# Major changes and renovations to the Revit API

# API changes

## NewFamilyInstance() exception when presented inactive FamilySymbols

* All NewFamilyInstance() overloads on Autodesk.Revit.Creation.Document and Autodesk.Revit.Creation.ItemFactoryBase now check to ensure that the input FamilySymbol is active (FamilySymbol.IsActive).  Symbols that are not used in the document may be deactivated to conserve memory and regeneration time. When the symbol is inactive, its geometry is empty and cannot be accessed. In order to access the geometry of a symbol that is not active in the document, the symbol should first be activated by calling FamilySymbol.Activate().
* If the symbol is not active the method will now throw.  This prevents the situation where the new FamilyInstance created will not generate proper geometry or intersections with related elements.

## TextNote and Leader API behavior and interface changes

The Revit API classes related to TextNotes and Leaders have been significantly renovated.

### TextElement class

New methods and properties

* TextElement.Width - Width of the area of the text content
  + This replaces **TextNote.LineWidth**
* TextElement.HorizontalAlignment  – Horizontal alignment of the text content within the text box of the element.
* TextElement.VerticalAlignment - Vertical alignment of the text content within the text box of the element.
  + Collectively, these two new properties replace **TextNote.Align**
* TextElement.KeepRotatedTextReadable - A flag to control how text behaves within a rotated text box.
* TextElement.IsTextWrappingActive - A flag identifying whether text-wrapping is currently active in this text element or not.

### TextNote class

New methods and properties

* TextNote.Create – Creates a new text note element with the given properties.  This includes a TextNoteOptions object, which encapsulates several options used for creating notes.
  + This replaces **ItemFactoryBase.NewTextNote()**
  + This change also fixes an issue where creating a text note would incorrectly wrap text lines to be too short
* TextNote.LeaderCount - The number of leader objects currently attached to the text note.
* TextNote.LeaderLeftAttachment -  Gets/sets the attachment position of leaders on the left side of the text note.
* TextNote.LeaderRightAttachment - Gets/sets the attachment position of leaders on the right side of the text note.
* TextNote.GetLeaders - Returns a collection of leaders currently attached to the text note.
  + This replaces **TextNote.Leaders**

### Leaders

The LeaderArray class has been deprecated, thus members which referred to it have been replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **Replacement** |
| TextNote.Leaders | TextNote.GetLeaders() |
| AnnotationSymbol.Leaders | AnnotationSymbol.GetLeaders() |

The Leader class now includes new members:

* Leader.Anchor  – Anchor point of the Leader.
* Leader.LeaderShape - Geometric style of the leader.
* Leader.IsOwned - Validates that the leader is attached to (owned by) a valid element, such as an annotation tag.

## DatumPlane API (Levels, Grids, Reference Planes) changes

### DatumPlane

The new class:

* Autodesk.Revit.DB.DatumPlane

is now a base class for element types that represent a datum surface (level, grid or reference plane)

This base classes offers many new methods and properties:

* DatumPlane.ShowBubble() - Shows bubble in the specified view.
* DatumPlane.HideBubble() - Hides bubble in the specified view.
* DatumPlane.IsBubbleVisibleInView() - Identifies if the bubble is visible or not in a view.
* DatumPlane.HasBubble() - Identifies if the DatumPlane has bubble or not.
* DatumPlane.IsVisibleInView() - Checks if the datum plane is visible in the specified view.
* DatumPlane.Maximize3DExtents() - Maximize the 3D extents of datum plane.
* DatumPlane.GetDatumExtentType() - Identifies if the end of the datum plane is aligned with 3D extents or is set to vary specifically in the indicated view.
* DatumPlane.SetDatumExtentType() - Sets whether or not the end of the datum plane is aligned with 3D extents or is set to vary specifically in the indicated view.
* DatumPlane.GetCurvesInView() - Gets the extents to match the curves within a view.
* DatumPlane.SetCurveInView() - Sets the extents to match the curve.
* DatumPlane.IsCurveValidInView() - Checks if the curve is valid to be as the extents for the datum plane in a view. The curve must be bound and coincident with the original one of the datum plane.
* DatumPlane.GetPropagationViews() - Gets a list of candidate views which are parallel to the current view and to which the 2D extents of the datum may be propagated.
* DatumPlane.PropagateToViews() - Propagates the 2D extents applied to this datum to the specified parallel views.
* DatumPlane.IsLeaderValid() - Checks if the specified leader is valid for the datum plane in the given view.
* DatumPlane.GetLeader() - Gets a copy of the leader applied to the indicated end of the datum plane.
* DatumPlane.SetLeader() - Sets the leader to the indicated end of the datum plane.
* DatumPlane.AddLeader() - Adds leader to the indicated end of the datum plane.

### Level

Some members related to levels have been obsoleted and replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| Level.LevelType | Element.GetTypeId()  Element.ChangeTypeId() |
| Level.PlaneReference | Level.GetPlaneReference() |
| ItemFactoryBase.NewLevel() | Level.Create() |

### Grid

Some members related to grids have been added, or obsoleted and replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| Grid.GridType | Element.GetTypeId()  Element.ChangeTypeId() |
| Grid.ExtendToAllLevels() | DatumPlane.Maximize3DExtents() |
| N/A | Grid.SetVerticalExtents() - Adjusts the grid to extend through only the vertical range between bottom and top. |
| Autodesk.Revit.Creation.Document.NewGrid() | Grid.Create() |
| Autodesk.Revit.Creation.Document.NewGrids() | Use Grid.Create() repeated as necessary. |

### ReferencePlane

Some members related to levels have been obsoleted and replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| ReferencePlane.Plane | ReferencePlane.GetPlane() |
| ReferencePlane.Reference | ReferencePlane.GetReference() |

## Structural API changes

### ContFooting and ContFootingType class and members renamed

The ContFooting and ContFootingType classes have been replaced by new classes:

* WallFoundation
* WallFoundationType

The following members have been modified or added to WallFoundation:

|  |  |
| --- | --- |
| **Removed member** | **New/replacement members** |
| ContFooting.FootingType | WallFoundation.GetFoundationType()  WallFoundation.SetFoundationType() |
| Document.NewFoundationWall(ContFootingType, Wall) | Document.NewFoundationWall(WallFoundationType, Wall) |
| N/A | WallFoundation.WallId |

### AnalyticalModel

* AnalyticalModel.GetLocalCoordinateSystem() now supports cylindrical walls and bent analytical model sticks.

### AnalyticalModelSweptProfile class renamed and relocated

The class:

* Autodesk.Revit.DB.Structural.AnalyticalModelSweptProfile

has been renamed and moved to become:

* Autodesk.Revit.DB.SweptProfile

All members of the class stay the same.

Two members have been transferred from the AnalyticalModel class to the FamilyInstance class:

|  |  |
| --- | --- |
| **Removed member** | **New/replacement members** |
| AnalyticalModel.HasSweptProfile | FamilyInstance.HasSweptProfile |
| AnalyticalModel.GetSweptProfile() | FamilyInstance.GetSweptProfile() |

### Loads

The Revit API classes related to Loads have been significantly renovated.

The new class

* Autodesk.Revit.DB.Structure.LoadBase

has been introduced as a new base class for PointLoad, LineLoad and AreaLoad. Properties of this class affect properties of all 3 types of loads:

* LoadBase.LoadCaseId - read/write property representing the load case for this load
* LoadBase.IsHosted
* LoadBase.LoadCase
* LoadBase.IsReaction

In the subclasses of LoadBase, there are some new members, and many members have been deprecated and replaced, as shown in the following table.

