**The New Autodesk® Revit® Family API –   
Everything is Relative**

Jeremy Tammik – Autodesk Inc.

**CP9118-1** Understand and use the new Revit 2010 family API, which provides full access to programming facilities within the family editor context. Use it to automate Revit content creation based on existing part databases, and other library specifications. Programmatically create and modify family content. Extract family definitions from existing projects. Define references and constraints to parametrically drive family type model geometry, specify formulae to drive parameter values, and add annotation and dimensioning. Control detailed visibility of family types and their elements. Control family loading behaviour. Understand and reuse the functionality provided by the Revit SDK family API samples. Implement automated batch mass processing of all family definition and manipulation processes, which you were previously forced to complete individually and manually. This class assumes basic knowledge of Revit programming.

**About the Speaker:** Jeremy is a member of the AEC workgroup of the Autodesk Developer Network ADN team, providing developer support, training, conferences, and blogging on the Revit API.

He joined Autodesk in 1988 as the technology evangelist responsible for European developer support to lecture, consult, and support AutoCAD application developers in Europe, the U.S., Australia, and Africa. He was a co-founder of ADGE, the 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.

Jeremy graduated in mathematics and physics in Germany, worked as a teacher and translator, then as a C++ programmer on early GUI and multitasking projects. He is fluent in five European languages, vegetarian, has four kids, plays the flute, likes reading, travelling, theatre improvisation, and carpentry, loves mountains, oceans, sports, and especially climbing.

[jeremy.tammik@eur.autodesk.com](mailto:jeremy.tammik@eur.autodesk.com)

### Content

This document presents the new family API introduced in Revit 2010. These are the main topics covered:

Creating a Family through the User Interface 2

Creating a Family Programmatically 5

Family API Samples in the Revit SDK 6

Family API Labs: Creating an Example Family 8

### Background

The concept of families is an enormous strength of Revit, but until Revit 2010, no programming access was available in the family context. Therefore, two large and completely disjunct developer communities have evolved around the Revit products, creating either:

* Revit applications using the API
* Revit content with no API access

The Family API was the top wish list item and was made available for the first time in Revit 2010. It provides huge potential for synergy uniting the two separate camps. It enables:

* Use of the Revit API in the family editor
* Extract and modify existing or create new family content
* Automatic library generation

## Creating a Family through the User Interface

A non-trivial family can have a complex internal structure and many decisions need to be taken. A large body of experience around building families and family libraries has been developed before the introduction of the API. It is useful to gather some experience creating families manually before implementing code to do so automatically. Just like the standard Revit API, almost all the features provided by the family API are available through the user interface as well.

### What is a Revit Family?

Before discussing the family API, it is important to understand the basics of Revit families and their definition.

A Revit family is a representation of building objects and symbols. It can include geometry in 2D or 3D as well as data that supports the definition and creation of object instances. A family defines one or more types or symbols. A type or symbol can be inserted into the project to create a family instance.

There are three different classes of families: system, standard and in-place:

* System families are stored in the project template and used for objects such as Walls, Roofs, Floors, Ceilings, Rebar, etc.
* Standard families are defined externally in freestanding RFA files and used for objects such as Windows, Doors, Furniture, Beams, Ductwork, etc.
* In-Place families are used for "one of kind objects".

The new family API provided in Revit 2010 addresses the standard families.

### Revit Families - Where to Begin

A new family can be created from scratch based on a family template file, or from an existing family, which is enhanced in some way. Regardless of whether you are using the UI or API, the first thing you need to decide is which template or family file you want to begin with.

* Create a completely new family starting from a family template.
* Enhance an existing family.

There are plenty of templates to choose from. To make the most appropriate choice you should consider:

* The family category
* Is the family 2D or 3D?
* Model or detail component?
* Hosted or non-hosted: wall, ceiling, etc.
* Placement type: free, two point...
* Specialty: lighting, RPC...

### Revit Family Flavours

Just like the Revit product, Revit families also come in three flavours: for architecture, MEP (mechanical, engineering, and plumbing), and structure. Most of the functionality is common to all three, but there are some distinctions as well:

Architecture families:

* Basic building components with simplistic interactions in the model
* Free placement objects - casework, furniture, etc.
* Two point placement objects - detail components, hosted objects
* Hosted objects: windows, doors, columns ("level to level"), ceiling or "wall based" lighting fixtures

Structure families:

* Additional components with complex interactions with other objects
* Framing - beams ("beams to beam", "beam to column"), columns
* Trusses - layout for girder trusses; boundary conditions
* Span direction symbols; reinforcement symbols - area reinforcement expands to find edges, path reinforcement

MEP families:

* Connectors allowing objects to resize based on connected neighbour elements

### Revit Family Editor

Revit offers six basic family editors: 3D model, annotation, detail, rebar, truss and new conceptual mass

Of the six basic family editors, the conceptual mass creation one is new to 2010. Depending on the editor, you will see a different set of available tools and building blocks. For instance, you will see tools to create forms in the model editor, but not in the annotation one. If you are using the truss editor, you will have access to the top and bottom chord, which will be shown in the model editor.

