# Major changes and renovations to the Revit API

# API changes

## .NET 4.5

Revit's API is now built with and requires .NET 4.5 for compilation.

## Visual C++ runtime 11 update 4 (Visual Studio 2012)

Revit is now built with and installs runtime libraries from VC11 update 4 (Visual Studio 2012).  Third party applications which include native components may want to upgrade to the same VC runtime as there is no guarantee that Revit will install any other runtime on client machines.

## Units API

A small set of units API functions introduced in Revit 2014 have been obsoleted.  Their replacements already existed in Revit 2014, and are listed here:

|  |  |
| --- | --- |
| **Obsoleted member** | **Replacement** |
| Units.GetDisplayUnitType() | Units.GetFormatOptions(UnitType.UT\_Length).DisplayUnits |
| Units.SetDigitalSymbolAndGrouping() | Units.DecimalSymbol, Units.DigitGroupingSymbol, Units.DigitGroupingAmount (setters) |
| Units.IsValidDigitalSymbolAndGrouping() | All combinations are now valid |
| DigitGroupingSymbol.Tick (enum value) | DigitGroupingSymbol.Apostrophe |
| FormatOptions(UnitSymbolType, DisplayUnitType) (constructor) | FormatOptions (DisplayUnitType, UnitSymbolType) |
| FormatOptions.GetRounding() | FormatOptions.Accuracy |
| FormatOptions.HasRounding() | All display units have accuracy values |
| FormatOptions.UseGrouping | FormatOptions.UseDigitGrouping |
| FormatOptions.GetName() | UnitUtils.GetTypeCatalogString() |
| FormatValueOptions.IsFormatOptionsValidForUnitType() | FormatOptions.IsValidForUnitType() |
| UnitFormatUtils.FormatValueToString() | UnitFormatUtils.Format() |
| ValueParsingOptions.FormatOptions | ValueParsingOptions.GetFormatOptions() |
| FormatUtils.Format() | UnitFormatUtils.Format() |

## Parameter API changes

### Element.Parameter[String]

The indexed property Element.Parameter[String] is obsolete.  The recommended replacement is:

* Element.GetParameters(String)- looks up all of the parameters matching the input name on the given element (it is possible that multiple parameters coming from different sources can have the same name).
* Element.LookupParameter(String) -attempts to find a parameter on this Element whose name matches the input.

### Element.Parameters

The collection returned from this property now includes more parameters.   Previously, it returned only parameters that Revit showed in the Properties Palette, but now it also includes parameters available in schedules and tags but not shown in the Properties Palette.  This can cause behavioral changes for existing applications because some of the newly visible parameters will have duplicate names, but different ids, with other parameters.   To get a list of just the parameters available from the Properties Palette, use Element.GetOrderedParameters().

### Shared parameter creation - description and user modifiability

Revit now allows the assignment of two new properties to shared parameters:

* Description - this value will be saved with the shared parameter definition, and for all instances of this parameter the text is used as the parameter tooltip.
* UserModifiable - this value will be saved with the shared parameter definition.  If set to false the user will see the parameter and its value as grayed out, but any API application may modify the value as needed.

The new method:

* Definitions.Create(Autodesk.Revit.DB.ExternalDefinitonCreationOptions options);

create a new shared parameter definition using an options class where any or all of the available options may be set (name and type are required, but GUID, user visible, user modifiable, and description are optional).  This method replaces all other specific versions of Definitions.Create() which are now obsolete.

Note that for shared parameters IsReadOnly can return false for shared parameters whose UserModifiable property is also false, because the value of those parameters can be modified by the API. If a parameter is governed by a formula,  IsReadOnly would return true, even if the flag for UserModifiable was set to true when the shared parameter was created.

The new properties:

* Parameter.UserModifiable
* FamilyParameter.UserModifiable

provide information about whether the flag is set to the API.

## Worksharing API changes

### WorksetConfiguration methods

The methods

* WorksetConfiguration.CloseAll()
* WorksetConfiguration.OpenLastViewed()

have been obsoleted.  Instead of using these methods after constructing the WorksetConfiguration, use the constructor for WorksetConfiguration accepting one of the following options:

* WorksetConfigurationOption.OpenAllWorksets
* WorksetConfigurationOption.CloseAllWorksets
* WorksetConfigurationOption.OpenLastViewed

### SynchronizeWithCentralOptions.CompactCentralFile

The property

* SynchronizeWithCentralOptions.CompactCentralFile

duplicated the property

* SynchronizeWithCentralOptions.Compact

but was not settable.  It has been removed completely in Revit 2015.

## Settings classes are now Elements

Several pre-existing classes:

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

are now subclasses of Element.  You can use limited Element related functionality on these members (for example, get their ids to check them out in a local workshared model, or add ExtensibleStorage to them).  FilteredElementCollector will now return these types if they pass the associated filters.