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| PointLoad.Force | PointLoad.ForceVector |
| PointLoad.Moment | PointLoad.MomentVector |
| N/A | PointLoad.Point - this property is now read/write. |
| LineLoad.Point[] | LineLoad.StartPoint  LineLoad.EndPoint  LineLoad.SetPoints() |
| LineLoad.Force[]  LineLoad.Force1  LineLoad.Force2 | LineLoad.ForceVector1  LineLoad.ForceVector2 |
| LineLoad.Moment[]  LineLoad.Moment1  LineLoad.Moment2 | LineLoad.MomentVector1  LineLoad.MomentVector2 |
| LineLoad.UniformLoad | LineLoad.IsUniform |
| N/A | LineLoad.IsProjected |
| Document.NewLineLoad() | LineLoad.Create() |
| AreaLoad.NumLoops  AreaLoad.NumCurves[]  AreaLoad.Curve[] | AreaLoad.GetLoops()  AreaLoad.SetLoops() |
| AreaLoad.Force[]  AreaLoad.Force1  AreaLoad.Force2  AreaLoad.Force3 | AreaLoad.ForceVector1  AreaLoad.ForceVector2  AreaLoad.ForceVector3 |
| AreaLoad.RefPoint[] | AreaLoad.GetRefPoint() |
| N/A | AreaLoad.IsProjected |
| N/A | AreaLoad.Area |
| Document.NewAreaLoad() | AreaLoad.Create() |

### Load cases and load combinations

The Revit API classes related to Load Cases and Load Combinations have been significantly renovated. This has introduced new members to the LoadCombnation, LoadCase, LoadNature and LoadUsage class, and resulted in several previously existing member being marked deprecated and replaced, as shown in this table:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| Document.NewLoadCombination() | LoadCombination.Create() |
| LoadCombination.CombinationCaseName[] | LoadCombination.GetCaseOrCombinationIds(), obtain the name from associated LoadCase element |
| LoadCombination.CombinationNatureName[] | LoadCombination.GetCaseOrCombinationIds(), obtain the name from associated LoadNature element |
| LoadCombination.Factor[]  LoadCombination.NumberOfComponents | LoadCombination.GetComponents()  LoadCombination.SetComponents() |
| LoadCombintation.UsageName[]  LoadCombination.NumberOfUsages | LoadCombination.GetUsageIds()  LoadCombination.SetUsageIds() |
| LoadCombination.CombinationState  LoadCombination.CombinationStateIndex | LoadCombination.State (translate the enum to string or int as required) |
| LoadCombination.CombinationType  LoadCombination.CombinationTypeIndex | LoadCombination.Type (translate the enum to string or int as required) |
| N/A | LoadCombination.IsThirdPartyGenerated |
| Document.NewLoadCase() | LoadCase.Create() |
| N/A | LoadCase.Name is now connected to this element's name properly |
| N/A | LoadCase.Number |
| N/A | LoadCase.NatureId |
| N/A | LoadCase.NatureCategory |
| Document.NewLoadNature() | LoadNature.Create() |
| N/A | LoadNature.Name is now connected to this element's name properly |
| Document.NewLoadUsage() | LoadUsage.Create() |
| N/A | LoadUsage.Name is now connected to this element's name properly |

### LoadComponent

The new class:

* Autodesk.Revit.DB.Structure.LoadComponent

is responsible for association of LoadCase or LoadCombination ids as factors in the load combination.

### BoundaryConditions

In the BoundaryConditions class, there are some new members, and some members have been deprecated and replaced, as shown in the following table.

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| BoundaryConditions.AssociatedLoad  BoundaryConditions.associateWithLoad() | BoundaryConditions.AssociatedLoadId |
| N/A | BoundaryConditions.HostElementId |
| BoundaryConditions.Curve[] | BoundaryConditions.GetCurve() |
| BoundaryConditions.NumCurves  BoundaryConditions.Curve[] | BoundaryConditions.GetLoops() |

### RebarShapeDefinition

New validation has been introduced to confirm that the name of a shared parameter to be added was notalready used by another shared parameter on the element. This new validation was added to methods:

* RebarShapeDefinition.AddParameter()
* RebarShapeDefinition.AddFormulaParameter()

and this situation will now trigger an exception.

## Geometry API behavior and interface changes

### PlanarFace

The properties:

* PlanarFace.Normal
* PlanarFace.Vector[int]

have been obsoleted.

The replacement properties are:

* PlanarFace.FaceNormal
* PlanarFace.XVector
* PlanarFace.YVector

Note that PlanarFace.FaceNormal will return a vector consistently pointing out of the solid that this face is a boundary for (if it is a part of a solid) (unlike PlanarFace.Normal which it replaced).

### Curve.MakeBound() and Curve.MakeUnbound()

These previously threw an exception if the Curve was marked read-only (IsReadOnly).  These methods now will succeed, but cause the read-only Curve handle to automatically contain a copy of the original curve which is disconnected from its source.  It is still not possible to modify a read-only geometry object directly through manipulation of its geometry.

### CurveLoop iteration

When iterating Curves contained in a CurveLoop, a copy is now received instead of a reference to the original CurveLoop Curve.  This is to prevent possible instability due to attempting to modify the CurveLoop's Curves directly, or if the CurveLoop is collected by the garbage collector while a Curve is still in use.

### CurveLoop.CreateViaThicken()

This method now enforces that the thickness value parameter will result in a curve which exceed Revit's short curve tolerance (Application.ShortCurveTolerance).

### Point creation

New methods to create a Point object had been added.

* Point.Create(XYZ) creates a Point at given coordinates.
* Point.Create(XYZ, ElementId) creates a Point at given coordinates and associates with it a GStyle with the specified ElementId

The methods replace the deprecated:

* Autodesk.Revit.Creation.Application.NewPoint()

### TessellatedShapeBuilderOutcome

The enumerated value SolidOrSheet has been removed from this outcome, and replaced by separate values Solid and Sheet. That allows the TessellatedShapeBuilder caller to detect if the builder was able to create a true solid or an open manifold ("sheet")..

## Energy Analysis and gbXML API changes

### EnergyAnalysisDetailModel creation from building elements and volumes

The function:

* EnergyAnalysisDetailModel.Create()

now offers the ability to create energy model based on analysis of building element boundaries and volumes (set EnergyAnalysisDetailModelOptions.EnergyModelType to BuildingElement).  This option matches the default energy model creation used by Revit's user interface.  The generated energy model is affected by settings in EnergyDataSettings, including the option to use the new enumerated value:

* AnalysisMode.ConceptualMassesAndBuildingElements

This option sets the generation of the EnergyAnalysisDetailModel to use the combination of conceptual masses and building elements.

#### EnergyAnalysisDetailModelOptions

The new property:

* EnergyAnalysisDetailModelOptions.EnergyModelType

indicates whether the energy model is based on rooms/spaces or building elements..  Options are:

* SpatialElement - Energy model based on rooms or spaces.  This is the default for calls when this option is not set, and matches behavior in Revit 2015.
* BuildingElement - Energy model based on analysis of building element volumes.

### Lifecycle of EnergyAnalysisDetailModel elements

Users and API application are now able to create a persistent energy model which can be activated in displays and views.   As a result,

* EnergyAnalysisDetailModel.Create()

now requires a transaction to be active so the new elements can be created in the associated document.

The new function:

* EnergyAnalysisDetailModel.GetMainEnergyAnalysisDetailModel()

returns the main EnergyAnalysisDetailModel contained in the given document - this is the model that may be displayed in associated views.

As a result of this behavioral change, the class EnergyAnalysisDetailModel and the sub-objects EnergyAnalysisSpace, EnergyAnalysisSurface, and EnergyAnalysisOpening are now subclasses of Element.  These elements can be found by element filtering and other Revit API tools.

Because the energy model is now an element, the function:

* EnergyAnalysisDetailModel.Destroy()

has been obsoleted in favor of using Document.Delete() on the EnergyAnalysisDetailModel instead to remove the model and all associated elements.

It is recommended that applications call Document.Delete() on the EnergyAnalysisDetailModel elements that they create, but any energy models created after the main energy model will be deleted automatically before document saving or synchronization.

### EnergyAnalysisSpace API changes

The property:

* EnergyAnalysisSpace.SpaceName

has been renamed from EnergyAnalysisSpace.Name.  This change was necessary due to the inherited Name property on the new parent class, Element.

### EnergyAnalysisSurface API changes

The properties:

* EnergyAnalysisSurface.SurfaceName
* EnergyAnalysisSurface.SurfaceId

has been renamed from EnergyAnalysisSurface.Name and EnergyAnalysisSurface.Id.  These changes were necessary due to the inherited Name and Id properties on the new parent class, Element.

