**Reinforce Your Design:   
Refit® Structure API for Rebar and Detailing**

Jeremy Tammik – Autodesk Inc.

**DE205-1** An important part of the Refit Structure workflow is the production of high-quality construction documents. Use the Refit Structure API to automate parts of the reinforcement workflow, especially repetitive manual tasks. Generate the complete 3D rebar model using the API, either from add-on analysis and design software or from built-in macros. Automatically generate and extract rebar information; drive the framing modelling; generate section and drafting views; import and export .DWG files such as schedules; and add text, dimensioning, and annotations. Completely control the Refit Structure 3D rebar model including single bars, sets of bars with rules, area reinforcement for slabs and walls, and path reinforcement for slabs, and provide advantages such as quantities, complex layout, automatic reinforcement drawings, and country-specific scheduling.

**About the Speaker:**Jeremy joined Autodesk in 1988 as the technology evangelist responsible for European developer support. In this capacity, he wrote articles, consulted, lectured on AutoCAD application programming techniques, and supported AutoCAD application developers in Europe, the U.S., Australia, and South Africa. He was a cofounder of ADGE (AutoCAD Developer Group Europe) and a prolific author on AutoCAD application development. He left Autodesk in 1994 to work as an HVAC application developer, and then rejoined the company in 2005 to work in the AEC group of the Autodesk Developer Network Technical Services Team.

# RST API for Rebar and Detailing

**Jeremy Tammik, Autodesk Inc., DevTech EMEA**

[1. Introduction 3](#_Toc182040497)

[2. Rebar 5](#_Toc182040498)

[3. Detailing 7](#_Toc182040499)

## Introduction

**Content:** An important part of the Revit Structure or RST workflow is the production of high-quality construction documents. Use the RST API to automate parts of the reinforcement workflow, especially repetitive manual tasks. Generate the complete 3D rebar model using the API, either from add-on analysis and design software or from built-in macros. Automatically generate rebar and extract rebar information, drive the framing modelling, generate section and drafting views, import and export DWG files such as schedules, and add text, dimensioning and annotations. Completely control the RST 3D rebar model including single bars, sets of bars with rules, area reinforcement for slabs and walls, path reinforcement for slabs, providing advantages such as quantities, complex layout, automatic reinforcement drawings and country-specific scheduling.

**Target Audience, Who Should Attend:** Application developers and Revit users interested in Revit Structure programming, reinforcement, and automating production of high-quality construction documents, with intermediate-level .NET and general Revit product knowledge.

**Session Objectives:** After attending this session, you will be able to:

* Implement a Revit Structure add-in to automate repetitive manual tasks in production of high-quality construction documents
* Use the 3D rebar model to generate country-specific DWG schedules and import them back as a drafting view in RST
* Automatically drive the framing modelling, generate rebar and extract rebar information
* Programmatically generate section and drafting views and import and export DWG files such as schedules
* Create RST macros to add text, dimensioning and annotations

**Companion Presentations:** Please also refer to the following companion presentations, which expand on the topics discussed here:

* DE105-1 The ABCs of Programming in Revit
* DE305-1 A Closer Look at the Database with the Revit API
* DE111-1 The Revit SDK Sample Smörgåsbord
* DE115-1 Ten Steps for Enhancing Your Revit® Add-In
* DE201-1 Create Your Own Bidirectional Revit Structure Stress Analysis Integration Link
* DE315-1 Answers to Your API Questions: Experts Talk About Revit and AutoCAD Architecture & MEP APIs

**Materials:** This presentation is accompanied by some enhanced Revit SDK samples.

### Agenda

* Overview

Revit Structure Rebar Workflow

* Rebar

Generate and extract rebar and its information

* Detailing

Generate views and sheets

Import and export dwg

Add text, dimensioning and annotations

Sample and API feature walkthrough in the rebar workflow context.

### Revit Structure Rebar Workflow

Working with reinforcements in Revit Structure involves two phases. The initial phase is the building analysis, which may result in design changes. In the second phase, reinforcements are added, analysed, and detailed. Both phases may include multiple iterations. Many of the tasks required can be automated. This is where our applications can save a lot of time, effort, and eliminate sources for errors.