## Family API changes

### FamilyBase class removed

The FamilyBase class has been completely removed as a base class of Family.  Family now inherits directly from Element.  The members which belonged to FamilyBase have moved to Family:

* FamilyBase.FamilyCategory
* FamilyBase.StructuralMaterialType

Code that accesses elements as Family and uses these members should not need to be changed, but must be recompiled due to the removal of the intermediate level.

Code that accesses elements as FamilyBase will need to be updated to use Family instead.  In Revit, any FamilyBase element found will actually be a Family element and can be cast accordingly.

### Family.Symbols

The property Family.Symbols has been obsoleted - use

* Family.GetFamilySymbolIds()

as a replacement.

### Family.CurtainPanelHorizontalSpacing and Family.CurtainPanelVerticalSpacing

Access to these properties is now supported only from an owner family obtained from a curtain panel family document.

An exception will result attempting to access these properties from a family which is not the owner family of the document.

Code which previously tried to use these properties from a non-owner family will now need to edit the family in order to access this information.

## View API changes

### View display settings

The new class:

* ViewDisplayModel

contains settings related to graphics display, such as transparency, silhouette settings, hidden lines, and smooth edges. Access these settings from:

* View.GetViewDisplayModel()
* View.SetViewDisplayModel()

This access replaces direct properties of View:

* View.SurfaceTransparency
* View.ShowEdges
* View.ShowSilhouettes
* View.SilhouetteLineStyleId

which have been marked obsolete.

### ViewSheet members related to Revisions

Several ViewSheet methods related to Revisions have been replaced with new names.  The original members have been obsoleted.  This table lists the changes:

|  |  |
| --- | --- |
| **Obsoleted member** | **Replacement** |
| IList<ElementId> GetAllProjectRevisionIds() | IList<ElementId> GetAllRevisionIds() |
| ICollection<ElementId> GetAdditionalProjectRevisionIds() | ICollection<ElementId> GetAdditionalRevisionIds() |
| SetAdditionalProjectRevisionIds(ICollection<ElementId>) | SetAdditionalRevisionIds(ICollection<ElementId>) |

## Structural API changes

### AnalyticalModel members obsoleted

Several methods of AnalyticalModel have been marked obsolete:

* AnalyticalModel.IsValidProjectionType()
* AnalyticalModel.IsValidDirectionForProjection()
* AnalyticalModel.IsValidSelectorAndDirection()
* AnalyticalModel.IsValidProjectionDatumPlane()
* AnalyticalModel.HasAlignment()
* AnalyticalModel.HasExtension()
* AnalyticalModel.GetAnalyticalProjectionType()
* AnalyticalModel.SetAnalyticalProjectionType()
* AnalyticalModel.SetAlignmentMethod()
* AnalyticalModel.GetAnalyticalProjectionDatumPlane()
* AnalyticalModel.GetAnalyticalProjectionDatumPlane()
* AnalyticalModel.IsSurface()

Their replacements are more specialized additions in AnalyticalModelSurface and the new classes AnalyticalModelStick and AnalyticalModelColumn, described below.

To check if an analytical model is a surface type of element, try to downcast it to AnalyticalModelSurface.

### AreaReinforcement API changes

The method

* AreaReinforcement.GetCurveElementIds()

has been obsoleted - use

* AreaReinforcement.GetBoundaryCurveIds()

as a replacement.

A new overload has been introduced for

* AreaReinforcement.Create()

accepting ids for the type, bar type and hook type.  The previous version has been obsoleted.

### PathReinforcement API changes

A new overload has been introduced for

* PathReinforcement.Create()

accepting ids for the type, bar type and hook types.  The previous version has been obsoleted.

### FabricArea API changes

The method

* FabricArea.GetCurveElementIds()

has been obsoleted - use

* FabricArea.GetBoundaryCurveIds()

as a replacement.

A new overload has been introduced for

* FabricArea.Create()

accepting ids for the type and sheet type.  The previous version has been obsoleted.

### RebarHookType API changes

The method

* RebarHookType.GetDefaultHookLength()

has been obsoleted.  Use

* RebarHookType.GetDefaultHookExtension()

as a replacement.

### Miscellaneous changes

#### Removed classes

The following classes:

* FabricReinSpanSymbolControl
* RebarSystemSpanSymbolControl

have been obsoleted.  They did not have specific API access available in the first place.

#### Removed enumerated value

The enumerated value BoundaryConditionsType.Nothing was removed from the enumerated type.  This was a default value which could never be returned or set for any element.

## Selection API changes

The new method

* Selection.SetElementIds(ICollection<ElementId> ids)

supports setting of the selected elements in the active document.

The following items are now obsolete:

* SelElementSet (class)
* Selection.Elements

## MEP API changes

### Connector properties removed

The properties

* Connector.JointType
* Connector.GenderType

did not match properties available in the Revit user interface, and thus should not have been exposed in the API.  They have been removed completely in Revit 2015.

### Creation.Document.NewWire()

The method NewWire() has been obsoleted and replaced by the new method

* Wire.Create()

The new method supports creation of straight, arc, chamfer, and spline wires.