Each family editor is tied to the chosen family template and provides a specific feature set:

* Geometry - extrusions, blends, sweeps, revolves
* Lines - model, symbolic, detail
* Basic tools - copy, mirror, paint, join/unjoin, cut geometry/don't cut
* References - reference planes, reference lines
* Annotation tools - labels
* Advanced tools - formulae, nesting, arrays, type catalogues
* MEP tools - add connectors

### Revit Families Best Practice

As said, families are a powerful feature in Revit. Creating a family can be fun, and it can also be complex. When it becomes complex, it requires good planning. Here are some suggestions for a process for manually building families by the Autodesk Revit content manager Steve Campbell. As we will see below, the same applies to a programmatic approach. A key to understanding the family API is to understand the UI.

It is highly recommended to follow this structured process when building families. It needs to be learned and practiced. Systematically following this process is one of the most important aspects of family creation.

Process order:

1. Plan (insertion point, parametric origin)
2. Lay out reference planes (the bones)
3. Add parameters
4. Add multiple host thickness types
5. Add two or more types
6. Flex types and host (testing procedure)
7. Add a single level of geometry
8. Repeat steps 6 and 7 until you are satisfied with the results
9. Test in project environment (create testing project)

### Revit Family Possibilities

You can create quite complex objects and behaviour using Revit families. Here are a few of the possibilities:

* Formulae to control behaviour, visibility, arrays
* Arrays and nesting for repeatable, resizable elements across an array
* Advanced nesting with subcomponents that can be swapped
* Reference lines and angular movement

Formulae can be used to control behaviour, visibility, arrays, e.g. to define arrays of bolts depending on the size of a plate.

Arraying nested components allows the user to create families with repeatable elements across an array that can resize based on user input or rules. For example, a bookshelf with arrayed shelves, mullion patterns based on rules, and open web joists that adjust based on length and height.

Advanced nesting can make use of nested families with family type parameters, which can provide flexible components with swappable sub-components such as nested door panels, frames, hardware, playground equipment, swappable panels and components.

Reference lines allow geometry to be rotated around an axis. They contain two endpoints and two built in work planes that can be parametrically controlled. Some simple usage examples include a door swing that can change the opening angle, or a light fixture head that moves and points in a specified direction. A more complex example is an excavator arm that can bend and rotate about three or more pivot points.

## Creating a Family Programmatically

Now that the basics of Revit families are clear, we can look at the new access to this functionality provided by the new family API.

### Family API Usage

Exposure of the family API is probably the most important enhancement to the Revit API in 2010. The concept of component family is a unique feature and strength of Revit. This was the most wanted feature in the Revit API community and we expect the effect and growth in possibilities with the availability of family API will be dramatic.

An obvious opportunity provided by the new family API is the automatic generation of content from databases or other library sources. It is also possible to extract a family definition out of a project and store it back into an external family file. The document ‘Revit Platform API Changes and Additions.doc’ in the Revit SDK folder provides an overview of the family API. Family API specific samples are located in the Revit SDK samples FamilyCreation subfolder.

Here are some of the new supported features:

* Enable use of the Revit API within the family editor context
* Create and modify family content
* Automatic library generation from database or other library specification
* Extract family definitions from existing projects
* Define references and constraints to drive model geometry parametrically, formulae to drive parameter values, and annotation and dimensioning
* Control detailed visibility of family types and their elements
* Control loading behaviour of a family

### Document and Family Manager Classes

The Revit API Document now has some added methods and properties for managing families:

* EditFamily - edit a family loaded in a project document
* FamilyCreate - return a FamilyItemCreate object to create new instances of elements within a family document, analogous to the Create object in a project
* FamilyManager - return a FamilyManager object providing access to family types and parameters
* IsFamilyDocument - identify whether the current document is a family document
* OwnerFamily - return the owning family of this family document

Within a family document, the family manager class provides the following new functionality:

* Add, remove and rename types
* Add and remove parameters
* Set parameter values and formulae

### Creating Family Content

The FamilyCreate property on the family document returns a FamilyItemFactory instance. This family item factory object is a utility object used to create new instances of elements within the family document. Just like other Revit elements, these are instantiated using dedicated methods instead of the .NET new operator. This ensures that the elements created are correctly added to and hooked up within the family document. Elements types that can be created include:

* Levels
* Alignment
* Annotation
* Curves and divided surfaces
* Connector: duct, pipe and electrical
* Dimensioning: angular, linear, radial and arc length
* Solids forms for conceptual design such as blend, extrusion and sweep

The number of supported types is expected to increase in future releases.

