Revit 2015 API News

Notes from the Revit 2015 API presentation by Anthony Hauck and Scott Conover for DevDays at Autodesk University, Monday 2013-12-02.

All of the information presented is covered by the What's New section in the Revit API help file. Please refer to that for more details.

Agenda

- Revit 2015 Product Themes
- Application Porting
- New Features + API

Revit 2015 Product Themes

These are some of the main themes addressed by this release of the Revit product. You will recognise them from previous years as well:

- Construction – Modeling and detailing solutions for construction and fabrication
- Analysis + Simulation – Predicting building performance and behavior
- Scalability – Supporting projects and teams of any size and complexity
- Interoperability – Leveraging the Autodesk portfolio and industry standards
- Productivity – Providing increased value for existing customers

These product themes obviously strongly influence the API development as well.

Now we turn to the main two aspects of interest to application developers: application porting, discussing changes affecting existing applications, and the exciting area of all the new API features added in this release.

Application Porting

Steps required for an existing add-in to support the next release.

Application Porting | Platform

- .NET 4.5.
- The native runtime libraries are from Visual C++ runtime 11, update 3.
- APIs marked obsolete in earlier releases have been removed.

Application Porting | Units

A small API set introduced in Revit 2014 is obsolete. Replacements existed in Revit 2014 and should be used from now on.

Application Porting | Parameters

- Obsolete Element.Parameter[String]

Replaced by:

- Element.GetParameters(String) – Returns all parameters matching the input name
- Element.LookupParameter(String) – Returns the first matching parameter with the input name
- Element.Parameters – Includes Schedule and Tags parameters as well as Properties Palette ones
- Element.GetOrderedParameters – returns parameters in the Properties Palette

The Parameters property now returns more parameters than it used to, including scheduling and tag properties, not just property palette ones.

Application Porting | Settings

Settings classes are now Elements and support limited standard Element functionality:

- Checkout via id
- Add ExtensibleStorage

FilteredElementCollector now returns:

- DefaultDivideSettings
- StructuralSettings
- ElectricalSetting
- DuctSettings
- DuctSizeSettings
- PipeSettings
- ReinforcementSettings
- WorksetDefaultVisibilitySettings
- RevisionSettings
- ConceptualSurfaceType
- EnergyDataSettings
- StartingViewSettings
- AreaVolumeSettings

Application Porting | Families

The FamilyBase class has been removed.

Family now inherits directly from Element.

Members of FamilyBase moved to Family.

Family.Symbols obsolete; use GetFamilySymbolIds instead.

The Family CurtainPanelHorizontalSpacing and CurtainPanelVerticalSpacing are now accessed from owner family obtained from a curtain panel family document. If previously accessed from a non-owner family, access by editing the family. The curtain spacing methods only work when editing a family.

Application Porting | Selection

Obsolete:

- SelElementSet class
- Selection.Elements.

Replaced by:

- Selection.SetElementIds(ICollection <ElementId> ids)

Set Selected Elements in the active document.

Application Porting | Worksharing

Obsolete:

- WorksetConfiguration.CloseAll
- WorksetConfiguration.OpenLastViewed
- SynchronizeWithCentralOptions.CompactCentralFile (duplicated SWCO.Compact)

Replaced by:

- WorksetConfiguration constructor accepting options for:
- WorksetConfigurationOption.OpenAllWorksets
- WorksetConfigurationOption.CloseAllWorksets
- WorksetConfigurationOption.OpenLastViewed

Application Porting | Reinforcement

Obsolete:

- AreaReinforcement.GetCurveElementIds
- FabricArea.GetCurveElementIds

Replaced by:

- GetBoundaryCurveIds

Application Porting | Views

- ViewDisplayModel – Display settings replace individual View class members
- ViewSheet – Several ViewSheet Revision methods are renamed; corresponding original members are obsolete

View properties moved into a special class, silhouette etc.