### Related Wire API additions

Several new methods have been introduced to edit the layout of an existing wire:

* Wire.AppendVertex(XYZ vertexPoint) - Appends one vertex to the end of the wire.
* Wire.GetVertex(int index) - Returns the vertex position at the given index.
* Wire.InsertVertex(int index, XYZ vertexPoint) -  Inserts a new vertex before the given index.
* Wire.RemoveVertex(int index) - Removes one vertex from the given index.
* Wire.SetVertex(int index, XYZ vertexPoint) - Sets one vertex's position from the given index
* Wire.AreVertexPointsValid(IList<XYZ> vertexPoints, Connector startConnector, Connector endConnector) - Identifies if the given vertex points are valid for the wire or not.
* Wire.ConnectTo(Connector startConnectorTo, Connector endConnectorTo) -  Connects a wire to another wire

New properties on wire:

* NumberOfVertices  - Number of vertices of the wire, including the start and end point
* WiringType - The wiring type(Arc or Chamfer) for the wire.

## IFC API changes

### Obsoleted functions, classes, and enums

* The IFCDoorStyleOperation and IFCWindowStyleOperation enums have been obsoleted and replaced entirely by their .NET equivalents.
* IFCExporterUtils.GetWallHeightDirection is no longer necessary, as all internal Revit walls have a direction of (0,0,1).
* IFCTypeInfo has been deprecated, and is no longer used by IFC Export.
* ExporterIFC.GetBuilding has been deprecated.  The IfcBuilding handle is now expected to be stored in the exporting application if needed.

### Changed functions, classes, and enums

Several API routines that returned HashSet<IFCAnyHandle> now return an ICollection<IFCAnyHandle> instead.

### New functions, classes, and enums

IFCFile has a new Read() function that returns the number of errors and warnings reported by the toolkit during the read operation.  This is in addition to the existing Read() function.

IFCAnyHandle now has SetAttribute() functions for each data type to reduce the use of the IFCData class.

IFCLegacyStairOrRamp has two new functions, GetBoundaryLines() and GetWalkLines(), that return the curve loops associated with the boundaries and walking lines of the legacy stair or ramp, respectively.

## Energy Analysis API changes

### ElementId properties in EnergyAnaysisDetailModel contents

The properties:

* EnergyAnalysisSpace.SpatialElementId
* EnergyAnalysisOpening.OriginatingElementId
* EnergyAnalysisSurface.OriginatingElementId

have been obsoleted.  Because the EnergyAnaysisDetailModel is intended to be generated at a single point in time, and is not persistently updated as changes are made to the original elements used to generate the energy analysis model, the model has been decoupled from direct relationships with the generating Revit elements.  As replacements, the API now offers:

* EnergyAnalysisSpace.CADObjectUniqueId
* EnergyAnalysisOpening.CADObjectUniqueId
* EnergyAnalysisOpening.CADLinkUniqueId
* EnergyAnalysisSurface.CADObjectUniqueId
* EnergyAnalysisSurface.CADLinkUniqueId

These properties offer the unique id of the related elements instead. In the case of potential relationships with linked model elements, two unique ids are available.

As a reminder, the EnergyAnaysisDetailModel is intended to be generated and used immediately - none of the data or relationships that it contains is updated as changes are made to the corresponding Revit model.  Dispose of the generated EnergyAnaysisDetailModel using the Destroy() method as soon as you have extracted the needed information.

## Miscellaneous API changes

### Material API

The property Material.MaterialCagtegory has been obsoleted and replaced by Material.MaterialCategory.

### TableSectionData.InsertColumn(int index, bool bCreateCellData)

This method has been obsoleted - use

* TableSectionData.InsertColumn(int index)

as a replacement.

### BoundaryConditions

This class has moved from namespace Autodesk.Revit.DB to Autodesk.Revit.DB.Structure.

### BuiltInCategory.OST\_MassWindow

This enumerated type value has been renamed to OST\_MassGlazing.

### ElementIntersectsElementFilter

Previously, the ElementIntersectsElementFilter would report the target element as an "intersection" with itself.  This behavior has been changed; the filter will no longer pass the target element.

### ExtensibleStorageFilter

The ExtensibleStorageFilter class has moved from namespace Autodesk.Revit.DB to Autodesk.Revit.DB.ExtensibleStorage.

### MeshTriangle

The MeshTriangle class no longer inherits from APIObject.

### CurtainGridLine.Move()

The method CurtainGridLine.Move() has been obsoleted.  Use ElementTransformUtils.MoveElement() to accomplish the same result.

### CurveLoop.CreateViaThicken()

Previously, when this function couldn't create a compatible CurveLoop, it would return null.  It now throws an exception in this situation.