The central hub is the Revit Structure three-dimensional object model, which integrates the physical and analytical models used respectively for the building design and analysis. The 3D rebar model is integrated in this database. The physical model is used to generate concrete drawings including rebar specific details. The rebar model can be used to generate rebar specific details and quantity takeoffs. The analytical model is used to connect to third party analysis and design packages, which in turn can help define the rebar model.

### The 3D Rebar Model

Refit Structure has a 3D rebar model composed of

* Single bar
* Set, i.e. array of bars with rules
* Area reinforcement for slabs and walls
* Path reinforcement for slabs

The 3D rebar model can be generated

* Manually by user
* Loaded from a group that defines a cage
* Generated with the API from analysis and design software or from macros

Advantages of 3D rebar model

* Quantities
* Complex layout
* Automatic reinforcement drawings
* Basic scheduling

Scheduling is specific to each country. This provides an opportunity for third party developers to use the 3D rebar model to generate DWG schedules adapted to country and standards specific needs and import them back as a drafting view in Refit Structure.

### Rebar and Detailing Topics

Rebar

* Generate rebar
* Extract rebar information

Detailing

* Generate a section view
* Generate a drafting view
* Create a sheet
* Import and export dwg
* Add text, dimensioning and annotations

Here are some relationships between these rebar topics and the rebar workflow: after the structural framing analysis part is done, an engineer will take the values of the load and stress information for each element and design the rebar for each element. You design the size of the column, number of rebars, and number of rib-bars. Based on this information, a rebar can be generated externally. It may need modification, which requires extraction and analysis of existing information. The extracted information may also be used for estimation purposes. To show the rebar placement in a column or beam, you need a section view. The Refit SDK samples CreateSimpleAreaRein and CreateComplexAreaRein generate reinforcement elements, and can be used in conjunction with a section view demo. For structure detailing, we need a drafting view where we may want to import a dwg file. Adding text, annotations and dimensioning is further typical detailing functionality. Some of the items listed here are general, but especially useful as part of the rebar workflow for structural detailing.

## Rebar

This section discusses the generation of new rebar elements, and the analysis and extraction of existing rebar information.

### Generate Beam or Column Rebar

We can use the Refit SDK sample Reinforcement to demonstrate the creation of new beam or column reinforcement elements. It creates a bar set in a selected concrete beam or column element that does not have any reinforcement. Three kinds of beam rebar are created: top, bottom and transverse rebar. Two kinds of column rebar are created: transverse and vertical. It uses the Autodesk.Revit.Creation.Document.NewRebar() method to create rebar for the selected host. The NewRebar() method expects two parameters RebarBarType and RebarHookType. Suitable values for these two parameters can be retrieved by iterating the active document elements. The main method used is:

Rebar createdRebar = rvtDoc.Create.NewRebar(

rebarType, startHook, endHook, m\_hostObject,

ref origin, ref normal, curves, (int) startOrient, (int) endOrient );

### Reinforcement Implementation

In the implementation of the Reinforcement sample, the FrameReinMakerFactory class manages all the work, including:

* Precondition checks

Is the selected object a beam or column?

Is it made of concrete?

Does it already contain any rebar elements?

* Create a corresponding FrameReinMaker and the reinforcement rebars

It passes on the data to the host-dependant working classes BeamFramReinMaker and ColumnFramReinMaker.

The GeomUtil implements some basic geometric utility methods.

GeometrySupport is a base class for BeamGeometrySupport and ColumnGeometrySupport. It manages the solid of beam or column, the extent or sweep path of the beam or column, the beam or column director vector, lists to store the edges and points, the transformation etc.

Another helper class ParameterUtil provides utility methods find or set certain parameters.

### Generate Floor or Wall Rebar

We can use the Revit SDK sample NewPathReinforcement to demonstrate how to create a new PathReinforcement element through the API. The PathReinforcement host is a floor or a wall. The method used is:

NewPathReinforcement( PathReinforcementType, Element, CurveArray, bool );

We can use Document.Create.NewPathReinforcementType() method to create a PathReinforcementType if there is none in current document. The Element argument is the host of the PathReinforcement. The CurveArray stores its path. The Boolean value indicates on which side of the path the PathReinforcement is located.

