joint. Then, the joint calculation module must set the modified parameters back into the joint.
* **Export** -The current joint configuration in the format required by the external calculation engine.
* **Import** -The joint configuration from a previously created file by the external calculation engine.
* Displaying the results: Ideally, the external calculation engine must be able to produce two types of reports (after a “check” or a “presize” operation):
* **Quick report** - In TXT format (which will be displayed in the “Code Checking” page) and which contains general information.
* **Detailed report** - which should be in HTML/RTF format and should contain detailed information about the calculation results.

### Communication interface

The joint calculation module must implement the **INodeStressAppModule** interface**.**



### Description of interface methods

We will describe below the main methods which need to be implemented for a calculation module and the type of actions which should be performed when each of them is called by Steel Connections.

* ModuleName - You need to return from this method the internal name of the calculation module, as it was specified in the database during configuration.
* Standard - Currently, from the get method you need to return “Default”.
* GetExportName() -This method is called to obtain the string required to complete the text displayed in the “Code Checking” page on the button “Report”.
* GetImportName() - Get the calculation module name
* GetLastReportFilename(bool bQuickReportFilename) - On the “Code Checking” page, after a check or presize operation, reports should be available in order to see the results of the calculation. This method must return the full path of the last created quick report and detailed report files. Depending on the values of the input parameter “QuickReportFilename” you must return true (the quick report file path) or false (the detailed report file path).
* CheckJoint() - This method is called in two situations:
* When you press the “Check” button from “Code Checking” page.

After the check operation is finished, you need to return the joint status, depending on the results returned by the calculation engine (the joint is correctly designed, incorrectly designed or not possible to calculate).

* PresizeJoint() - This method is called when the “Presize” button is pressed. You need to pass required parameters to the calculation engine, read back the results and set the new configuration of the joint, as well as returning the new status of the joint (correctly designed, incorrectly designed or impossible to calculate).
* ImportJoint()/ExportJoint () – These functionalities are not accessible from the Revit UI. In case of export: you need to save the current joint configuration in a format which can be read by the calculation engine. In case of import: you need to set the joint parameters to the corresponding values, depending on the imported data, which has been created previously by the calculation engine. You must display the “Save As…” or “Open…” dialogs when the methods are called.
* GetUserPages(RulePageArray pagesRet, PropertySheetData pPropSheetData) - Called when the “Forces” button is pressed – you need to prepare the dialogs which display the efforts/loadings from the Node (if it exists) and also allow the creation of new sets of efforts/loadings.
* GetUserTorsorsPage(RulePage pageRet) - Called when the “Code Checking” page is displayed. You need to prepare the display of the “torsor” values on the “Code Checking” page.
* GetUserSettingsPages(RulePageArray pagesSettingsRet, PropertySheetData pPropSheetData) - Called when the “Settings” button is pressed. You need to prepare the dialogs which allow the setting of other data required by the calculation engine (other than efforts and joint parameters) - like detailed report format (HTML, RTF …), reports creation (only in certain conditions, like calculation failed), etc…

In addition to the implementation of the INodeStressAppModule interface, the joint calculation module requires the use of an object of IStressModuleJoint type.

Methods and properties available in IStressModuleJoint interface:

* InputObjects - obtain the array of input objects for the joint.
* CreatedObjects - obtain the array of objects created by the joint.
* CreateObjectsArray - create an array for storing objects.
* UseDetailedTorsors - check if all efforts/loadings should be used in calculation, or only the maximum torsor is to be considered.
* Units - get current Steel Connections units.
* JointName - identify the internal name of the joint currently being calculated.
* GetJointParam - obtain the specified parameter value from the joint (requires the use of a string that identifies the parameter - must be the same string used in the joint OutField/InField methods).
* SetJointParam - set the specified joint parameter value - you need to use this after a presize or import operation (requires the use of a string that identifies the parameter - must be the same string used in the joint OutField/InField methods).
* GetJointNSAModuleSpecificData - retrieve the value of a persistent parameter set by the calculation module (similar to the joint InField functionality).
* SetJointNSAModuleSpecificData - set the value of a persistent parameter for the calculation module (similar to the joint OutField functionality).
* GetNodeInputObjectIds - retrieve the IDs assigned to the specified input object (one or two IDs) - when a Node is defined.
* GetNodeEffortSetsLengthAtId - retrieve the number of effort sets for the specified ID - when a Node is defined.
* GetNodeEffortSetAtId - retrieve, from the specified ID, the effort set with the specified index - when a Node is defined.
* GetJointNSAModuleLoadingCases - retrieve the efforts/loadings defined in the calculation module.
* SetJointNSAModuleLoadingCases - save the new set of efforts/loadings for the calculation module.
* CreateJointEffort - create a new object for storing one effort.
* CreateJointEfforts - create an object for storing efforts.
* CreateJointLoadingCase - create one loading object.
* CreateJointLoadingCases - create an array of loadings.
* GetModuleOutputFileName - get the name of the file for saving current joint data (full path, supplied by Steel Connections - contains joint name and joint ID).
* GetJointUserId - get the ID of this joint (unique in the model).
* CreateRulePage - create an IRulePage object (for dialogs).