### Visibility Settings

A critical topic when building family content are the visibility settings. They are now accessible for each element in a family through the new FamilyElementVisibility class.

Each element in a family has its own visibility settings which define which levels of detail and which types of views it appears in. These options are critical to building good content. For example, intricate details of a family should only be visible in the fine detail views. 3D solid content could optionally be suppressed in plan views, where light weight 2D line work could be displayed instead. Such an approach can make a substantial performance difference, especially in large building models.

Every element in the family has own visibility settings managed by the FamilyElementVisibility class, which can define which levels of detail and which types of views it appears in.

### Loading Control

The Document.LoadFamily method has been enhanced and new overloads of existing methods have been added to help to handle situations such as when a family already exists in the project:

* LoadFamily(Document) - load the contents of this family document into another document.
* LoadFamily(string) - load an entire family and all its types into the document.
* LoadFamily(string, Family) - load an entire family and all its types into the document and provide a reference to the loaded family.
* LoadFamily(Document, IFamilyLoadOptions) - load the contents of this family document into another document.

The IFamilyLoadOptions argument to the last method defines an interface which specifies two call-backs for handling family load situations: OnFamilyFound and OnSharedFamilyFound. These are called when a family or a shared family is already present in the target document.

## Family API Samples in the Revit SDK

New samples illustrating the family API have been added to the Revit SDK. They are located in the FamilyCreation subfolder in the SDK Samples directory. See below a short description of these samples.

### AutoJoin

AutoJoin automatically joins geometry of multiple generic forms for use in family modelling and massing. It uses the method Document.CombineElements to join geometry between overlapping generic forms. It also includes a utility method to check geometry object overlap, based on the Face.Intersect(Curve) method.

### AutoParameter

AutoParameter implements batch mode automatic addition of shared or non-shared parameters to one or more family documents. It optionally processes either the currently active family document or all families in a specified folder. It uses the FamilyManager class AddParameter methods and reads its input data from parameter text files in a format similar to the Revit shared parameter files.

### CreateAirHandler - RME

CreateAirHandler is a Revit MEP sample to create an air handling unit including MEP pipe and duct connectors. It shows how to check the template family category to verify that a valid starting point is selected. It make use of the FamilyItemFactory class NewExtrusion, NewPipeConnector, and NewDuctConnector methods, sets up the proper connector parameters, and uses Document.CombineElements to join the extrusions to make up the air handler body.

### CreateTruss - RST

CreateTruss is a Revit Structure sample that creates a mono truss in a truss family document. The truss curves are created using NewModelCurve, the truss type is set through the ModelCurve TrussCurveType property, and constraints are added to the truss curves with NewAlignment.

### DWGFamilyCreation

DWGFamilyCreation shows how to import a DWG file into a family document add two type parameters to the imported instance: DWGFileName specifying the DWG file name, and ImportTime storing the data and time when it was imported.

### GenericModelCreation

GenericModelCreation creates a generic model using extrusion, blend, revolution, sweep and swept blend elements. It checks that the open document is indeed a family one or otherwise creates a new family document. It exercises the CreateSketchPlane, NewLineBound, and FamilyItemFactory methods to create profiles and shapes.

### TypeRegeneration

The TypeRegeneration sample uses the FamilyManager Types property to determine all types defined in the current family document, and CurrentType to iterate through them. It reports whether all types regenerated successfully, and logs any errors that occurred to a file.

### ValidateParameters

ValidateParameters checks whether every type in the current family document has valid values for certain parameters and logs the results to a file. This sample can be run in two modes, either as an external application subscribing to DocumentSaving and DocumentSavingAs events to run the check automatically every time a document is opened, or as an external command to be launched manually when required.

### WindowWizard

WindowWizard shows how to create a new window family via a wizard style user interface. It needs to be started in a window family template, e.g. Metric Window.rft. It prompts the user to define input dimensions for various window parameters and materials, and then creates the required geometry, constraints and types using elements including extrusions, alignments, dimensions, reference planes, and family types.