Application Porting | Energy Model

Obsolete:

- EnergyAnalysisSpace.SpatialElementId
- EnergyAnalysisOpening.OriginatingElementId
- EnergyAnalysisSurface.OriginatingElementId

Replaced by:

- EnergyAnalysisSpace.CADObjectUniqueId
- EnergyAnalysisOpening.CADObjectUniqueId
- EnergyAnalysisOpening.CADLinkUniqueId
- EnergyAnalysisSurface.CADObjectUniqueId
- EnergyAnalysisSurface.CADLinkUniqueId

Energy model: these properties did not really refer to Revit model element ids, so they have been renamed.

Application Porting | Other

Namespace changes:

- BoundaryCondition – Autodesk.Revit.DB to Autodesk.Revit.DB.Structure
- ExtensibleStorageFilter – Autodesk.Revit.DB to Autodesk.Revit.DB.ExtensibleStorage
- Parameter changes – BuiltInCategory.OST_MassWindow renamed to OST_MassGlazing

Application Porting | Other

- ElementIntersectsElementFilter – Filter no longer passes target element
- MeshTriangle – No longer inherits from APIObject
- CurtainGridLine.Move obsolete – Use ElementTransformUtils.MoveElement
- TableSectionData.InsertColumn (int index, bool bCreateCellData) – Replaced by TableSectionData.InsertColumn (int index)
- Wire API – NewWire obsolete and replaced by Wire.Create

New Features + API

Here are the opportunities for new automation and customization, lots of potential for improvements to existing add-ins. The trend to 'eating our own dog food' is continued...

Scalability | Graphics Consolidation

- Optimized graphics for repeated materials
- Similar Elements drawn together
- Increased View drawing performance

Graphics is a rich field for performance improvement and hundreds of tests are run constantly.

Scalability | Performance Improvements

- Optimized family handling in memory
- Improved geometry handling
- Regeneration optimizations

Note that individual developers are named in the images presented here. The long-term family separation project is included in this release, so model open and workset test are slowed down. One long-term target is to provide multi-processor support for regeneration.

Productivity | Sketchy Lines

New visualizations in standard Views

Modify Sketchy Lines visibility in Views using the View methods GetSketchyLines and SetSketchyLines. Jitter etc. is adjustable.

This is a user facing feature; some users can get angry when the final result does not end up exactly like the initial beautiful rendering. Video: 1_sketchy_lines.mp4.

Productivity | Family Browser

The family browser has been a source for many end user feature requests.

Anthony has a 12 meter long printout in 10 point font of a project browser that is not even fully expanded hanging in his office.

We now provide a new dockable family browser built using the public API and the following features:

- Dockable Family Browser
- Intelligent Search
- Favorites Management
- Sub-item quantity display

You can search for a string like 'door', for example, and define 'favourite' families for quick and easy lookup. Video: 2_project_browser.mp4.

Productivity | Family Browser View Interaction

UIDocument.ActiveGraphicalView now returns the active graphical View of the active Document canvas and excludes Project Browser, System Browser, and similar Views. It allows you to read the currently active graphical view of the currently active document. Unlike UIDocument.ActiveView, this property will never return auxiliary views like the Project Browser or System Browser if the user has happened to make a selection in one of those views.

UIDocument.RequestViewChange requests to change the active View and executes on return from API context. It even works from the Idling and ExternalEvent. It requests to change the active view by posting a message asynchronously. Unlike setting the ActiveView property, this will not make the change in active view immediately. Instead the request will be posted to occur when control returns to Revit from the API context. This method is permitted to change the active view from the Idling event or an ExternalEvent callback.

This functionality is used by the browser, and arose because we are internally making use of the API, plus have the ability enhance it to fit our needs :-)

Productivity | Family Browser Prompts

The new UIDocument PostRequestForElementTypePlacement method requests the user to place instances of the specified ElementType. Again, it executes on return from the API context. Similarly:

- PromptToPlaceViewOnSheet
- PromptToPlaceElementTypeOnLegendView
- PromptToMatchElementType

PostRequestForElementTypePlacement places a request on Revit's command queue for the user to place instances of the specified ElementType. This does not execute immediately, but when control returns to Revit from the current API context. This method starts its own transaction. In a single invocation, the user can place multiple instances of the input element type until they finish the placement (with Cancel or ESC or a click elsewhere in the UI). This method invokes the UI when control returns from the current API context; because of this, the normal Revit UI options will be available to the user, but the API will not be notified when the user has completed this action. Because this request is queued to run at the end of the current API context, only one such request can be set (between this and the commands set by UIApplication.PostCommand). This differs from UIDocument.PromptForFamilyInstancePlacement as that method can be run within the current API context, but the user is not permitted full access to the user interface options during placement.