### EnergyAnalysisOpening API changes

The properties:

* EnergyAnalysisOpening.OpeningName
* EnergyAnalysisOpening.OpeningId

has been renamed from EnergyAnalysisOpening.Name and EnergyAnalysisOpening.Id.  These changes were necessary due to the inherited Name and Id properties on the new parent class, Element.

### Export to gbXML behavioral changes

The method:

* Document.Export(string, string, GBXMLExportOptions)

no longer generates the energy model.  It is now required that there be a main energy model stored in the document before this export is invoked.  If no energy model of the designated type exists prior to the call to export, this method will throw an exception.

## RevisionSettings API changes

 Enhanced RevisionSettings provide greater control over Revision numbering.

### Alphanumeric revision settings

Alphanumeric revision settings replace the Alphabetic setting available prior to 2016.

The new class AlphanumericRevisionSettings offers the following members:

* AlphanumericRevisionSettings.Prefix - a prefix to be prepended to each revision number with alphanumeric type.
* AlphanumericRevisionSettings.Suffix - a suffix to be appended to each revision number with alphanumeric type.
* AlphanumericRevisionSettings.GetSequence() - the sequence is a ist of arbitrary strings to be used in consecutive sequence as revision numbers.
* AlphanumericRevisionSettings.SetSequence()

The enumerated value:

* RevisionNumberType.Alphanumeric

replaces the now removed RevisionNumberType.Alphabetic.

In the RevisionSettings class, new members were introduced to provide access to the AlphanumericRevisionSettings:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| RevisionSettings.GetRevisionAlphabet() | RevisionSettings.GetAlphanumericRevisionSettings()  For 2016, GetRevisionAlphabet() returns information if the revision can be represented with alphabetic settings; otherwise, an exception is thrown. |
| RevisionSettings.SetRevisionAlphabet() | RevisionSettings.SetAlphanumericRevisionSettings()  For 2016, SetRevisionAlphabet() redirects to set up an alphanumeric revision setting. |

### Numeric revision settings

The new class NumericRevisionSettings offers the following members:

* NumericRevisionSettings.Prefix - a prefix to be prepended to each revision number with numeric type.
* NumericRevisionSettings.Suffix - a suffix to be appended to each revision number with numeric type.
* NumericRevisionSettings.StartNumber property - the value to be used as the first number in the sequence of numeric revisions.

 The numeric revision settings can be accessed from:

* RevisionSettings.GetNumericRevisionSettings()
* RevisionSettings.SetNumericRevisionSettings()

## ExternalDefinitionCreationOptions

The class:

* ExternalDefinitionCreationOptions

has been renamed from ExternalDefinitonCreationOptions to correct a spelling error.

## MirrorElements

The new overload:

* ElementTransformUtils.MirrorElements(Document, ICollection<ElementId>, Plane, bool)

allows mirroring with or without copying the elements first. The previously existing overload has been obsoleted in favor of this new method.

## ReferenceIntersector

The behavior of ReferenceIntersector with the flag:

* ReferenceIntersector.FindReferencesInRevitLinks

set to true has been improved.   Previously, an applied filter was ignored when returning elements encountered in Revit links, and any element encountered would be returned.  Now the filter will be evaluated for the elements found in the links, and those elements returned only if they pass the filter.   Note that results may not be as expected if the filter applied is geometric (such as a BoundingBox filter or ElementIntersects filter).  This is because the filter will be evaluated for linked elements in the coordinates of the linked model, which may not match the coordinates of the elements as they appear in the host model.  Also, ElementFilters that accept a Document and/or ElementId as input during their instantiation will not correctly pass elements that appear in the link, because the filter will not be able to match link elements to the filter's criteria.

As before, if a list of target ElementIds is set, references in links will be returned only if the ElementId matches the id of the intersected RevitLinkInstance, and the ids will not be compared with the target list.

## BaseImportOptions

The new property:

* BaseImportOptions.ReferencePoint

replaces the BaseImportOptions.SetRefPoint() and  BaseImportOptions.GetRefPoint() methods.

## BoundarySegment

As a cleanup, the unused duplicate classes Autodesk.Revit.DB.Architecture.BoundarySegment and Autodesk.Revit.DB.Mechanical.BoundarySegment have been removed from the API. The API interfaces already used Autodesk.Revit.DB.BoundarySegment instead.

The following changes were made to the BoundarySegment class:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| BoundarySegment.Document | There is no replacement as this class will not carry a reference to the created document after this property is removed. |
| BoundarySegment.Element | BoundarySegment.ElementId |
| BoundarySegment.Curve | BoundarySegment.GetCurve() |
| N/A | BoundarySegment.LinkElementId - the element id of the element in a link instance that forms this boundary. |

## ParameterType enum change

The enumerated value ParameterType.Image has been reordered internally in the ParameterType enumeration.  Be sure to rebuild all API clients that use ParameterType and to only compare ParameterType values to each other and not to their internal integer values.

## Schedule view rotation behavior

Some improvements have been made to API behavior for ScheduleSheetInstance if the instance is currently pinned (Element.Pinned is true):

* Settting ScheduleSheetInstance.Rotation will throw an exception.
* Viewport parameter VIEWPORT\_ATTR\_ORIENTATION\_ON\_SHEET cannot be modified.

## Creating HostedSweeps (Fascia, Gutter, SlabEdge)

The validation of the input edges for:

* Autodesk.Revit.DB.Creation.Document.NewFascia()
* Autodesk.Revit.DB.Creation.Document.NewGutter()
* Autodesk.Revit.DB.Creation.Document.NewSlabEdge()

has been relaxed, and should allow edges that are also allowed by the Revit UI.

## DirectShape API and behavioral changes

### DirectShapes in families

DirectShape elements can now be added directly to families and their geometry will be reflected in instances of that family placed in a host document.

### DirectShape category

DirectShape elements may now only be assigned to top-level Model categories.

### WireframeBuilder and DirectShape support for curves and points

The new class:

* WireframeBuilder

supports input of curves and points to form a geometry representation.  This representation can be assigned to or appended to the geometry shown by a DirectShape or DirectShapeType element.

Curves and points can also now be directly passed to DirectShapes and DirectShapeType via SetShape(ShapeBuilder) or AppendShape(ShapeBuilder).

### ViewShapeBuilder

The methods:

* ViewShapeBuilder.SetShape(DirectShape)
* ViewShapeBuilder.SetShape(DirectShapeType)

have been deprecated.  Their replacements are DirectShape.SetShape(ShapeBuilder) and DirectShapeType.SetShape(ShapeBuilder).

### DirectShape options

The new class:

* DirectShapeOptions

offers options that the API developer can use to control the behavior of created DirectShape elements.  Options can be set on either DirectShape or DirectShapeType.

### Referencing option

DirectShape elements now support by default element references, including dimensions, alignments, and face hosting, as well as snapping.

The property:

* DirectShapeOptions.ReferencingOption

supports options related to the referenceablility of the DirectShape or DirectShapeType - if set to NotReferenceable, the geometry may not be used for dimensioning, snapping, alignment, or face-hosting.   The element may still be selected by the user for operations which do not reference individual geometry objects.

### RoomBounding option

DirectShape elements now supports the ability to participate in room boundary calculations, if they are of an appropriate category for room boundary calculations, and if the associated "Room Bounding" parameter is set to true.

The property:

* DirectShapeOptions.RoomBoundingOption

identifies whether the DirectShape supports an option for the "Room Bounding" parameter to permit participation in room boundary calculations.   The default value is NotApplicable, but this will be changed automatically to SetByParameter for applicable DirectShapes.

## Wire API behavioral changes

Several methods offer modified behavior from Revit 2015, so that they work in a manner more consistent with the Revit UI.

* Wire.Create() - in Revit 2016, after creating the wire, the wire ends display on the device boundary, not the device connector position.
* Wire.ConnectTo() - in Revit 2016, if the wire is already connected when this method is used, the old connection will be disconnected and the wire connected to the new target.
* Wire.RemoveVertex() - in Revit 2016, if the wire vertex is already connected to an element, this method will fail to remove the vertex.  
  In order to remove this vertex, it should be disconnected first, then removed, and then reconnected (if required).
* Wire.InsertVertex() - in Revit 2016, if the start vertex already connects to an element, it is not permtted to insert a new vertex before the start vertex.
* Connector.ConnectTo() - in Revit 2016, if one device already connects multiple wires, only the affected wire's connection will be removed or changed.