## Family API Labs: Creating an Example Family

The Revit Family API Labs is a collection of exercises which introduce you step by step to the creation of a column family. The objective is to learn the basics of the family API. The labs start at zero and proceed through the absolute beginning steps up to slightly more advanced aspects. Full documentation of and instructions for each step are included in separate documents for C# and VB. Here are the four steps of increasing complexity covered:

1. Define a rectangular profile column family.
2. Define an L-shaped profile column family.
3. Add formulae and materials.
4. Add visibility control.

Here is an overview of the main family lab files and directories:

* rfa\_labs.sln – common C# and VB Visual Studio solution file
* cs – C# source code directory
  + LabsCs.csproj
  + 1\_ColumnRectangle.cs
  + 2\_ColumnLshape.cs
  + 3\_ColumnFormulaMaterial.cs
  + 4\_ColumnVisibility.cs
  + Util.cs
* csdoc – C# documentation and detailed step-by-step instructions directory
  + Family Lab1 - Create Rectangular Column\_CS.rtf
  + Family Lab2 - Create L-Shape Column\_CS.rtf
  + Family Lab3 - Add Formula and Material\_CS.rtf
  + Family Lab4 - Add Visibility Control\_CS.rtf
* vb – VB source code directory
  + LabsVb.vbproj
  + 1\_ColumnRectangle.vb
  + 2\_ColumnLshape.vb
  + 3\_ColumnFormulaMaterial.vb
  + 4\_ColumnVisibility.vb
* vbdoc – VB documentation and detailed step-by-step instructions directory
  + Family Lab1 - Create Rectangular Column.rtf
  + Family Lab2 - Create L-Shape Column.rtf
  + Family Lab3 - Add Formula and Material.rtf
  + Family Lab4 - Add Visibility Control.rtf

The following summaries provide a high-level overview of the implementation. Detailed information and step-by-step instructions are provided in the source code comments and the dedicated csdoc and vbdoc documents.

### Lab 1 - Create Rectangular Column

The first lab demonstrates how to implement the following four essential basic steps for creating a new family:

1. Check the family context and category.
2. Create a simple solid using extrusion.
3. Set alignments.
4. Add types.

It makes use of the following classes and methods:

doc.IsFamilyDocument

doc.OwnerFamily.FamilyCategory.Name

doc.FamilyCreate.NewExtrusion()

doc.FamilyCreate.NewAlignment()

familyMgr = doc.FamilyManager

familyMgr.NewType()

familyMgr.Parameter(); familyMgr.Set()



Here is an overview of the four steps listed above in slightly greater detail:

1. In the first lab, we create a simple rectangular column family defining three column types with different dimensions. Before proceeding with the definition of the geometry and types, we need to ensure that we are working in a family template file and the correct family category. The document provides a property IsFamilyDocument to check the former, and the owner family category gives us the category of the template file.
2. Creating the geometry is easily achieved by defining a rectangular profile in the XY plane and extruding it using the NewExtrusion method. When creating the geometry, we need to account for the Revit database units and convert all our length measurements to feet.
3. One method to drive the geometry parametrically is to specify alignments between certain parts of the geometry and reference planes. In this very simple case, the template file already provides six reference planes for the upper and lower levels and the right, left, front and back faces, and we have six corresponding faces on the column. We need to specify alignments between these reference planes and the corresponding faces of our solid extrusion. This is done by identifying the matching pairs of solid face and reference plane and calling NewAlignment for each. In this simple case, the solid faces can be identified by their normal vectors and the reference planes by their names. In a more complex case, such as the next example in lab 2, we may have to create the reference planes ourselves, and use more complex algorithms to identify the reference planes and the solid geometry faces to associate with each other.
4. Finally, we create a few sample types for the family. In this simple case again, the template file already defines the parameters that we require for this, namely Width and Depth. Lab 2 will demonstrate how to define our own new parameters. Since the parameters are given, all we need to do is access the FamilyManager object and use its AddType and get\_Parameter methods to specify the types to create and their dimensions.

Further in-depth details on the process and source code snippets to achieve each of these four steps is given in the separate documents ‘Family Lab1 - Create Rectangular Column.rtf’ in the csdoc and vbdoc subdirectories for C# and VB, respectively.

### Lab 2 - Create L-Shaped Column

The second lab builds on the first to define a column with a slightly more complex L-shaped cross section profile.



The predefined reference planes and parameters provided by the template file are not sufficient to drive all aspects of the more complex L-shaped geometry, so we have to create our own additional ones. To be precise, we need reference planes, parameters and dimensions to determine and measure the thickness of the two ‘legs’ of the L shape.