PromptToPlaceViewOnSheet prompts the user to place a specified view onto a sheet. Set its allowReplaceExistingSheetViewport argument to true to allow the user to replace the existing viewport.

All main instance placement options are accessible.

Productivity | Family Browser Drag + Drop

The new interface IControllableDropHandler inherits from IDropHandler and includes an extra interface executed when custom data is dragged and dropped into the Revit UI that enables the handler to verify whether the drop event can be executed on the given view.

You can implement its CanExecute predicate method to inform Revit whether the drop event can be executed onto the given view.

Productivity | Family Browser Default Element Type

Revit has a default type for different categories. This default type is shown in the user interface when the related tool is invoked to create an element of this category.

Families:

These members provide read and write access to the default type for a given family category id:

- Document.GetDefaultFamilyTypeId(categoryId) – Gets the default family type id associated to the given family category id.
- Document.SetDefaultFamilyTypeId(categoryId, familyTypeId) – Sets the default family type id associated to the given family category id.
- Document.IsDefaultFamilyTypeIdValid(categoryId, familyTypeId) – Checks whether the family type id is valid to set as default for the given family category id.
- ElementType.IsValidDefaultFamilyType(familyCategoryId) – Identifies if a type is a valid default family type for the given family category id.

Other types:

- Document.GetDefaultElementTypeId (ElementTypeGroup)
- Document.SetDefaultFamilyTypeId (ElementTypeGroup, elementTypeId)
- Document.IsDefaultFamilyTypeIdValid(ElementTypeGroup, elementTypeId)

Demo:

This is an external application defining a docked panel. Define a wall type, note how it changes the default type in the wall properties. Place a wall, change default type, place another.

Productivity | Ordered Parameters

- Move parameters in parameter group
- New Parameters added at group end
- FamilyManager.GetParameters
- Element.GetOrderedParameters
- FamilyManager.ReorderParameters

Productivity | Shared Parameters

Create read-only Shared Parameters editable by any API application.

The Definitions.Create options class argument includes:

- Name
- Type
- GUID
- Visibility
- Description (displayed as UI Tooltip)

Read-only shared parameters can be defined and can display their own custom tooltip.

Demo: An example to show ordering could be added. Create a shared parameter, set its value, show that it cannot be edited, although it can be changed programmatically.

It provides a way to store data that user can see but cannot change. Use extensible storage for data that should not be displayed to the user at all.

This can be used to place ids or other identifiers on content etc.

Productivity | Referenced File Service

Provide Revit with content from arbitrary external sources.

Revit 2015 introduces a new service which allows applications to provide Revit with the content of resources that are used in the model but stored in external files, such as the keynote data or a Revit link.

- Reference Service Framework
- RVT Links
- Keynote File
- Assembly Code File
- Link Assembly Code file
- Provide Keynote and Uniformat standards
- Realtime online update

The new IExternalResourceServer interface allows an application to provide external resources (such as linked files) from arbitrary locations using the following methods:

- IExternalResourceServer.LoadResource – Allows an application to create a custom implementation to load a requested resource into a Revit project.
- IExternalResourceServer.SupportsExternalResourceType – Gets whether a server can provide the specified type of external resource.
- IExternalResourceServer.SetupBrowserData – Sets the available resources and subfolders that a server can provide.
- IExternalResourceServer.HasResource – Gets whether the given ExternalResourceReference represents a resource which the server can provide.
- IExternalResourceServer.IsResourceWellFormed – Gets whether the given ExternalResourceReference is syntactically valid for the application.
- IExternalResourceServer.GetIconPath – Gets the path to an icon file which will be displayed in Revit user interfaces related to this server.
- IExternalResourceServer.GetInSessionPath – Gets the path and name that will be used for the current session within Revit's user interface to identify the specified resource to the user.
- IExternalResurceServer.GetResourceVersionStatus – Gets whether the given version of a resource is current or not.
- IExternalResourceServer.GetResourceCachedPath – Gets the local path where a server will place the Revit Link corresponding to the given external resource. This function is not applicable to servers which do not support Revit Links. Non-Revit-link servers should return the empty string.