To run the sample, draw a Structure wall or slab, select it, and run this command. Draw the path of PathReinforcement you want to create. You can click right mouse button to finish drawing. Click the “Preview” button to preview the path reinforcement to be created. Select or unselect the “Flip” check box to create the PathReinforcement on the left or right side of the path. Click the “Create” button to create the PathReinforcement. You can clean and redraw the sketch of the path using the “Clean” button. Here is the central code snippet given an array of points defining the path:

public override PathReinforcement CreatePathReinforcement(

List<Vector4> points, bool flip)

{

XYZ p1, p2; Line curve;

CurveArray curves = m\_appCreator.NewCurveArray();

for (int i = 0; i < points.Count - 1; i++)

{

p1 = new XYZ(points[i].X, points[i].Y, points[i].Z);

p2 = new XYZ(points[i + 1].X, points[i + 1].Y, points[i + 1].Z);

curve = m\_appCreator.NewLine(ref p1, ref p2, true);

curves.Append(curve);

}

PathReinforcementType pathReinforcementType = m\_docCreator

.NewPathReinforcementType();

return m\_docCreator.NewPathReinforcement( pathReinforcementType,

m\_data, curves, flip);

}

Also, note the useful vector and matrix classes included in the NewPathReinforcement module MathTools.cs.

### Extract Rebar Information

All the generic data extraction and debugging utilities such as RvtMgdDbg and BuiltInParameterChecker can be used to read parameters and other information from a rebar element.

In addition, the Revit SDK sample AreaReinParameters shows how to use the API to display and modify AreaReinforcement parameters. We have expanded the sample to list rebar elements and their parameters as well in a separate command RebarParas. The enhanced sample is included in the presentation material.

Another sample for extracting rebar information is the BarDescriptions sample. It shows how to find all BarDescriptions in the project, display their properties in a DataGridView, and export their parameter information to an external comma delimited \*.csv file. BarDescription is a property of an AreaReinforcement element, so some AreaReinforcement elements should be drawn first.

Some rebar information is accessible through built-in parameters and can be determined using BuiltInParameterChecker, some is accessible only through hard-coded localised parameter names. The AreaReinParameters sample only works with floor or wall area reinforcements, not with rebar. Not all rebar parameters are accessible as built-in parameters. We can however retrieve almost all necessary information with its instance parameters, hence the RebarParas enhancement.

## Detailing

The following topics that can be fruitfully addressed for RST detailing work using the Revit API:

* Generate a section view
* Generate a drafting view
* Create a sheet
* Import and export dwg
* Add text, dimensioning and annotations

Either these topics are covered by existing generic Revit SDK samples, or we have enhanced the existing samples to cover them.

### Generate Section View

The Revit SDK sample CreateViewSection demonstrates the creation of a detail view. Given a linear element such as a wall, floor or beam, it generates a section view across the midpoint of the element using the NewViewSection() method. It performs the following steps:

* Retrieve the selected linear element
* Generate a BoundingBoxXYZ instance to be used in NewViewSection()
* Set its Max and Min properties
* Generate a Transform instance for the BoundingBoxXYZ Transform property, which defines the origin and directions of the created view, including RightDirection, UpDirection and ViewDirection
* Create the section view using the BoundingBoxXYZ.

The NewViewSection method takes a parameter of type BoundingBoxXYZ. The key part of the code is actually to create the box, and the key part for the box is to create its transform matrix. The actual matrix generated depends on the selected element type. Separate implementations are provided for beam, floor and wall. These are called from within GenerateTransform():

/// <summary>

/// Generate a BoundingBoxXYZ instance which used in NewViewSection() method

/// </summary>

/// <returns>true if the instance can be created; otherwise, false.</returns>

Boolean GenerateBoundingBoxXYZ()

{

// First new a BoundingBoxXYZ, and set the MAX and Min property.

m\_box = new BoundingBoxXYZ();

m\_box.Enabled = true;

XYZ maxPoint = new XYZ( LENGTH, LENGTH, 0 );

XYZ minPoint = new XYZ( -LENGTH, -LENGTH, -HEIGHT );

m\_box.Max = maxPoint;

m\_box.Min = minPoint;

// Setting the Transform property is the most important thing.