BrowserOrganization enum namespace change

The enumerated type BrowserOrganizationType has been moved to the Autodesk.Revit.DB namespace.

## AdaptivePointOrientationType enumerated items renamed

All items of this enumerated type were renamed to better align the names with the corresponding text in the Revit UI. The numeric values of the items weren't modified, allowing existing applications to work. However, to be able to rebuild an application, all point orientations need to be changed to their respective new names.

|  |  |
| --- | --- |
| **Old name** | **New Name** |
| HostReferenceStrictly | ToHost |
| HostReferenceAutoFlip | ToHostAndLoopSystem |
| PlacementVertical | ToGlobalZthenHost |
| PlacementOrthogonal | ToGlobalXYZ |
| FamilyVertical | ToInstanceZthenHost |
| FamilyOrthogonal | ToInstance |

## CompoundStructure.SetLayers() behavioral change

The method:

* CompoundStructure.SetLayers()

now unsets the structural material layer index automatically..  Code that needs this property to be set after changing layers will need to set it explicitly.

## API events - behavioral change

Although the Revit API has never officially supported such a work-flow it is now enforced that registering to and unregistering from events must happen while executing on the main thread. An exception will be thrown if an external application attempts to register to (or unregister from) events from outside of valid API context.

## Collections removed from API

The following unused Revit API collection types have been removed from the API in 2016:

* Autodesk.Revit.DB.Structure.LoadTypeBaseSet
* Autodesk.Revit.DB.Structure.PointLoadTypeSet
* Autodesk.Revit.DB.Structure.LineLoadTypeSet
* Autodesk.Revit.DB.Structure.AreaLoadTypeSet
* Autodesk.Revit.DB.Mechanical.SpaceTagTypeSet
* Autodesk.Revit.DB.AnnotationSymbolTypeSet
* Autodesk.Revit.DB.BoundarySegmentArray
* Autodesk.Revit.DB.BoundarySegmentArrayArray
* Autodesk.Revit.DB.AreaTagTypeSet
* Autodesk.Revit.DB.BeamSystemTypeSet
* Autodesk.Revit.DB.FamilySymbolSet
* Autodesk.Revit.DB.Architecture.FasciaTypeSet
* Autodesk.Revit.DB.FloorTypeSet
* Autodesk.Revit.DB.GenericFormSet
* Autodesk.Revit.DB.GridTypeSet
* Autodesk.Revit.DB.Architecture.GutterTypeSet
* Autodesk.Revit.DB.LevelTypeSet
* Autodesk.Revit.DB.MaterialSet
* Autodesk.Revit.DB.Structure.RebarBarTypeSet
* Autodesk.Revit.DB.Structure.RebarHookTypeSet
* Autodesk.Revit.DB.Structure.RebarCoverTypeSet
* Autodesk.Revit.DB.Structure.RebarShapeSet
* Autodesk.Revit.DB.RoofTypeSet
* Autodesk.Revit.DB.Architecture.BoundarySegmentArray
* Autodesk.Revit.DB.Architecture.BoundarySegmentArrayArray
* Autodesk.Revit.DB.Architecture.RoomTagTypeSet
* Autodesk.Revit.DB.SlabEdgeTypeSet
* Autodesk.Revit.DB.TextNoteTypeSet
* Autodesk.Revit.DB.Structure.TrussTypeSet
* Autodesk.Revit.DB.ViewSheetSets
* Autodesk.Revit.DB.WallTypeSet
* Autodesk.Revit.DB.InstanceArray
* Autodesk.Revit.DB.MeshArray
* Autodesk.Revit.DB.GeometryObjectArray
* Autodesk.Revit.DB.SolidArray
* Autodesk.Revit.DB.Mechanical.BoundarySegmentArray
* Autodesk.Revit.DB.Mechanical.BoundarySegmentArrayArray
* Autodesk.Revit.DB.CurtainSystemTypeSet
* Autodesk.Revit.DB.DimensionTypeSet
* Autodesk.Revit.DB.SpotDimensionTypeSet
* Autodesk.Revit.DB.ContFootingTypeSet

## Methods and properties removed from API

The following previously deprecated methods and properties have been removed from the API:

* Autodesk.RevitAddIns.RevitProduct.Language
* Autodesk.Revit.Creation.Document.NewViewDrafting()
* Autodesk.Revit.DB.Electrical.Wire Autodesk.Revit.Creation.Document.NewWire(Curve, View, Connector, Connector, WireType, WiringType);
* Autodesk.Revit.DB.Plumbing.Pipe Autodesk.Revit.Creation.Document.NewPipe(XYZ, Connector, PipeType)
* Autodesk.Revit.DB.Plumbing.Pipe Autodesk.Revit.Creation.Document.NewPipe(Connector, Connector, PipeType)
* Autodesk.Revit.DB.Element.Parameter[System.String]
* Autodesk.Revit.DB.PointOnEdge.PointOnEdge(Reference, double)
* Autodesk.Revit.DB.CurtainGridLine.Move(XYZ)
* Autodesk.Revit.DB.Line.get\_Bound(XYZ, XYZ)
* Autodesk.Revit.DB.Line.get\_Unbound(XYZ, XYZ)
* Autodesk.Revit.DB.Definitions.Create(System.String, ParameterType)
* Autodesk.Revit.DB.Definitions.Create(System.String, ParameterType, bool)
* Autodesk.Revit.DB.Definitions.Create(System.String, ParameterType, bool, System.Guid%)
* Autodesk.Revit.DB.ViewSheet.Views
* Autodesk.Revit.DB.Analysis.EnergyAnalysisOpening.OriginatingElementId
* Autodesk.Revit.DB.Analysis.EnergyAnalysisSpace.SpatialElementId
* Autodesk.Revit.DB.Analysis.EnergyAnalysisSurface.OriginatingElementId
* Autodesk.Revit.DB.ViewSheet.GetAllProjectRevisionIds()
* Autodesk.Revit.DB.ViewSheet.GetAdditionalProjectRevisionIds()
* Autodesk.Revit.DB.ViewSheet.SetAdditionalProjectRevisionIds(ICollection<ElementId>)
* Autodesk.Revit.DB.LayerModifier.Seperator
* Autodesk.Revit.DB.Units.GetDisplayUnitType()
* Autodesk.Revit.DB.Units.SetDecimalSymbolAndGrouping(DecimalSymbol, DigitGroupingSymbol, DigitGroupingAmount);
* Autodesk.Revit.DB.Units.IsValidDecimalSymbolAndGrouping(DecimalSymbol, DigitGroupingSymbol, DigitGroupingAmount)
* Autodesk.Revit.DB.FormatOptions.FormatOptions(UnitSymbolType, DisplayUnitType)
* Autodesk.Revit.DB.FormatOptions.GetRounding()
* Autodesk.Revit.DB.FormatOptions.HasRounding()
* Autodesk.Revit.DB.FormatOptions.UseGrouping
* Autodesk.Revit.DB.FormatOptions.GetName()
* Autodesk.Revit.DB.DefaultDivideSettings.GetLayout(UVGridlineType)
* Autodesk.Revit.DB.DefaultDivideSettings.SetLayout(UVGridlineType, SpacingRuleLayout)
* Autodesk.Revit.DB.DefaultDivideSettings.GetNumber(UVGridlineType)
* Autodesk.Revit.DB.DefaultDivideSettings.SetNumber(UVGridlineType, int)
* Autodesk.Revit.DB.DefaultDivideSettings.GetDistance(UVGridlineType)
* Autodesk.Revit.DB.DefaultDivideSettings.SetDistance(UVGridlineType, double)
* Autodesk.Revit.DB.FabricArea.Create(Document, Element, IList<CurveLoop>, XYZ, XYZ)
* Autodesk.Revit.DB.FabricArea.GetCurveElementIds()
* Autodesk.Revit.DB.Structure.AreaReinforcement.Create(Document, Element, IList<Curve>, XYZ)
* Autodesk.Revit.DB.Structure.AreaReinforcement.GetCurveElementIds()
* Autodesk.Revit.DB.Structure.PathReinforcement.Create(Document, Element, IList<Curve>, Boolean)
* Autodesk.Revit.DB.Structure.RebarHookType.GetDefaultHookLength(double)
* Autodesk.Revit.DB.Structure.AnalyticalModel.IsValidProjectionType(AnalyticalElementSelector, AnalyticalDirection, AnalyticalProjectionType)
* Autodesk.Revit.DB.Structure.AnalyticalModel.IsValidDirectionForProjection(AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.IsValidSelectorAndDirection(AnalyticalElementSelector, AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.IsValidProjectionDatumPlane(AnalyticalElementSelector, AnalyticalDirection, ElementId)
* Autodesk.Revit.DB.Structure.AnalyticalModel.HasAlignment(AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.GetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.SetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection, AnalyticalProjectionType)
* Autodesk.Revit.DB.Structure.AnalyticalModel.GetAlignmentMethod(AnalyticalElementSelector, AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.SetAlignmentMethod(AnalyticalElementSelector, AnalyticalDirection, AnalyticalAlignmentMethod)
* Autodesk.Revit.DB.Structure.AnalyticalModel.GetAnalyticalProjectionDatumPlane(AnalyticalElementSelector, AnalyticalDirection)
* Autodesk.Revit.DB.Structure.AnalyticalModel.SetAnalyticalProjectionDatumPlane(AnalyticalElementSelector, AnalyticalDirection, ElementId)
* Autodesk.Revit.DB.Structure.AnalyticalModel.IsSurface()
* Autodesk.Revit.DB.TableSectionData.InsertColumn(int, bool)
* Autodesk.Revit.DB.FormatValueOptions.AreValidForUnitType(UnitType)
* Autodesk.Revit.DB.UnitFormatUtils.FormatValueToString(Units, UnitType, Double, Boolean, Boolean)
* Autodesk.Revit.DB.UnitFormatUtils.FormatValueToString(Units, UnitType, Double, Boolean, Boolean, FormatValueOptions)
* Autodesk.Revit.DB.ValueParsingOptions.FormatOptions
* Autodesk.Revit.DB.Material.MaterialCagtegory
* Autodesk.Revit.DB.View.SurfaceTransparency
* Autodesk.Revit.DB.View.ShowEdges
* Autodesk.Revit.DB.View.ShowSilhouettes
* Autodesk.Revit.DB.View.SilhouetteLineStyleId
* Autodesk.Revit.DB.WorksetConfiguration.CloseAll()
* Autodesk.Revit.DB.WorksetConfiguration.OpenLastViewed()
* Autodesk.Revit.DB.IFC.IFCAnyHandle.SetAttribute(System.String name, System.Collections.Generic.ICollection<IFCAnyHandle > values);
* Autodesk.Revit.DB.IFC.IFCAnyHandle.SetAttribute(System.String name, System.Collections.Generic.ICollection<System.String > values);
* Autodesk.Revit.DB.IFC.IFCAnyHandle.SetAttribute(System.String name, System.Collections.Generic.ICollection<double> values);
* Autodesk.Revit.DB.IFC.IFCAnyHandle.SetAttribute(System.String name, System.Collections.Generic.ICollection<int> values);
* Autodesk.Revit.DB.IFC.IFCAnyHandle.SetAttribute(System.String name, System.Collections.Generic.ICollection<bool> values);
* Autodesk.Revit.UI.Selection.Selection.Elements