## Obsolete API removal

The following obsolete APIs and classes have been removed:

* Autodesk.Revit.Creation.Application.NewArc(Plane, Double, Double, Double)
* Autodesk.Revit.Creation.Application.NewArc(XYZ, Double, Double, Double, XYZ, XYZ)
* Autodesk.Revit.Creation.Application.NewArc(XYZ, XYZ, XYZ)
* Autodesk.Revit.Creation.Application.NewEllipse(XYZ, Double, Double, XYZ, XYZ, Double, Double)
* Autodesk.Revit.Creation.Application.NewHermiteSpline(IList<XYZ>, Boolean)
* Autodesk.Revit.Creation.Application.NewHermiteSpline(IList<XYZ>, Boolean, XYZ, XYZ)
* Autodesk.Revit.Creation.Application.NewLine(XYZ, XYZ, Boolean)
* Autodesk.Revit.Creation.Application.NewLineBound(XYZ, XYZ)
* Autodesk.Revit.Creation.Application.NewLineUnbound(XYZ, XYZ)
* Autodesk.Revit.Creation.Application.NewNurbSpline(IList<XYZ>, DoubleArray, DoubleArray, Int32, Boolean, Boolean)
* Autodesk.Revit.Creation.Application.NewNurbSpline(IList<XYZ>, IList<Double>)
* Autodesk.Revit.Creation.Application.NewSATExportOptions()
* Autodesk.Revit.Creation.Document.NewAreaReinforcement(Element, CurveArray, XYZ)
* Autodesk.Revit.Creation.Document.NewBeamSystem(CurveArray, Level)
* Autodesk.Revit.Creation.Document.NewBeamSystem(CurveArray, Level, XYZ, Boolean)
* Autodesk.Revit.Creation.Document.NewBeamSystem(CurveArray, SketchPlane)
* Autodesk.Revit.Creation.Document.NewBeamSystem(CurveArray, SketchPlane, XYZ, Boolean)
* Autodesk.Revit.Creation.Document.NewPathReinforcement(Element, CurveArray, Boolean)
* Autodesk.Revit.Creation.Document.NewRebarBarType()
* Autodesk.Revit.Creation.Document.NewRoomTag(Room, UV, View)
* Autodesk.Revit.Creation.Document.NewTopographySurface(IList<XYZ>)
* Autodesk.Revit.Creation.Document.NewTruss(TrussType, SketchPlane, Curve)
* Autodesk.Revit.Creation.Document.NewViewSheet(FamilySymbol)
* Autodesk.Revit.Creation.FamilyItemFactory.NewDividedSurface(Reference)
* Autodesk.Revit.Creation.ItemFactoryBase.NewSketchPlane(PlanarFace)
* Autodesk.Revit.Creation.ItemFactoryBase.NewSketchPlane(Plane)
* Autodesk.Revit.Creation.ItemFactoryBase.NewSketchPlane(Reference)
* Autodesk.Revit.DB.Architecture.BoundaryLocationType
* Autodesk.Revit.DB.Architecture.StairsRun.ExtensionBelowBase
* Autodesk.Revit.DB.Curve.EndParameter[Int32]
* Autodesk.Revit.DB.Curve.EndPoint[Int32]
* Autodesk.Revit.DB.Curve.EndPointReference[Int32]
* Autodesk.Revit.DB.Curve.Transformed[Transform]
* Autodesk.Revit.DB.Dimension.Label
* Autodesk.Revit.DB.DividedSurfaceData
* Autodesk.Revit.DB.Document.AnnotationSymbolTypes
* Autodesk.Revit.DB.Document.BeamSystemTypes
* Autodesk.Revit.DB.Document.ContFootingTypes
* Autodesk.Revit.DB.Document.CurtainSystemTypes
* Autodesk.Revit.DB.Document.DeckProfiles
* Autodesk.Revit.DB.Document.Delete(Element)
* Autodesk.Revit.DB.Document.DimensionTypes
* Autodesk.Revit.DB.Document.ElectricalEquipmentTypes
* Autodesk.Revit.DB.Document.Export(String, String, ViewSet, SATExportOptions)
* Autodesk.Revit.DB.Document.FasciaTypes
* Autodesk.Revit.DB.Document.FindReferencesWithContextByDirection(XYZ, XYZ, View3D)
* Autodesk.Revit.DB.Document.FloorTypes
* Autodesk.Revit.DB.Document.GridTypes
* Autodesk.Revit.DB.Document.GutterTypes
* Autodesk.Revit.DB.Document.LevelTypes
* Autodesk.Revit.DB.Document.LightingDeviceTypes
* Autodesk.Revit.DB.Document.LightingFixtureTypes
* Autodesk.Revit.DB.Document.MechanicalEquipmentTypes
* Autodesk.Revit.DB.Document.RebarBarTypes
* Autodesk.Revit.DB.Document.RebarCoverTypes
* Autodesk.Revit.DB.Document.RebarHookTypes
* Autodesk.Revit.DB.Document.RebarShapes
* Autodesk.Revit.DB.Document.RoofTypes