The new ExternalResourceLoadContent class and its subclasses allows an IExternalResourceServer to return the data that should be used within Revit from its LoadResource method. When LoadResource is invoked, Revit will pass in a subclass of ExternalResourceLoadContent that can hold the actual content data for a particular external resource supplied by a server.

This is another example of us eating our own dog food, making use of the API to implement product functionality.

The use of a referenced file service can make on-traditional data sources accessible.

You can build a connector to an arbitrary data source, as long as Revit sees what it expects on the receiving end.

RVT links, keynote files, assembly files can all be provides by a referenced file service.

This is also a step towards decomposing Revit into a set of services on desktop or cloud.

In future, we may decompose more of the platform into chunks and enable the API to replace some of them.

Revit 2014 already used this functionality to achieve this for MEP calculations.

Now you have more options where your data might live, and can easily enable real-time updates to all customers.

Here is a video 3_keynote_wiki.mp4 showing an example of maintaining Revit project and best practice data in a wiki using the following methods and properties:

- RevitLinkType.Create
- RevitLinkType.LoadFrom – ModelPath, ExternalResourceReference
- RevitLinkType.AttachmentType – Attachment, Overlay

This is not just enabling wiki as source, but anything at all. In the demo, the summary was changed, reloaded, ... use your imagination on this one!

Access your web service, deliver continuous updates. Use for any kind of data that requires maintenance and delivery to your users. Enable Revit to talk to a variety of data sources, talk to anything you want.

Productivity | ...and more!

A huge list of further important items:

- Trim + Extend multiple elements
- Duplicate View Naming
- Pinned Element retention
- Add Links in Manage Links dialog
- Tag Leader consistency
- MEP calculation per segment
- Change referenced views
- Shared Parameters in View Titles
- Default Divide Path settings
- Customizable tooltips
- Category.CategoryType
- ElementType.FamilyName
- ElementType duplication event
- Family loading Event

Duplicate view naming: a duplicated view now shows up next to the original, making it easier to access and compare the two.

Deleting a pinned element produced a warning message, or even an error. Now it simply will not delete.

Why not enable **add** links in the **manage** links dialogue?

Text and tags now work the same.

MEP calculation per segment: not the maximal flow overall, but the real flow at the picked segment.

Customizable tooltip: enable better add-in integration. An add-in can go deeper and deeper.

Family name: use to determine oval versus round duct etc.

Productivity | ...and even more!

Even more new functionality, related to view interaction:

- View.Title
- View.AreModelCategoriesHidden
- View.AreAnnotationCategoriesHidden
- View.AreAnalyticalModelCategoriesHidden
- AreImportCategoriesHidden
- View.IsAssemblyView
- ViewDrafting.Create
- View3D.OrientTo(XYZ forwardDirection)
- ReferenceableViewUtils manages reference sections and callouts
- ChangeReferencedView
- GetReferencedViewId

Read the documentation about this, no time for details.

The View3d OrientTo method takes one directional vector; it was hard to calculate and set the view yourself. We do not want to force you to do complex math! We are happy to simplify things for you.

Interoperability | Energy Model

The gbXML export now supports two different ways to determine the energy model via the new GBXMLExportOptions.ExportEnergyModelType that can be:

- SpatialElement – Energy Model defined by Rooms or Spaces
- BuildingElement – Energy model defined by analysis of building volumes

The BuildingEnvelopeAnalyzer class:

- Identifies Elements composing the building envelope
- Finds building Elements bounding enclosed volumes

The algorithm used to find the perimeter of spaces changed: it implements a voxelisation algorithm, pouring sugar cubes into a space until it is full. Reflexively studying geometry in model. Return primitive planes.

Interoperability | Analytical Model Alignment

Expose some things that were not accessible to control the alignment of surfaces and sticks, aligned to rest of model, how to project or extend, etc.:

AnalyticalModelSurface is the existing class of Floor | Wall | Slab providing new members:

- Alignment
- Projection
- Extension

We now also have an new AnalyticalModelStick class of Beam | Brace | Column providing access to:

- Alignment
- Projection
- Extension

The corresponding members of AnalyticalModel are obsolete.