Also, in the L-shaped column solid, the normal vector alone is not enough to identify all of the individual solid faces, so that algorithm needs to be refined as well. In our case, we use the reference plane associated with each solid face to identify it.

Finally, the lab also demonstrates adding new dimensioning to the family definition. The following steps are added to the existing ones:

* Add reference planes.
* Add parameters.
* Add dimensions.

The classes and methods used include:

doc.FamilyCreate.NewReferencePlane()

familyMgr.AddParameter()

doc.FamilyCreate.NewDimension()

Separate C# and VB versions of the detailed instructions, explanations, and source code snippets for this lab are provided by the document ‘Family Lab2 - Create L-Shape Column\_CS.rtf’ in the csdoc and vbdoc subdirectories of the sample material.

### Lab 3 - Add Formulae and Materials

In the third step, we further enhance the column family defined in the previous step by defining formulae for the L shape leg width parameters and by assigning a material to one of the family types.

* Add formulae.
* Add materials.



The classes and methods used include:

familyMgr.SetFormula()

pSolid.Parameter("Material")

familyMgr.AddParameter()

familyMgr.AssociateElementParameterToFamilyParameter()

The two parameters are defined as formulae so that the L shape leg width equals a quarter of the column width, and similarly for the depth. The key here is the family manager SetFormula method.

For the material, an instance parameter for the material group is added to the family and associated with the material parameter of the solid. Here, the key is the AssociateElementParameterToFamilyParameter method.

We assume the targeted material already exists in the family template. Currently, there is no API access to modify Render Appearance properties, which is critical for the appearance of a solid. Functionality to create a new material will probably be added in future releases. For now, please embed the desired material in the template file through the user interface. In this lab, we assign the material Glass, which is already present in the template, even though it may seem like an unrealistic material for most columns.

Again, separate C# and VB versions of the detailed instructions, explanations, and source code snippets for this lab are provided by the document ‘Family Lab3 - Add Formula and Material\_CS.rtf’ in the csdoc and vbdoc subdirectories of the sample material.

### Lab 4 - Add Visibility Control

We mentioned above that performance is a critical aspect of building family content, and significant performance improvements can be achieved by making use of the visibility settings accessible for each element in a family through the FamilyElementVisibility class, which can define which levels of detail and which types of views it appears in. That is the focus of this final lab, which implements the following:

* Implement a single line representation for the column in coarse view:
  + Single symbolic line representation in plan view.
  + Single model line representation in model view .
* Set the visibility control so that the single line representation is used in coarse view.



The classes and methods used include:

doc.FamilyCreate.NewSymbolicCurve()

doc.FamilyCreate.NewModelCurve()

FamilyElementVisibility(FamilyElementVisibilityType.ViewSpecific/Model)

FamilyElementVisibility.IsShownInFine, etc.

pLine.SetVisibility(pFamilyElementVisibility)

The detailed documentation and C# and VB code snippets for this lab are provided by the document ‘Family Lab4 - Add Visibility Control\_CS.rtf’ in the csdoc and vbdoc subdirectories of the sample material.

### Learning More

Here is an overview of some available resources for further learning:

* Online Help, Developer's Guide and SDK Samples
* Families Guide  
  <http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=13376394>
* DevTV Introduction to Revit Programming  
  <http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=2484975>
* Recording of Revit 2010 Programming Introduction Webcast  
  <http://www.adskconsulting.com/adn/cs/api_course_sched.php>
* Recording of Revit Family API Webcast  
  <http://www.adskconsulting.com/adn/cs/api_course_sched.php> > Revit Family API  
  [http://thebuildingcoder.typepad.com/blog/2009/08/the-revit-family-api.html](http://thebuildingcoder.typepad.com/blog/2009/06/revit-family-api.html)
* Discussion Group  
  <http://discussion.autodesk.com> > Revit Architecture > Revit API
* API Training Classes  
  <http://www.autodesk.com/apitraining>
* The Building Coder, Jeremy Tammik's Revit API Blog  
  <http://thebuildingcoder.typepad.com>
* Autodesk Developer Network  
  <http://www.autodesk.com/joinadn>
* DevHelp Online for ADN members  
  <http://adn.autodesk.com>

ADN members can ask unlimited API questions through our DevHelp Online interface. Also watch out for our regular ADN DevLab events. DevLab is a developer workshop where you can come and discuss your programming issues with the ADN DevTech team. DevLabs are free to both ADN and non-ADN members. To register, please email [dev-req@autodesk.com](mailto:dev-req@autodesk.com).