* Autodesk.Revit.DB.Document.RoomTagTypes
* Autodesk.Revit.DB.Document.SlabEdgeTypes
* Autodesk.Revit.DB.Document.SpaceTagTypes
* Autodesk.Revit.DB.Document.SpotDimensionTypes
* Autodesk.Revit.DB.Document.TextNoteTypes
* Autodesk.Revit.DB.Document.TitleBlocks
* Autodesk.Revit.DB.Document.TrussTypes
* Autodesk.Revit.DB.Document.ViewSheetSets
* Autodesk.Revit.DB.Document.WallTypes
* Autodesk.Revit.DB.Edge.EndPointReference[Int32]
* Autodesk.Revit.DB.Edge.Face[Int32]
* Autodesk.Revit.DB.Element.GetDividedSurfaceData()
* Autodesk.Revit.DB.Element.GetMaterialArea(Material)
* Autodesk.Revit.DB.Element.GetMaterialVolume(Material)
* Autodesk.Revit.DB.Element.Group
* Autodesk.Revit.DB.Element.Level
* Autodesk.Revit.DB.Element.Materials
* Autodesk.Revit.DB.IFC.IFCDoorWindowInfo
* Autodesk.Revit.DB.Line.Bound[XYZ, XYZ]
* Autodesk.Revit.DB.Line.Unbound[XYZ, XYZ]
* Autodesk.Revit.DB.Material.CutPattern
* Autodesk.Revit.DB.Material.GetCutPatternColor()
* Autodesk.Revit.DB.Material.GetCutPatternId()
* Autodesk.Revit.DB.Material.GetRenderAppearance()
* Autodesk.Revit.DB.Material.RenderAppearance
* Autodesk.Revit.DB.Material.SetRenderAppearance(Asset)
* Autodesk.Revit.DB.Material.SurfacePattern
* Autodesk.Revit.DB.MEPSystem.IsDefaultSystem
* Autodesk.Revit.DB.ParameterFilterElement.AllCategoriesFilterable(ICollection<ElementId>)
* Autodesk.Revit.DB.Plumbing.PipeConnectionType
* Autodesk.Revit.DB.Plumbing.PipeSettings.ElbowAngleIncrement
* Autodesk.Revit.DB.Plumbing.PipeType.ConnectionType
* Autodesk.Revit.DB.PointCloudInstance.GetPoints(PointCloudFilter, Int32)
* Autodesk.Revit.DB.SaveAsOptions.Rename
* Autodesk.Revit.DB.Settings.VolumeCalculationSetting
* Autodesk.Revit.DB.SketchPlane.Plane
* Autodesk.Revit.DB.SketchPlane.PlaneReference
* Autodesk.Revit.DB.StairsEditScope.Commit()
* Autodesk.Revit.DB.Structure.FabricArea.Create(Document, Element, IList<CurveLoop>, XYZ)
* Autodesk.Revit.DB.Structure.FabricArea.SetFabricLocation(FabricLocation)
* Autodesk.Revit.DB.Structure.FabricArea.SetFabricSheetTypeId(ElementId)
* Autodesk.Revit.DB.Structure.FabricArea.SetMajorSheetAlignment(FabricSheetAlignment)
* Autodesk.Revit.DB.Structure.FabricArea.SetMinorSheetAlignment(FabricSheetAlignment)
* Autodesk.Revit.DB.Structure.FabricSheet.SheetTypeId
* Autodesk.Revit.DB.Structure.FabricSheetType.PhysicalMaterialAsset
* Autodesk.Revit.DB.Structure.RebarShape.GetHookAngle(Int32)
* Autodesk.Revit.DB.Structure.RebarShape.GetHookOrientation(Int32)
* Autodesk.Revit.DB.Structure.RebarShapeDefinitionBySegments.AddBendDefaultRadius(Int32, Int32, RebarShapeBendAngle)
* Autodesk.Revit.DB.Structure.RebarShapeDefinitionBySegments.AddBendVariableRadius(Int32, Int32, RebarShapeBendAngle, ElementId, Boolean)
* Autodesk.Revit.DB.Transform.Reflection[Plane]
* Autodesk.Revit.DB.Transform.Rotation[XYZ, XYZ, Double]
* Autodesk.Revit.DB.Transform.Translation[XYZ]
* Autodesk.Revit.DB.View.CutColorOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.CutLinePatternOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.CutLineWeightOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.GetVisibility(Category)
* Autodesk.Revit.DB.View.ProjColorOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.ProjLinePatternOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.ProjLineWeightOverrideByElement[ICollection<ElementId>]
* Autodesk.Revit.DB.View.SetVisibility(Category, Boolean)
* Autodesk.Revit.DB.View3D.SectionBox
* Autodesk.Revit.DB.VolumeCalculationOptions
* Autodesk.Revit.DB.VolumeCalculationSetting
* Autodesk.Revit.Utility.AssetPropertyReference.Value