Interoperability | IFC Import + Linking

Linking IFC files and referencing IFC geometry is a long-time request from the Nordic countries:

- Link IFC files
- Reference IFC geometry
- IFC Manage Links dialog tab
- Improved geometry performance
- Uses new API Import Framework
- OpenSource Importer

Import IFC into an existing, not just into a new Revit model. Link IFC model in or reference it. Access via an own tab in manage links. All of this is built on the public API import framework like the IFC exporter on SourceForge, updated every six weeks. Note that it took two years to open up the IFC exporter completely, so it might take two releases for this as well. Supports both old and new action.

The RevitLinkType.CreateFromIFC method creates a new linked IFC type that can be placed with standard RevitLinkInstance methods. To load it, you can use Application.OpenIFCDocument taking the file path and IFCImportOptions supporting the following choices:

- Action (open or link)
- Intent (parametric or reference)
- AutoJoin (applies to parametric only)

Interoperability | Import Framework

A good support for importing IFC files requires an effective framework for handling arbitrary shapes. That is provided by the new direct shape functionality.

The new DirectShape element class supports:

- Category assignment
- Geometry validated for Revit
- Brep (solids | shells) or faceted bodies
- Maintained in the Project

The DirectShapeLibrary stores the geometry of shapes, permitting reuse in different types.

DirectShapeType provides the type element for the direct shape.

The TessellatedShapeBuilder class creates a solid or polymesh from connected planar facets, adding TessellatedFace objects one by one. It heals some imprecisions and discontinuities within inputs.

Consider the IFC pipeline. Direct shape allows bringing in arbitrary stuff and instantiate it inside Revit. This is not complete yet, and we would really like to hear some feedback on this, because we have no idea yet what you would like to bring in. We need to understand your workflow. A direct shape can be populated with solids using existing mechanisms or faceted bodies. The tessellated shape builder healing functionality helps avoid GIGO, i.e. garbage in, garbage out.

Demo: Load STL. The imported STL file can go right into a project without requiring a family shell.

Interoperability | ...and more!

Some structural stuff:

Enhancements for Load Orientation:

- LoadBase.OrientTo
- LoadBase.HostElementId
- LoadBase.WorkPlaneId

BoundaryConditions Orientation + Type methods:

- GetOrientTo
- SetOrientTo
- GetDegreesOfFreedomCoordinateSystem
- GetBoundaryConditionsType

Construction | Revisions

Revit 2015 provides complete support for manipulating revisions and revision clouds, including new drawing tools as well.

Access to revisions is a long-time request, not easy to handle, many different standards. Deleting revisions is not easy either. This functionality can form a basis for an entire new class of revision management applications. You can interact directly with the revision cloud.

Here are some of the API tools for accessing this to Read | Modify | Create | Reorder Revisions:

- Revision.CombineWithNext, CombineWithPrevious – Combine a Revision with the next or previous Revision
- ViewSheet.GetRevisionNumberOnSheet – access to the Revision Number when the numbering in the project is "by sheet"

Combining the Revisions means that the RevisionClouds and revision tags associated with the specified Revision will be reassociated with the next Revision and the specified Revision will be deleted from the model. This method returns the ids of the RevisionClouds that were reassociated.

Project Revision Settings:

- RevisionAlphabet – Characters for alphabetic Revisions
- RevisionCloudSpacing – Sizing of Revision Cloud graphics
- RevisionNumbering – by sheet or by project

The new Revision class allows an application to read and modify the existing revisions in a project and also to create new revisions. Revision is a subclass of element. Some other methods:

- Revision.GetAllRevisionIds provides an ordered list of all of the Revisions in the document.

- Revision.ReorderRevisions allows the ordering of the Revisions within the project to be changed.
- Revision.Create creates a new Revision in the document.