// It defines the orgin and the directions (including RightDirection,

// UpDirection and ViewDirection) of the created view.

Transform transform = GenerateTransform();

if( null == transform )

{

return false;

}

m\_box.Transform = transform;

// If all went well, return true.

return true;

}

Once the box and its transformation are in place, the view can be created:

// Create a section view.

ViewSection section = m\_project.Create.NewViewSection( m\_box );

To run this sample, register the command in Revit.ini, launch Revit Structure 2008, select a beam or column with rebar, and execute the external command. Note that a new section view is added to the detail views in the project explorer.

### Generate Drafting View

Generation of a drafting view is achieved using

Document doc = commandData.Application.ActiveDocument;

ViewDrafting drafting = doc.Create.NewViewDrafting();

The NewViewDrafting() method does not take any parameters. It creates an empty drawing view. We have added a second external command CreateDraftingView to the Revit SDK sample CreateViewSection to demonstrate this.

### Import and Export DWG

Importing and exporting DWG files is demonstrated by the Revit SDK sample ImportExportDWG. It shows how to export the current project to dwg files and import a dwg file into Revit. It also demonstrates how to and provides a user interface to set up the import and export options appropriately. To run it, start up Revit, open a suitable project, and execute the command.

To export, check the radio button “Export”. In the next dialog, specify the file name to export as, set the common options and click the “Option…” button to set the lower priority options; click the “Select…” button to select multi-views to export. Then click the “Save” button to perform the export.

To import, check the radio button “Import”. In the next dialog, specify the file to import from and set the other options. Then click the “Open” button to perform the import.

The code to import a DWG file looks like this:

//parameter: DWGImportOptions

DWGImportOptions dwgImportOption = new DWGImportOptions();

dwgImportOption.ColorMode = m\_importColorMode;

dwgImportOption.CustomScale = m\_importCustomScale;

dwgImportOption.OrientToView = m\_importOrientToView;

dwgImportOption.Placement = m\_importPlacement;

dwgImportOption.ThisViewOnly = m\_importThisViewOnly;

dwgImportOption.Unit = m\_importUnit;

dwgImportOption.View = m\_importView;

dwgImportOption.VisibleLayersOnly = m\_importVisibleLayersOnly;

//parameter: Element

Element element = new Element();

//Import

Document doc = m\_commandData.Application.ActiveDocument;

bool imported = doc.Import(m\_importFileName, dwgImportOption, ref element);

The last parameter of the Import() method is an element. It is not necessary to create a new element prior calling the Import() method. You can simply pass a variable of type Element and it can be null. This provides a possibility to return the element that was just imported, so that it can be accessed and modified after the import. For example, if you import an image, the import method gives you neither scaling nor rotating options. If you would need this functionality, you can achieve it by manipulating the image after the import.

Exporting DWG files can be done like this:

//parameter : String folder

String folder = m\_exportFolder;

//parameter : String name

String name = m\_exportFileName;

if (folder == null || name == null)

{

throw new NullReferenceException();

}

//parameter : ViewSet views

ViewSet views = new ViewSet();

if (m\_currentViewOnly)

{

views.Insert(m\_commandData.Application.ActiveDocument.ActiveView);

}

else

{

views = m\_selectViewsData.SelectedViews;

}

//parameter : DWGExportOptions dwgExportOptions

DWGExportOptions dwgExportOptions = new DWGExportOptions();

dwgExportOptions.ExportingAreas = m\_exportOptionsData.ExportAreas;

dwgExportOptions.ExportOfSolids = m\_exportOptionsData.ExportSolid;

dwgExportOptions.FileVersion = m\_exportFileVersion;

dwgExportOptions.LayerMapping = m\_exportOptionsData.ExportLayerMapping;

dwgExportOptions.LineScaling = m\_exportOptionsData.ExportLineScaling;

dwgExportOptions.MergedViews = m\_exportMergeFiles;

dwgExportOptions.PropOverrides = m\_exportOptionsData.ExportLayersAndProperties;

dwgExportOptions.SharedCoords = m\_exportOptionsData.ExportCoorSystem;

dwgExportOptions.TargetUnit = m\_exportOptionsData.ExportUnit;