start cut

# Major API additions

## View API changes

### Active graphical view

The new property

* UIDocument.ActiveGraphicalView

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.

### Sketchy lines settings

The new methods

* View.GetSketchyLines()
* View.SetSketchyLines()

allow full control over the Sketchy Lines settings for a given view.

## Default Type API

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

### Family Types

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

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

### Non-family Types

These members provide read and write access to the default type for a non-Family element type:

* Document.GetDefaultElementTypeId() - Gets the default element type id for a given non-Family element type.
* Document.SetDefaultElementTypeId() - Sets the default
* Document.IsDefaultElementTypeIdValid() - Checks whether the element type id is valid for a given non-Family element type.

## Structural API additions

### Reinforcement numbering

The new classes:

* NumberingSchema
* NumberingSchemaType

are used to define how objects of certain kind and scope are to be organized for the purpose of numbering/tagging them.  Each NumberingSchema controls numbering of elements of one particular kind.  Instances of NumberingSchema are also elements and there is always only one of each type in every Revit document. Available types of all built-in numbering schemas are enumerated in NumberingSchemaTypes class.

In this release NumberingSchema applies only to the built-in types matching elements of these categories:

* Rebar
* Fabric Reinforcement

### Reinforcement in parts

Reinforcement and Rebar is now allowed to be hosted in Parts if those Parts come from a structural layer of a valid reinforcement host.

The methods:

* Rebar.CreateFromCurves()
* Rebar.CreateFromCurvesAndShape()
* Rebar.CreateFromRebarShape()
* AreaReinforcement.Create()
* PathReinforcement.Create()
* FabricArea.Create()
* FabricSheet.Create()

accept compatible parts as host elements.

The new method:

* RebarHostData.IsValidHost()

identifies if a proposed host elements is valid for reinforcement.

### Rebar presentation mode

The new presentation mode capabilities allow the user to specify how rebar sets are shown in a given view. Bar presentation schemes simplify the view while maintaining an identifiable footprint in which the rebar set is placed.  In the API, the following members have been added to support this capability:

* Rebar.SetPresentationMode()
* Rebar.GetPresentationMode()
* Rebar.ClearPresentationMode()
* Rebar.HasPresentationOverrides()
* Rebar.SetBarHiddenStatus()
* Rebar.IsBarHidden()
* Rebar.FindMatchingPredefinedPresentationMode()
* Rebar.IsRebarInSection()
* Rebar.CanApplyPresentationMode()
* RebarInSystem.SetPresentationMode()
* RebarInSystem.GetPresentationMode()
* RebarInSystem.ClearPresentationMode()
* RebarInSystem.HasPresentationOverrides()
* RebarInSystem.SetBarHiddenStatus()
* RebarInSystem.IsBarHidden()
* RebarInSystem.FindMatchingPredefinedPresentationMode()
* RebarInSystem.IsRebarInSection()
* RebarInSystem.CanApplyPresentationMode()

The default settings for bar presentation can be accessed from

* ReinforcementSettings.RebarPresentationInView
* ReinforcementSettings.RebarPresentationInSection

### Place FabricSheet directly in host

The new members:

* FabricSheet.Create()
* FabricSheet.PlaceInHost()
* FabricSheet.GetSheetLocation()
* FabricSheet.IsSingleFabricSheetWithinHost()
* FabricSheet.HostId
* FabricSheet.FabricLocation
* FabricSheet.CoverOffset
* FabricSheet.FabricHostReference
* FabricSheet.IsCoverOffsetValid
* FabricArea.RemoveFabricReinforcementSystem()
* RebarHostData.GetFabricSheetsInHost()

support the new Revit capability where single instances of fabric sheets can be placed precisely to reinforce sections of concrete walls or floors.

### Creating default reinforcement types

The new methods:

* AreaReinforcementType.CreateDefaultAreaReinforcementType()
* PathReinforcementType.CreateDefaultPathReinforcementType()
* FabricAreaType.CreateDefaultFabricAreaType()
* FabricSheetType.CreateDefaultFabricSheetType()
* FabricWireType.CreateDefaultFabricWireType()
* RebarBarType.CreateDefaultRebarBarType()
* RebarHookType.CreateDefaultRebarHookType()

create a default element type for the given reinforcement class.  This is useful if there is no existing type element of this kind in the document.

### Miscellaneous reinforcement API additions

#### Create reinforcement based on host boundary

The new overloads:

* AreaReinforcement.Create(Document, Element, XYZ, ElementId, ElementId, ElementId)
* FabricArea.Create(Document, Element, XYZ, ElementId, ElementId)

create new reinforcement area elements automatically related to the host's boundary.

#### Rebar shape family

The property:

* RebarShape.ShapeFamilyId

gets the rebar shape family id.

#### RebarHostCategory enumerated type

The type of host for rebars. This type matches the value returned by the new parameter BuiltInParameter.REBAR\_HOST\_CATEGORY.

### AnalyticalModel API additions

#### AnalyticalModel coordinate system

The new method:

* AnalyticalModel.GetLocalCoordinateSystem()

returns the local coordinate system from analytical model element.

#### AnayticalModelSurface additions

The AnalyticalModelSurface class has been extended with several new members:

* AlignmentMethod
* ProjectionZ
* ProjectionPlaneZ
* HasExtension
* BottomExtensionMethod
* TopExtensionMethod
* BottomExtension
* TopExtension
* BottomExtensionPlaneId
* TopExtensionPlaneId

These new members support checking and manipulating alignment, projection and extension of AnalyticalModelSurface elements such floors, slabs and walls.