## Classes removed from API

The following previously deprecated classes have been removed from the API:

* Autodesk.Revit.DB.Plumbing.PipeConnectionType
* Autodesk.Revit.UI.Selection.SelElementSet
* Autodesk.Revit.DB.IFC.ExporterIFCRegistry

# API additions

## Worksharing API additions

### Workset creation

The new static method:

* Workset.Create()

creates a new Workset.

### WorksetTable operations

New static methods:

* WorksetTable.RenameWorkset() - Renames the Workset.
* WorksetTable.IsWorksetNameUnique() - Checks if the given Workset name is unique in the document.

The new method:

* WorksetTable.SetActiveWorksetId()

sets the active Workset.

## Parameter API additions

### Elements that store a reference to a parameter

The new class:

* ParameterElement

is an Element that stores information about a particular user-defined parameter in the document.  This parameter's id is used when referencing that user-defined parameter.  The ParameterElement class exposes:

* ParameterElement.GetDefinition()

The new class:

* SharedParameterElement

is an Element that stores the definition of a shared parameter which is loaded into the document.  This class is a child of ParameterElement, and exposes the following additional members:

* GuidValue - The Guid that identifies this shared parameter.
* Create()  - Creates a new shared parameter element in the document representing the parameter stored in the input ExternalDefinition.
* Lookup() - Finds the shared parameter element that corresponds to the given Guid.

## Geometry API additions

### Create loft

The new method:

* GeometryCreationUtilities.CreateLoftGeometry()

creates a solid or open shell geometry by lofting between a sequence of curve loops.

### Solid - copy

The new method:

* SolidUtils.Clone()

creates a new Solid which is a copy of the input Solid.

### Solid - create transformed

The new method:

* SolidUtils.CreateTransformed()

creates a new Solid which is the transformation of the input Solid.

### Solid - cut with half-space

The new methods:

* BooleanOperationsUtils.CutWithHalfSpace()
* BooleanOperationsUtils.CutWithHalfSpaceModifyingOriginalSolid()

produce a solid which is the intersection of the input Solid with the half-space on the positive side of the given Plane. The positive side of the plane is the side to which Plane.Normal points.  The first function creates a new Solid with the results, while the second modifies the existing solid (which must be a solid created by the application instead of one obtained from a Revit element).

### Curve - set graphics style

The new method:

* Curve.SetGraphicsStyleId()

Sets the graphics style id for the curve.  Many methods in the Revit API will not use the graphics style associated to this curve.  For example, curves used as portions of the sketch of an element will not read this property.  Newly created curve elements will not use this value either, as they inherit their graphical properties from their associated category.

### CurveLoop - transform

The new methods:

* CurveLoop.Transform()
* CurveLoop.CreateViaTransform()

allow transformation of an existing CurveLoop.  The former transforms the curves contained within the CurveLoop (modifying itself), while the latter creates a copy of the original curve loop containing the transformed curves.  In both cases, the new/modified CurveLoop is guaranteed to be valid with all consituent curves contiguous (assuming that the curves were contiguous in the input curve loop).

### FaceSecondDerivatives

The new class:

* Autodesk.Revit.DB.FaceSecondDerivatives

contains second partial derivatives of a face at a specified point.

### Face.ComputeSecondDerivatives

The new method:

* Face.ComputeSecondDerivatives

allows users to access second coordinate derivatives of a face.

## Custom Export API improvements

The API allowing custom export of 3D views and their contents has been extended to allow for more data in the output.

### IExportContext

IExportContext is now a base class for two other interfaces that support different contents on export.  This base class contains methods that are common to both the leaf interfaces. Although it is still possible to use classes deriving directly from this base interface (for backward compatibility), future applications should implement the new leaf interfaces only.

### IPhotoRenderContext

This new leaf interface should be used for processing elements in the view in the same manner that Revit's Render command does.  This is equivalent to what IExportContext allowed in Revit 2015 and earlier.

### IModelExportContext

This new interface should be used for processing elements in the view in the same manner that Revit's processes them in 3D views.  This context supports additional contents including model curves and text as shown in the 3D views.  The new interface methods are:

* **OnCurve** - export of a geometric curve, such as lines, arcs, Bezier curves, etc.
* **OnPolyline** - export of a polyline element
* **OnPoint** - export of a point element
* **OnLineSegment** - export of a tessellated line segment
* **OnPolylineSegments** - export of a tessellated polyline segments
* **OnText** - export of text annotation elements

The methods of this interface use the following new classes:

|  |  |
| --- | --- |
| ModelCurveNode | A base class of output nodes that represent various model curves. |
| CurveNode | An output node that represents a model curve. |
| PolylineNode | An output node that represents a 3D polyline. |
| PointNode | An output node that represents a 3D point |
| LineSegment | An output node that represents a tessellated line segment |
| PolylineSegment | An output node that represents a tessellated polyline segments. |
| TextNode | An output node representing a text annotation object. |
| FormatedTextRun | A structure that defines a single run of a formatted text. |
| LineProperties | A structure that provides access to pen properties of exported lines/curves |

## Export API additions

### DWFExportOptions

The new property:

* DWFExportOptions.ExportTexture

sets an option indicating whether to export textures from Revit to 3D DWF files.