//Export

Document doc = m\_commandData.Application.ActiveDocument;

bool exported = doc.Export( folder, name, views, dwgExportOptions );

### Create New Dimensioning

New dimensioning can be generated through the API using the NewDimension() method. Searching globally through all the samples discovers one single instance of this method, in the CreateDimensions sample. This sample adds a command taking a selection of basic structural walls and adds dimensioning from the start to the end of each. Dimensioning is created between graphical references stored in a ReferenceArray instance. The key part is the creation of the reference array.

For the basic wall, the location line of the wall is used to discover the two vertical edges at each end of one side of the wall. Some calculation and comparison work needs to be done in order to find the two right reference lines. For each of these two curves, a reference is added to the reference array. Note that the geometry options for obtaining the wall geometry have options.ComputeReferences set to true:

ReferenceArray referenceArray = new ReferenceArray();

Options options = m\_revit.Application.Create.NewGeometryOptions();

options.**ComputeReferences** = true;

options.View = m\_revit.Application.ActiveDocument.ActiveView;

Autodesk.Revit.Geometry.Element element = **wall.get\_Geometry**(options);

GeometryObjectArray geoObjectArray = element.Objects;

//enum the geometry element

for( int j = 0; j < geoObjectArray.Size; ++j )

{

GeometryObject geoObject = geoObjectArray.get\_Item(j);

Curve curve = geoObject as Curve;

if (null != curve)

{

// find the two upright lines beside the line

if (Validate(newLine, curve as Line))

{

referenceArray.Append( **curve.Reference** );

}

if (2 == referenceArray.Size)

{

break;

}

}

}

Once the reference array has been set up, the creation of the dimensioning is simple:

XYZ p1 = newLine.get\_EndPoint(0);

p1.X += 5;

p1.Y += 5;

XYZ p2 = newLine.get\_EndPoint(1);

p2.X += 5;

p2.Y += 5;

Line newLine2 = app.Create.NewLine( ref p1, ref p2, true );

Dimension newDimension = doc.Create.NewDimension(

doc.ActiveView, newLine2, referenceArray );

To extend the sample to work with other elements than structural wall, other methods for determining the references need to be developed. An architectural wall cannot get the reference from an edge. Not all geometry can provide a reference. An alternative is to use a face instead:

Solid solid = geoObject as Solid;

if(null != solid)

{

FaceArrayIterator faceItor = solid.Faces.ForwardIterator();

while (faceItor.MoveNext())

{

PlanarFace face = faceItor.Current as PlanarFace;

if (null != face)

{

// find the two upright lines beside the line.

// we pick up the face in term of its normal.

if( Validate( newLine, face.Normal ) )

{

referenceArray.Append( face.Reference );

}

if( 2 == referenceArray.Size )

{

break;

// . . .

Another example of obtaining geometry references and creating dimensioning for a non-structural wall is discussed in part 5, 'References and Dimensioning' of RevitGeometry.cs, the Revit geometry sample code accompanying the presentation **DE115-1** Ten Steps for Enhancing Your Autodesk® Revit® Add-In.

### Rebar Dimensioning

For rebar dimensioning, we need to extend the sample further to work with rebar elements. To do so, we need to dig further into geometry elements:

// get rebar reference

if (m\_rebar != null)

{

Revit.Geometry.Element gelement = m\_rebar.get\_Geometry(options);

GeometryObjectArray geoObjectArray = gelement.Objects;

// enum the geometry element

for (int j = 0; j < geoObjectArray.Size; j++)

{

GeometryObject geoObject = geoObjectArray.get\_Item(j);

Line line = geoObject as Line;

if (line != null && line.Reference != null)

{

referenceArray.Append(line.Reference);