### Attach design module to a joint

To attach a joint calculation module to a joint, you need to create records for the calculation module in the **AstorJointsCalculation** database.

**JointInternalName** - Key to **AstorRules**.*HRLDefinition* table.

**NSAModuleInternalName** - Key to **AstorJointsCalculation**.*NSAModuleDefinition.*

**DefaultModule** – 1

**ClassID** – the GUID value from “CreatePlateDesign”, your design class.

* **Tech** – Possible values are: 0 – for C++ COM, 1 – for COM in .NET, 2 – for pure .NET.

#### Relationships

**AstorJointsCalculation**.*NSAModule* table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key** | **StressAnalysisCodeID** | **JointInternalName** | **NSAModuleInternalName** | **DefaultModule** | **SortOrder** |
| 1 | 1 | CreatePlate | CreatePlateNSAModule | 1 | 1 |

**AstorJointsCalculation**.*NSAModuleDefinition* table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Key** | **ModuleRunName** | **InternalName** | **NSAModuleDll** | **ClassID** |
| 1 | My module | CreatePlateNSAModule | 12 | {C64ECF63-D01C-451F-B0E0-E3F85DFAC05E} |

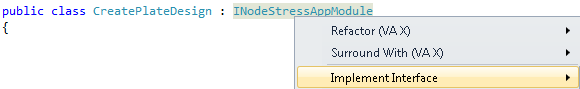
|  |
| --- |
|  |

**AstorJointsCalculation**.*NSAModuleDllSigned* table

|  |  |  |  |
| --- | --- | --- | --- |
| **Key** | **FileName** | **Tech** | **Signature** |
| 12 | SampleDesign.dll | 1 |  |

## Developing a joint design

* Use the same .bat file from developing joint section.
* Create a new Project.
* Choose Class Library.
* Add References:
* Interop.AstSteelAutomationLib5
* Interop.DSCGEOMCOMLib
* Interop.DSCRootsCOMLib
* DotNetRoots
* Implement the **INodeStressAppModule** interface.



**CheckJoint method** contains the joint design functionality.

|  |
| --- |
| public eJointStatus CheckJoint()  {  eJointStatus ret = eJointStatus.JointSupported;  //report format  int nReportFormat = (int)getJointNSAModuleSpecificData(m\_joint, this, "ReportFormat", 0);  //get efforts  List<double> dM, dN, dV;  List<string> sCaseName;  getEfforts(m\_joint, m\_joint.GetJointNSAModuleLoadingCases(this),  m\_joint.UseDetailedTorsors, out dM, out dN, out dV, out sCaseName);  //read some values from joint  bool bModifiable = false;  double dPlateThickness = (double)m\_joint.GetJointParam("PlateThickness", ref bModifiable);  double dPlateLength = (double)m\_joint.GetJointParam("PlateLength", ref bModifiable);  double dPlateWidth = (double)m\_joint.GetJointParam("PlateWidth", ref bModifiable);  return ret;  } |

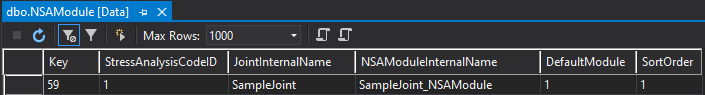
* Generate GUID.

|  |
| --- |
| [ComVisible(true)]  [Guid("C64ECF63-D01C-451F-B0E0-E3F85DFAC05E")] |

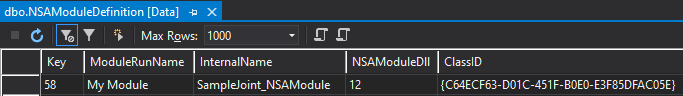
* After completing the code, you must build the project. From the Build menu, click Build Solution.
* Go to ***C:\ProgramData\Autodesk\Revit Steel Connections 2025\en-US*** (or your currently installed country folder) and open **AstorJointsCalculation.mdf** database.

**Note**: The Steel Connections databases are delivered archived when installing Revit. Please make sure to start Revit once and press the Connection Settings button to make sure the databases are unarchived in the installation folder. After this step all MDF databases should be in that folder. There should also be some LDF files created automatically.

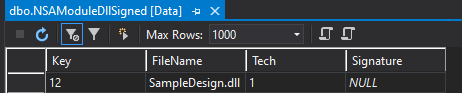
* Open *NSAModule* table and add a new record:



* Open *NSAModuleDefinition* table and add a new record:



* Open *NSAModuleDllSigned* table and add a new record:



* Save, close, and detach **AstorJointsCalculation** database.
* Compile your project.
* Copy output .dll in your Revit installation under **AddIns\SteelConnections** path (ex – default installation ***C:\Program Files\Autodesk\Revit 2025\AddIns\SteelConnections***).
* Run Steel Connections.
* Call joint. Now you should see a new “Code Checking” page.