### BaseExportOptions

The new property:

* BaseExportOptions.PreserveCoincidentLines

sets an option indicating whether the export process to DWG, DXF or DGN preserves coincident lines during export.

There is a newly supported string value for:

* BaseExportOptions.LayerMapping

The value "DGN7" can be used only for DGN export.

## Dimension API additions

### Dimension class

The following new members have been added to support adjustment of the text location and corresponding leaders of a given single dimension:

* Dimension.Origin  - returns the origin of the dimension (the middle point of the dimension line that makes up the dimension).
* Dimension.LeaderEndPosition - a read/write property representing the position of the dimension's leader end point.
* Dimension.TextPosition - a read/write property representing the position of the dimension text's drag point.
* Dimenson.IsTextPositionAdjustable() - indicates if this dimension is supported to get and set TextPosition/LeaderPosition.
* Dimension.ResetTextPosition() - resets the text position of the dimension to the initial position determined by its type and parameters.

### DimensionSegment

The following new members have been added to support adjustment of the text location and corresponding leaders of a given dimension segment in a multi-segment dimension:

* DimensionSegment.LeaderEndPosition - a read/write property representing the position of the dimension segment's leader end point.
* DimensionSegment.TextPosition - a read/write property representing the position of the dimension segment's text's drag point.
* DimensionSegment.IsTextPositionAdjustable() - indicates if this dimension segment is supported to set/get TextPosition/LeaderPosition
* DimensionSegment.ResetTextPosition() - resets the text position of the segment to the initial position determined by its type and parameters.

## Reinforcement API additions

### RebarContainer

The new element RebarContainer represents an aggregation of multiple Rebar sets.  At this time it can only be created via the API.  The advantages of using a RebarContainer are:

* Defining new types of rebar distributions not possible with the Revit user interface
* Improve rebar performance by combining multiple rebar sets into the definition of a single element

A RebarContainer element contains a collection of RebarContainerItem objects.  RebarContainerItem is a new class offering an API similar to that of the Rebar element.  RebarContainerItems can be created directly or duplicated from the properties of an existing Rebar element.  RebarContainer has support for iteration of the items directly from the RebarContainer object.

#### RebarContainer API

RebarContainer offers several collections of APIs similar to those offered by Rebar for management of rounding settings, presentation in a given view, and host.  In addition, it offers the following members for management of the container and its items:

* RebarContainer.Create() - Creates a new instance of a RebarContainer element within the project.
* RebarContainer.AppendItemFromRebar() - Appends an Item to the RebarContainer. Fills its data on base of the Rebar.
* RebarContainer.AppendItemFromCurves() - Appends an Item to the RebarContainer. Fills its data on base of the Rebar.
* RebarContainer.AppendItemFromRebarShape() - Appends an Item to the RebarContainer. Fills its data on base of the Rebar.
* RebarContainer.AppendItemFromCurvesAndShape() - Appends an Item to the RebarContainer. Fills its data on base of the Rebar.
* RebarContainer.RemoveItem() - Removes Item from the RebarContainer.
* RebarContainer.ClearItems() - Clears all the Items stored in this RebarContainer element.
* RebarContainer.Contains() - Checks if the RebarContainer has this item as one of its members.
* RebarContainer.ItemsCount - The count of Items in this RebarContainer.

#### RebarContainerItem API

RebarContainerItem offers many of the same APIs that Rebar offers, along with a few new members that allow management of the item within its parent container:

* RebarContainerItem.SetFromRebar() - Set an instance of a RebarContainerItem element according to the parameters list
* RebarContainerItem.SetFromCurves() - Set an instance of a RebarContainerItem element according to the parameters list.
* RebarContainerItem.SetFromRebarShape() - Set an instance of a RebarContainerItem element, as an instance of a RebarShape.
* RebarContainerItem.SetFromCurvesAndShape() - Set an instance of a RebarContainerItem element according to the parameters list.
* RebarContainerItem.BarTypeId -  The identifier of the rebar bar type.

#### RebarContainerType API

The new class:

* RebarContainerType

represents the type element used in the generation of a RebarContainer.

#### RebarContainer parameters management

The new class:

* RebarContainerParameterManager

allows an application to manage and apply parameter overrides to a RebarContainer parameter.  Normally parameters of the RebarContainer are derived from the parameters of the individual items it contains (if the parameter exists and has the same value in all items, it will exist and have the same value in the RebarContainer, while if the parameters are different in the individual items, the parameter will display without a value).   Overrides allow an application to set a different value for a given parameter, or to add new shared parameters directly to the individual RebarContainer element.

The method:

* RebarContainer.GetParametersManager

returns an object used to manage parameters of the RebarContainer.

The following methods are provided by the RebarContainerParameterManager class:

* RebarContainerParameterManager.AddOverride - Adds an override for the given parameter as its value will be displayed for the RebarContainer element.
* RebarContainerParameterManager.RemoveOverride - Removes an overridden value from the given parameter.
* RebarContainerParameterManager.ClearOverrides - Clears any overridden values from all parameters of the associated RebarContainer element.
* RebarContainerParameterManager.SetOverriddenParameterReadonly - Sets this overridden parameter to be read-only.
* RebarContainerParameterManager.SetOverriddenParameterModifiable - Sets this overridden parameter to be modifiable.
* RebarContainerParameterManager.IsOverriddenParameterModifiable - Checks if overridden parameter is modifiable.
* RebarContainerParameterManager.IsParameterOverridden - Checks if the parameter has an override.
* RebarContainerParameterManager.IsRebarContainerParameter - Checks if the parameter is a RebarContainer parameter
* RebarContainerParameterManager.AddSharedParameterAsOverride - Adds a shared parameter as one of the parameter overrides stored by this RebarContainer element.

### Bent Fabric Sheets

It is possible using the Revit 2016 API to create a fabric sheet bent along bending curves.  This is not possible in the Revit user interface.

It is not possible to convert a Fabric Sheet between flat and bent.

The following members allow creation and modification of bend fabric sheets:

* FabricSheet.Create(.., CurveLoop BendProfile) - creates a bent fabric sheet driven by the input BendProfile
* FabricSheet.GetBendProfile() - gets the curves that drive the shape of bent fabric sheet
* FabricSheet.SetBendProfile() - modifies the curves that drive the shape of bent fabric sheet
* FabricSheet.GetBendProfileWithFillets() - gets the curves that drive the shape of bent fabric sheet.including automatically generated fillets if they exist
* FabricSheet.IsBent - identifies if the fabric sheet is bent or flat
* FabricSheet.BentFabricBendDirection - read/write to control which set of wires will be bent
* FabricSheet.bentFabricWiresOrientation - read/write to control the bars' orientation
* FabricSheet.bentFabricLongitudinalCutLength - read/write to control a cut to be applied - if the sheet should be shortened and the amount

### PathReinforcement additions

The new static method:

* PathReinforcement.Create() - Creates a new PathReinforcement object from an array of curves and given Rebar Shape id.

New methods:

* PathReinforcement.GetOrCreateDefaultRebarShape() - Creates a new RebarShape object with a default name or returns existing one which fulfills PathReinforcement bending data requirements
* PathReinforcement.IsAlternatingLayerEnabled() - checks if the alternating bars exist in the Path Reinforcement instance.
* PathReinforcement.IsValidRebarShapeId() - checks if the ElementId corresponds to valid RebarShape for use in Path Reinforcement.
* PathReinforcement.IsValidPrimaryBarOrientation() - checks the orientation is valid for primary bars.
* PathReinforcement.IsValidAlternatingBarOrientation() - checks the orientation is valid for alternating bars.