#### Stick and column elements

The new class:

* AnalyticalModelStick

represents a stick in the structural analytical model (A beam, brace or column).  This class contains several members to check and manipulate the alignment, projection and extension properties:

* GetAlignmentMethod()
* GetProjectionY()
* GetProjectionZ()
* GetProjectionPlaneY()
* GetProjectionPlaneZ()
* SetProjection()

The new class:

* AnalyticalModelColumn

represents an analytical model of structural column.  It is a subclass of AnalyticalModelStick.  This class contains members to check and manipulate extension properties:

* BaseExtensionMethod
* TopExtensionMethod
* BaseExtensionPlaneId
* TopExtensionPlaneId
* BaseExtension
* TopExtension

### Loads and Boundary Conditions API

#### LoadBase class

The new properties:

* OrientTo
* HostElementId
* WorkPlaneId

provide the ability to read and change how the load is oriented to the associated host or related work plane.

#### BoundaryConditions class

The new methods:

* BoundaryConditions.GetOrientTo()
* BoundaryConditions.SetOrientTo()

access the Boundary Conditions element orientation feature that allows orientation of boundary conditions to the local coordinate system of the associated analytical model.

The new method:

* BoundaryConditions.GetDegreesOfFreedomCoordinateSystem()

 gets the definition of the coordinate system that is used by the element's translation and rotation parameters (e.g. X Translation or Z Rotation).

The new method:

* BoundaryConditions.GetBoundaryConditionsType()

accesses the Boundary Conditions type (Point, Line or Area).

#### Structural Section Parameters

Revit now supports defined data structures to represent standard structural section shapes.  In support of this feature a hierarchy of classes were introduced to the API:

|  |  |
| --- | --- |
| **Class** | **Represents** |
| StructuralSection | The base class for StructuralSection specific classes, designed to provide common parameters and ability to differentiate between different structural section shapes. |
| StructuralSectionRectangular | The base class for rectangular sections. |
| StructuralSectionRound | The base class for round sections. |
| StructuralSectionCParallelFlange | C-channel Parallel Flange structural section. |
| StructuralSectionCSlopedFlange | C-channel Sloped Flange structural section. |
| StructuralSectionHotRolled | Hot rolled structural sections. |
| StructuralSectionIParallelFlange | I-shape Parallel Flange structural section. |
| StructuralSectionISlopedFlange | I-shape Sloped Flange structural section. |
| StructuralSectionISplitParallelFlange | I-split Parallel Flange structural section. |
| StructuralSectionISplitSlopedFlange | I-split Sloped Flange structural section. |
| StructuralSectionIWelded | I-shape Welded structural section. |
| StructuralSectionIWideFlange | I-shape Wide Flange structural section. |
| StructuralSectionLAngle | L-angle Flange structural section. |
| StructuralSectionPipeStandard | Pipe section. |
| StructuralSectionRectangleHSS | Parameterized rectangle HSS structural section. |
| StructuralSectionRectangleParameterized | Parameterized rectangle structural section. |
| StructuralSectionRectangularBar | Rectangular Bar structural section. |
| StructuralSectionRoundBar | Round Bar structural section. |
| StructuralSectionRoundHSS | Pipes known as Round HSS (HollowStructuralSection). |
| StructuralSectionStructuralTees | Structural Tees structural section. |

Only beams, braces and structural columns can have a structural section.  At the level of the Family, these members:

* Family.HasStructuralSection()
* Family.StructuralSectionShape

identify if the family carries a structural section and its shape.

At the level of the FamilySymbol, these members:

* FamilySymbol.HasStructuralSection()
* FamilySymbol.GetStructuralSection()
* FamilySymbol.SetStructuralSection()

provide access to the specific parameterized structural section for a given FamilySymbol.

The new method:

* LabelUtils.GetStructuralSectionShapeName()

returns the user-visible name of structural section shape.

### StructuralFramingUtils

The methods:

* StructuralFramingUtils.CanSetEndReference()
* StructuralFramingUtils.IsEndReferenceValid()
* StructuralFramingUtils.RemoveEndReference()
* StructuralFramingUtils.GetEndReference()
* StructuralFramingUtils.SetEndReference()

support setting / getting / removing the end references for family instances of a structural framing type.

Family instances need to be non-concrete and joined at the given end.  As the new end reference can be set appropriate face of the joined element at the given end.  The setback value will be changed as a result of any reference change.

The methods:

* StructuralFramingUtils.DisallowJoinAtEnd()
* StructuralFramingUtils.AllowJoinAtEnd()
* StructuralFramingUtils.IsJoinAllowedAtEnd()

 support disallowing / allowing structural framing elements to join at the end to others.

Family instances need to be of a structural framing category. If the framing element is already joined at the end, and becomes disallowed to join, it will become disconnected.  If the framing element end is allowed to join and if that end is near other elements it will become joined.