Revision properties:

- Revision.Description
- Revision.Issued
- Revision.IssuedBy
- Revision.IssuedTo
- Revision.NumberType
- Revision.RevisionDate
- Revision.Visibility
- Revision.SequenceNumber
- Revision.RevisionNumber

Customer Satisfaction | Revision Clouds

Here is the API functionality supporting Read | Modify | Create | Reorder Revision Clouds:

- Element.Geometry – Returns the curved lines in the cloud
- RevisionCloud.GetSketchLines – Returns the lines in the cloud sketch
- ViewSheet.GetRevisionCloudNumberOnSheet – Revision Number for a RevisionCloud numbered by sheet.

The new RevisionCloud class allows an application to access information about the revision clouds that are present within a model and to create new revision clouds. RevisionCloud.Create create a new instance. RevisionCloud.RevisionId read and modify the associated Revision. RevisionCloud.IsRevisionIssued lets you check whether a RevisionCloud is associated with a Revision that has already been issued. RevisionCloud.GetSheetIds returns the ids of the ViewSheets the RevisionCloud may appear in. ViewSheet.GetRevisionCloudNumberOnSheet accesses the Revision Number for a RevisionCloud when the numbering in the project is by sheet.

Revision cloud geometry: the Element.Geometry property has been enhanced to return geometry from RevisionCloud elements and will return the actual curved lines that make up the cloud. RevisionCloud.GetSketchLines reads the Lines that form the RevisionCloud's sketch, i.e. the sketched lines and not the actual curved cloud geometry.

Construction | Revisions API

Demo: The non-rectangular revision cloud generator uses another new piece of geometry functionality, CurveLoop.CreateViaOffset, to offset the cloud outwards from the room boundary polygon to make the cloud bigger than the room.

Construction | Schedules

Type and instance images in schedule: last year images could be added to the header, this year to a row as well. The schedule view just shows the name of image, and the sheet shows the actual image itself.

- ParameterType.Image
- ElementId of ImageType
- ImageType.Create
- ImageType.Reload
- ImageType.ReloadFrom

Additional Wall Parameters:

- Base Constraint
- Base Offset
- Unconnected Height
- Top Constraint
- Top Offset

Custom Grand Total Title.

More Schedule Filters: the number of filters was limited to four by the UI dialogue, and no other reason at all; customers tell us that six is what we need; we decided to add support for eight now.

Construction | Schedules

Demo: ScheduleImages provides several buttons; one is thumbnail; takes furniture family and displays previews; turns into image and adds to instances, sheet displays them.

Construction | Part Reinforcement

Reinforcement can now be hosted by parts. Video 4_reinforcement.wmv.

The existing reinforcement creation functions now accept part element hosts:

- Rebar.CreateFromCurves
- Rebar.CreateFromCurvesAndShape
- Rebar.CreateFromRebarShape
- AreaReinforcement.Create
- PathReinforcement.Create

This new function identifies valid host elements for reinforcement:

- RebarHostData.IsValidHost

Construction | Reinforcement Numbering

Reinforcement numbering makes use of the new numbering framework that may be used in future for any kind of numbering based on what is in model. It is currently limited to just rebar. It gives a unique address to every piece of rebar. You define a schema for numbering. Currently supports rebar and fabric reinforcement, will probably be expanded. The entire reinforcement numbering dialogue is written completely in API. The features include:

- Automatic numbering
- Configurable schemas
- Numbering by host

Video 5_numbering.wmv.

Two new classes are provided by the API to define Element organization for numbering and tagging:

- NumberingSchema – Controls Element numbering for a Category
- NumberingSchemaType – Rebar and Fabric Reinforcement

Live demo: In the model, two things are achieved: first, new rebar is automatically added to parts created from host objects that support rebar, e.g. floor. Second, rebar numbering makes use of a parameter on the rebar element called partition name of the group of rebar that should be numbered consecutively from start number onward. New rebar is added at the end. You can change the starting number and remove gaps in the numbering. The UI is created based on the API and is accessible under Rebar panel > Rebar numbering in the slide-down panel along with reinforcement settings. The macro does two things: rebar cannot normally be copied from one host to another, but the macro achieves this by creating new rebar in parts, in this case fabric, by querying their boundary. The partition parameter to use is copied from the source rebar and adapted based on the level. In our case, that generates a sequence with a new starting number, e.g. copying from Level 1 to 4 shifts the numbering from 100-110 to 400-410. In Revit 2015, the numbering framework is not extensible to other types besides rebar and fabric areas, but it could be in the future.