New properties:

* PathReinforcement.PrimaryBarShapeId -  The RebarShape element that defines the shape of the primary bars of the PathReinforcement.
* PathReinforcement.PrimaryBarOrientation - The orientation of the primary bars.
* PathReinforcement.AlternatingBarShapeId - The RebarShape element that defines the shape of the alternating bars of the PathReinforcement.
* PathReinforcement.AlternatingBarOrientation - The orientation of the alternating bars.

### Reinforcement Settings API additions

The new methods:

* ReinforcementSettings.GetReinforcementAbbreviationTag()
* ReinforcementSettings.GetReinforcementAbbreviationTags()
* ReinforcementSettings.SetReinforcementAbbreviationTag()

provide access to the settings related to tagging abbreviations for area or path reinforcement.

### Reinforcement rounding API additions

The new members:

* RebarRoundingManager.TotalLengthRoundingMethod
* RebarRoundingManager.SegmentLengthRoundingMethod
* RebarRoundingManager.ApplicableTotalLengthRoundingMethod
* RebarRoundingManager.ApplicableSegmentLengthRoundingMethod
* FabricRoundingManager.TotalLengthRoundingMethod
* FabricRoundingManager.ApplicableTotalLengthRoundingMethod

provide access to the rounding method applied to different values found in reinforcement lengths.  The new property:

* FormatOptions.RoundingMethod

specifies the rounding method used to for specific format options.  This property is currently only supported for Rebar parameters. FormatOptions objects used in other contexts must contain the default rounding method (Nearest).

## Structural Analytical Model API additions

### Member Forces for Analytical Model

The new class:

* MemberForces

defines the internal forces and moments applied to the start or end of an AnalyticalModelStick element.  Access these forces through the new methods:

* AnalyticalModelStick.GetMemberForces() – Gets the Member Forces associated with the element.
* AnalyticalModelStick.SetMemberForces() – Sets and adds Member Forces to the element.
* AnalyticalModelStick.RemoveMemberForces() – Removes Member Forces defined for the given position.
* AnalyticalModelStick.RemoveAllMemberForces() – Removes all Member Forces associated with the element.

### AnalyticalModelStick

New Methods:

* AnalyticalModelStick.GetLocalCoordinateSystem(XYZ point) - Get the local coordinate system in a specified point on the analytical model.
* AnalyticalModelStick.GetLocalCoordinateSystem(double parameter) - Get the local coordinate system in a specified point on the analytical model.

### AnalyticalModelSurface

New Method:

* AnalyticalModelSurface.GetLocalCoordinateSystem(XYZ point): allow to possess LCS in specified point on analytical model

Change in the SurfaceElementProjectionZ enumeration:

The enumerated value SurfaceElementProjectionZ.Center has been replaced by SurfaceElementProjectionZ.CenterOfElement.

## MEP Fabrication API

The new MEP Fabrication API allows users to connect various FabricationConfigurations to Revit. With a FabricationConfiguration, they can create and use FabricationParts in their project, and define connectors to those parts.

### ****FabricationConfiguration****

The new class FabricationConfiguration contains information about the fabrication configuration settings used by the project. Using this class, users can get and set the fabrication configuration settings for the document. They can also load and unload services, reload the fabrication configuration, get loaded services, get fabrication specifications, get material and insulation information from the configuration, and get connector information.

The new static method:

* FabricationConfiguration.GetFabricationConfiguration

gets the fabrication configuration element in the document.

Some new methods include:

* static FabricationConfiguration.GetFabricationConfiguration - Gets the fabrication configuration settings in the document.
* FabricationConfiguration.SetConfiguration() - Sets the fabrication configuration for the document.
* FabricationConfiguration.LoadServices() - Load the specified fabrication services into the project.
* FabricationConfiguration.UnloadServices() - Unload the specified fabrication services from the project.
* FabricationConfiguration.ReloadConfiguration() - Reloads the fabrication configuration from its source fabrication configuration.
* FabricationConfiguration.GetAllSpecifications() - Gets all specification identifiers in the fabrication configuration.
* FabricationConfiguration.GetAllLoadedServices() - Returns all the loaded fabrication services.

### ****FabricationService****

The new class FabricationService is part of the fabrication configuration and defines what FabricationServiceButtons can be used.

Some new properties include:

* FabricationService.Name – The name of the service.
* FabricationService.TabCount – The number of tabs in the service.

Some new methods include:

* FabricationService.GetButtonCount() – Gets the number of buttons for a given tab in the service.
* FabricationService.GetButton() – Gets the service button for a given tab index and button index from the service.

### ****FabricationPart****

The new class FabricationPart represents a fabrication component in the Autodesk Revit MEP product. Using this class, users can create, place, move and align fabrication parts in a Revit model. Users can also get or set the dimensions of the fabrication part, and get the fabrication hosted information and rod information.

Some new methods include:

* static FabricationPart.Create() – Creates a fabrication part based on a FabricationServiceButton.
* static FabricationPart.CreateHanger() – Creates a fabrication hanger on another fabrication part.
* static FabricationPart.AlignPartByConnectors() - Moves and aligns fabrication part by one of its connectors to align to another connector.
* FabricationPart.GetDimensionValue() - Gets the value of fabrication dimension.
* FabricationPart.SetDimensionValue() - Sets the fabrication dimension value.
* FabricationPart.GetHostedInfo() - Gets the fabrication hosted element information.
* FabricationPart.GetRodInfo() - Gets the fabrication rod information.

### ****FabricationServiceButton****

The new class FabricationServiceButton defines a button used in a FabricationService. A fabrication service button defines what items to use for different conditions.

Some new methods include:

* FabricationServiceButton.GetConditionLowerValue() – Gets the condition lower value for a given condition.
* FabricationServiceButton.GetContionUpperValue() – Gets the condition upper value for a given condition.
* FabricationServiceButton.IsAHanger()-Checks if the fabrication service button is a hanger.

### ****Other Fabrication Classes****

The following fabrication-based classes are also added to Revit:

* FabricationPartType - Defines the type of a FabricationPart.
* FabricationRodInfo – Gives rod information for a FabricationPart.
* FabricationHostedInfo – Contains hosting information for a FabricationPart and provides the ability to disconnect from the host.
* FabricationConnectorInfo – Contains information about the connectors of a FabricationPart.
* FabricationUtils – Provides a new method to check the validity of connections.
* FabricationDimensionDefinition – Contains information about a fabrication dimension.
* FabricationConfigurationInfo – Contains information about the properties of a FabricationConfiguration.
* ConfigurationReloadInfo – Contains results from reloading a FabricationConfiguration.
* ConnectionValidationInfo – Contains connection-related warnings generated by reloading a FabricationConfiguration.

### Other Fabrication API related changes

#### Connector API

Several new members were added to the Connector class:

* Connector.Id - A unique identifier to identify this connector.
* Connector.GetFabricationConnectorInfo()  - Gets fabrication connectivity information from a connector.
* Connector.GetMEPConnectorInfo() - Gets MEP connector information from a connector.

In addition, the new static method:

* Pipe.IsPipingConnector()

checks if the given connector is a valid piping connector.

## View3D additions

Several new methods provide support for switching camera targets and for switching between perspective and isometric for a 3D view:

* View3D.CanResetCameraTarget() - Checks whether the camera target can be reset for this view.
* View3D.ResetCameraTarget() - Resets the camera target to the center of the field of view.
* View3D.CanToggleBetweenPerspectiveAndIsometric() - Checks whether the view can toggle between Perspective and isometric.
* View3D.ToggleToPerspective() - Toggles the view to perspective.
* View3D.ToggleToIsometric() - Toggles the view to isometric.

## ScheduleDefinition

New Properties:

* ScheduleDefinition.ShowTitle – Indicates if the title will be displayed in the schedule.
* ScheduleDefinition.ShowHeaders - Indicates if the headers will be displayed in the schedule.

## StairsType API changes

The properties:

* StairsType.LeftSideSupportType
* StairsType.RightSideSupportType

 are now correctly allowed to be set when the left or right string style is set to Closed.