}

}

}

### Add Tags

The Revit API provides the method NewTag() to create a new tag. It is demonstrated by the Revit SDK sample TagBeam, which adds a tag to a beam element. We extended this sample by adding a new external command class TagRebar, which adds a tag to a rebar element as well.

if( elem is Rebar )

{

// cast to Rebar and get its first curve

Rebar rebar = elem as Rebar;

Autodesk.Revit.Geometry.Curve curve = rebar.Curves.get\_Item( 0 );

// create a rebar tag at the first end point of the first curve

IndependentTag tag = doc.Create.NewTag( view, rebar, true,

TagMode.TM\_ADDBY\_CATEGORY, TagOrientation.TAG\_HORIZONTAL,

curve.get\_EndPoint( 0 ) );

}

The NewTag() method signature and its parameters:

public IndependentTag NewTag(

View dbview,

Element objelem,

bool leader,

TagMode tagmode,

TagOrientation tagorientation,

XYZ pnt

);

Parameters

* dbview the view in which the tag is to be visible
* objelem the tag host object of tag
* leader whether it has a leader
* tagmode the tag mode: add by category, multi-category or material
* tagorn tag orientation
* pnt tag position

We used category tag mode and horizontal tag orientation here.

### Add Text

To demonstrate the addition of text, we added another external command named CreateText to the TagBeam sample. The required API method is NewTextNote(). It takes the following parameters:

public [TextNote](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Elements.TextNote.html) NewTextNote(

[View](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Elements.View.html) pView,

[XYZ](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Geometry.XYZ.html) origin,

[XYZ](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Geometry.XYZ.html) baseVec,

[XYZ](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Geometry.XYZ.html) upVec,

[double](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDoubleClassTopic.asp) textSize,

[double](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemDoubleClassTopic.asp) lineWidth,

[TextAlignFlags](mk:@MSITStore:C:\Dl\ADSK\Revit\RS%202008%20Beta1\Revit%202008%20SDK\RevitAPI%202008.chm::/Autodesk.Revit.Enums.TextAlignFlags.html) textAlign,

[string](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemStringClassTopic.asp) strText );

**Parameters**

* *pView* current view
* *origin* text origin
* *baseVec* text base vector
* *upVec* text up vector
* *textSize* text size
* *lineWidth* line width of the out rectangle
* *textAlign* text alignment
* *strText* text string

The textSize parameter of the NewTextNote method is actually not used and can be ignored. The actual size value comes from the font or family type.

Making use of this method to add a text string to a rebar element:

if( elem is Rebar )

{

// cast to Rebar and get its first curve

Rebar rebar = elem as Rebar;

Autodesk.Revit.Geometry.Curve curve = rebar.Curves.get\_Item( 0 );

// calculate necessary arguments

Autodesk.Revit.Geometry.XYZ origin = curve.get\_EndPoint( 0 );

origin.X += curve.Length \* 10; // draw the text at the right size

XYZ baseVec = new XYZ( 1, 0, 0 );

XYZ upVec = new XYZ( 0, 0, 1 );

double textSize = curve.Length / 10;

double lineWidth = curve.Length / 50;

string strText = "This is " + rebar.Category.Name + " : " + rebar.Name;

// create the text

TextNote text = doc.Create.NewTextNote( view, origin,

baseVec, upVec,

textSize, lineWidth,

TextAlignFlags.TEF\_ALIGN\_CENTER | TextAlignFlags.TEF\_ALIGN\_MIDDLE,

strText );

text.Width = curve.Length \* 10; // set the text width

}

### Revit Structure Product and SDK

Before closing, we would like to point out that Revit 2008 SP2 version is available for download from the product site

* <http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=9281007>

This version includes the Revit SDK in

* <Revit>\Download\Utilities\Common\Software Development Kit\Revit 2008 SDK.zip

The Revit 2008.2 SDK includes two new samples:

* RoomSchedule
* RDBLink

### The End

Thank you very much for your attention. I wish you much success and enjoyment with your Revit development work.

### Acknowledgement

The enhanced samples presented were implemented and tested by Bill Zhang, Developer Technical Services, Autodesk Inc. Thanks also to Nicolas Mangon, Structural Product Manager, Autodesk Inc., for fruitful input and support.