Construction | Framing References

Framing references members like beams and defines how they intersect. An end condition beam can be framed into another beam. The old behaviour cut it off at the bounding box. Now you can set a plane reference defining where it should frame into. Setting the position of framing ends is demonstrated by video 6_framing.wmv.

API support is provided by the StructuralFramingUtils class supporting choice of face for setback calculation, e.g.:

- CanSetEndReference
- RemoveEndReference
- GetEndReference
- SetEndReference

Note that these methods are used to set the reference of joined elements only and allow setting setback of structural framing by selecting faces of adjoined elements. Similar tools is also available in the UI context for non-concrete and joined structural framings. The StructuralFramingUtils methods cannot be applied to stand-alone objects, however.

Construction | Structural Sections

Structural sections can now be parameterised. We have standard steel section types W flange, C, hollow rectangle etc., so you can define your types according to those rules. The API has classes surrounding each section shape, so we can now access width, height, thickness, etc. using standard parameters in a unified way for each shape. This makes it much easier to identify which parameters to use.

StructuralSection is an adjunct class associated with steel framing FamilySymbols. It supports multiple subclasses for different configurations, e.g.

- StructuralSectionWideFlange
- StructuralSectionRoundHSS
- Etc.

Get | Set section dimensions consistently:

- Access by FamilySymbol.GetStructuralSection
- Section type by FamilySymbol.GetStructuralSectionType

Construction | Wire

We now have API access to all MEP wire types: the wire API is completely revamped and supports all different UI options for wire editing, e.g. add, move vertex, etc.

Wire.Create supports straight, arc, chamfer, and spline wires.

Wire read | write:

- WiringType
- NumberOfVertices
- AppendVertex
- InsertVertex
- GetVertex
- SetVertex
- RemoveVertex
- ConnectTo – connect wires via connectors

… and more!

Full access to stacked wall relationships:

- GetStackedWallMemberIds
- IsStackedWall, IsStackedWallMember
- StackedWallOwnerId

Dynamic Dynamic Updater Control:

- UpdaterRegistry.EnableUpdater
- UpdaterRegistry.DisableUpdater
- UpdaterRegistry.IsUpdaterEnabled

Geometry enhancements:

- CurveElement.SetGeometryCurve
- CurveElement.SetSketchPlaneAndCurve
- FreeFormElement.UpdateSolidGeometry

Stacked walls is a really long-standing wish list, how to identify the relationships between the main wall and its components.

The DMU enhancement enables and disables updaters dynamically.

An important geometry enhancement is the ability to avoid auto-join attempts after moving a curve. Previously, if two curves are connected at their endpoints and you move one, the other will always follow it, whether you want it to or not. Curve methods: when a curve endpoint is modified, an attached curve will normally moved. That can now be suppressed. This is to support dynamo operations. Another set of geometry enhancements includes curve offset + curve loop offset functionality, and we saw it used for the revision cloud generator. A curve reversal method has also been added.

Questions and Answers

- IFC import: access to geometry and properties? yes, direct shape object has actual geometry, material, colour.
- Read-only shared params: convert existing params to read-only, cannot convert on the fly, you have to replace.
- AutoCAD core engine; something for Revit? under consideration.
- Elevation parameter?
- JavaScript API in Revit? We use SharpDevelop as a macro environment, and whatever that supports.

Demos

As a post scriptum, here are some final notes on some of the accompanying sample applications and models:

- You can use an empty model for the STLimport.
- One sample model is the archictectural scheduleimagesproject.
- One model for structure demo + macro for rebar.
- STL sample file modular shrine comes in as a solid, even though it just defines a bunch of triangles, in good quality STL, with no leaking holes.
- STL import
- Schedule images
- Samples incl. revision default type
- Read-only shared params
- Macro
- For import STL.rvt
- For rebar operations.rvt
- ReadonlySharedParameters
- ReinforcementNumbering
- Revit2015Samples
- STLImport
- Schedule images project.rvt
- ScheduleImages