## ElectricalSetting API additions

The following new properties have been exposed:

* ElectricalSetting.CircuitSequence   - Accesses the circuit sequence numbering schema
* ElectricalSetting.CircuitNamePhaseA - Accesses the circuit naming by phase (Phase A Label).
* ElectricalSetting.CircuitNamePhaseB - Accesses the circuit naming by phase (Phase B Label).
* ElectricalSetting.CircuitNamePhaseC - Accesses the circuit naming by phase (Phase C Label).

## PanelScheduleView

New Methods

* PanelScheduleView.GetCellsBySlotNumber () – Returns a range of cells for the given slot number
* PanelScheduleView.CanMoveSlotTo()   - Verifies if can circuits in the source slot to the specific slot.
* PanelScheduleView.MoveSlotTo() - Move the circuits in the source slot to the specific slot.

## FamilyInstance additions

The new method:

* FamilyInstance.HasModifiedGeometry() - Identifies if the geometry of this FamilyInstance has been modified from the automatically generated default.

New properties and methods introduced for reading the information about spatial calculation point(s) directly from family instances:

* FamilyInstance.HasSpatialElementCalculationPoint - Identifies if this instance has a single SpatialElementCalculationPoint used as the search point for Revit to identify if the instance is inside a room or space.
* FamilyInstance.HasSpatialElementFromToCalculationPoints - Identifies if this instance has a pair of SpatialElementCalculationPoints used as the search points for Revit to identify if the instance lies between up to two rooms or spaces. The points determine which room or space is considered the "from" and which is considered the "to" for a family instance which connects two rooms or spaces, such as a door or window.
* FamilyInstance.GetSpatialElementCalculationPoint() - Gets the location of the calculation point for this instance.
* FamilyInstance.GetSpatialElementFromToCalculationPoints() - Gets the locations for the calculation points for this instance. For a family instance which connects two rooms or spaces, such as a door or window, the points determine which room or space is considered the "from" and which is considered the "to".

## Category API additions

The new static methods:

* Category.GetCategory(Document, ElementId)
* Category.GetCategory(Document, BuildInCategory)

act as a shortcut to lookup and return a Category object.

## Family API additions

### Family.GetFamilyTypeParameterValues method

Returns all applicable values for a **FamilyType** parameter of a family.

The values are Element Ids of all family types that match the category specified by the definition of the given parameter. The elements are either of class **ElementType** or **NestedFamilyTypeReference**. The second variant is for the types that are nested in families and thus are not accessible otherwise.

### NestedFamilyTypeReference class

A class representing a proxy element for a nested family type.

In Revit, this element represents a value of a **FamilyType** parameter of a loaded family. Each such element corresponds to a nested FamilyType element in the original family document where the family was defined. This element stores only basic information about the nested FamilyType, such as the name of the Type, name of the Family, and a Category.

It is possible to obtain a set of applicable elements of this class for a particular FamilyType parameter of a family by calling Family.**GetFamilyTypeParameterValues**.

### Creation of adaptive component instances

The new constructor:

* FamilyInstanceCreationData(Autodesk::Revit::DB::FamilySymbol^ symbol, System::Collections::Generic::IList<Revit::DB::XYZ^>^ adaptivePoints)

creates an instance of a class wrapping the arguments used to place adaptive components.  This may be more efficient than placing individual adaptive components one-by-one.

## ComponentRepeater additions

New Methods

* ComponentRepeater.CanElementBeRepeated() - Determines whether an element can be repeated using the RepeatElements method.
* ComponentRepeater.RemoveRepeaters() - Removes component repeaters from the document, but leaves the individual repeated components in their respective locations and hosted on their original hosts.

## Point cloud API additions

The new method:

* PointCloudType.GetPath()

gets the path of the link source from which the points are loaded.

The new property:

* Application.PointCloudsRootPath

gets the root path for point cloud files.  This root path is used by Revit to calculate relative paths to point cloud files.

## Application additions

### Application.BackgroundColor

The new property:

* Application.BackgroundColor

allows read and write of the background color to use for model views in this session.

### Application.OptimizePerformanceDuringRedraw

The new property:

* Application.OptimizePerformanceDuringRedraw

enables or disables the option to allow view manipulation during redraw.

### Application.IsLoggedIn

The new static property:

* Application.IsLoggedIn

checks if the user is logged in from this session to their Autodesk account.

### Application.LoginUserId

The new property:

* Application.LoginUserId

returns the user id of the user currently logged in.  This is an internal id used by Autodesk to represent the logged in user.  This user id is in human unrecognizable form. In conjunction with the Autodesk Exchange Store Entitlement REST API, a publisher of an Autodesk Exchange Store application can verify if the current user has purchased their app from the store. For more information about the Exchange Store, please refer to [www.autodesk.com/developapps](http://www.autodesk.com/developapps).

### Application.IsSubscriptionUpdate

The new property:

* Application.IsSubscriptionUpdate

checks if the running Revit is a subscription update.

### Application.ShowGraphicalWarningHangerDisconnects

The new property

* Application.ShowGraphicalWarningHangerDisconnects

indicates whether or not to show the graphical warnings for Fabrication Hanger disconnects.

## Thin lines options

A utility class **ThinLinesOptions** is added that contains the setting related to the Thin Lines options which affects the display in the UI.

The static property:

* ThinLinesOptions.AreThinLinesEnabled

defines if the 'Thin Lines' setting is on or off in this session.

## 3D view API additions

### ViewNavigationToolSettings

The new class

* ViewNavigationToolSettings

carries settings related to the View Cube and other view navigation tools.   There is one element carrying these settings in each document.  Use:

* ViewNavigationToolSettings.GetViewNavigationToolSettings(Document)

to obtain it.

The methods:

* ViewNavigationToolSettings.GetHomeCamera() - Returns an object of type HomeCamera which has information about the camera and view.
* ViewNavigationToolSettings.IsHomeCameraSet() - Checks if the home view is set in the settings.

### HomeCamera

The new class HomeCamera contains information about the camera and view for the Home view orientation stored in the model.

New properties:

* HomeCamera.EyePosition
* HomeCamera.Center
* HomeCamera.UpDirection
* HomeCamera.Pivot
* HomeCamera.LeftAngleOfFieldOfView
* HomeCamera.RightAngleOfFieldOfView
* HomeCamera.TopAngleOfFieldOfView
* HomeCamera.BottomAngleOfFieldOfView
* HomeCamera.OrthogonalProjectionWidth
* HomeCamera.OrthogonalProjectionHeight
* HomeCamera.ViewId

## UI API additions

### UIDocument.SaveAs

The new class:

* Autodesk.Revit.UI.UISaveAsOptions

offers options for UIDocument.SaveAs method as below.

The new overload method:

* UIDocument.SaveAs(UISaveAsOptions)

takes a UISaveAsOptions parameter to allow the dialog prompting the user to overwrite existing files to be shown or suppressed.

### TableViewUIUtils.TestCellAndPromptToEditTypeParameter

The new static method:

* TableViewUIUtils.TestCellAndPromptToEditTypeParameter(TableView, SectionType, int, int)

involves the Revit UI and operate on schedule views or MEP electrical panel schedules, and prompts the end-user to control whether a type parameter contained in the specified table cell should be allowed edited,

It replaced the method:

* TableView.IsOkToEditParam()

which has been removed completely in Revit 2016.

### SetupEnergySimulationDialog

The new class:

* Autodesk.Revit.UI.SetupEnergySimulationDialog

represents the Revit dialog which typically precedes invocation of an Energy Simulation run on the Green Building Studio server to get user selection of project and run info.

New methods and properties:

* SetupEnergySimulationDialog.Show() - Shows the SetupEnergySimulationDialog to the user as a modal dialog.  The user has the option to select the project and run name, these can be ready by the application when the user closes the dialog.
* SetupEnergySimulationDialog.ProjectId - The identifier of the project (on the Green Building Studio server) that was selected by the user.
* SetupEnergySimulationDialog.ProjectName - The project name (representing a project on the Green Building Studio server) selected or supplied by the user.
* SetupEnergySimulationDialog.RunName - The name of the analysis run that was supplied by the user.

## AddInUtility API additions

### RevitProduct.IsSubscriptionUpdate

The new property:

* RevitProduct.IsSubscriptionUpdate

allows users to check if a particular Revit is a subscription update.