## Revisions

Revit 2015 introduces new API classes and members for accessing project Revisions, their settings and associated Revision Clouds.

### RevisionSettings class

The new RevisionSettings class allows an application to read and modify the project-wide settings that affect Revisions and Revision Clouds.

The new property

* RevisionSettings.RevisionAlphabet

determines the characters used to populate the Revision Number parameter of alphabetic Revisions.

The new property

* RevisionSettings.RevisionCloudSpacing

determines the sizing of the cloud graphics for Revision Clouds in the project.

The new property

* RevisionSettings.RevisionNumbering

determines whether revision numbers for the project are determined on a per sheet or a whole project basis.

### Revision class

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.

The new method

* Revision.GetAllRevisionIds()

provides an ordered list of all of the Revisions in the document.

The new method

* Revision.ReorderRevisions()

allows the ordering of the Revisions within the project to be changed.

The new method

* Revision.Create()

creates a new Revision in the document.

The data associated with a Revision, and its associated settings within the project, can be read and modified through the following new properties:

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

The new methods

* Revision.CombineWithNext()
* Revision.CombineWithPrevious()

allow an application to combine a specified Revision with the next or previous Revision in the model.  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.

The new method

* ViewSheet.GetRevisionNumberOnSheet()

provides access to the Revision Number for a Revision when the numbering in the project is by sheet.

### RevisionCloud class

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.

The new method

* RevisionCloud.Create()

allows an application to create a new RevisionCloud in a specified view based on a series of lines and curves.

The new property

* RevisionCloud.RevisionId

allows an application to read an modify the Revision associated with the RevisionCloud.

The new method

* RevisionCloud.IsRevisionIssued()

allows an application to easily check whether a RevisionCloud is associated with a Revision that has already been issued.

The new method

* RevisionCloud.GetSheetIds()

allows an application to obtain the ids of the ViewSheets where the RevisionCloud may appear (either because the RevisionCloud is placed directly on the ViewSheet or because the RevisionCloud is visible in some View on the ViewSheet).

The new ViewSheet method

* ViewSheet.GetRevisionCloudNumberOnSheet()

provides access to the Revision Number for a RevisionCloud when the numbering in the project is by sheet.

### Revision cloud geometry

The property:

* Element.Geometry

has been enhanced to return geometry from RevisionCloud elements.  This will return the actual curved lines that make up the cloud.

The new method

* RevisionCloud.GetSketchCurves()

allows an application to read the Curves that form the RevisionCloud's sketch.  This will return the sketched curves that define the basic outline of the cloud and not the arcs that Revit attaches to these curves to create the cloud appearance.

## Parameters API additions

### Parameter order

Revit now allows users to reorder parameters within their groups for a given family, ensuring that the parameters are presented to the user in the most logical order.  Several API changes have been introduced related to this.

The new methods

* FamilyManager.GetParameters()
* Element.GetOrderedParameters()

returns the parameters associated to family types or elements in the specified order.   Note that for Element.GetOrderedParameters(), the returned collection only includes parameters that are shown in the Properties Palette (unlike Element.Parameters).

The new method

* FamilyManager.ReorderParameters(IList<FamilyParameter> parameters)

reorders the family parameters within the family according to the specified input.

The new method

* FamilyManager.SortParameters(ParametersOrder order)

sorts the family parameters according to the desired automatic sort order.

### Family parameter creation - description

The new method:

* FamilyManager.SetDescription(FamilyParameter familyParameter, String description);

sets the description for a family parameter.

The new property:

* Definition.Description

gets the stored tooltip description of the parameter definition.

## Wall API additions

### Stacked wall

Several new members added to the Wall class provide support for reading information about stacked wall and stacked wall members.

The new method

* Wall.GetStackedWallMemberIds()

gets the sub walls which belong to a stacked wall, with the ids returned in order from bottom to top.

The new properties

* Wall.IsStackedWall
* Wall.IsStackedWallMember
* Wall.StackedWallOwnerId

identify if the wall is a stacked wall, a member of a stacked wall, and if the wall is a member of the stacked wall, the id of the stacked wall that owns this wall.

### Wall Function

The new property

* WallType.Function

provides read/write access to the Function property of wall types.

## Schedule API additions

### Schedule filters

As schedules now support up to 8 applied filters, the following methods now allow up to 8 filters to be applied:

* ScheduleDefinition.AddFilter()
* ScheduleDefinition.InsertFilter()
* ScheduleDefinition.SetFilter()
* ScheduleDefinition.SetFilters()

### ScheduleDefinition.GrandTotalTitle

The new property

* ScheduleDefinition.GrandTotalTitle

provides the ability to customize the name of grand total title row for a schedule.

### Images in schedules

Images can now be added to schedules via parameters defined as type "Image".  In schedule views, the image parameter will display the path of the ImageType, but the image itself displays in ScheduleSheetInstances placed on a sheet.  The value of the "Image" parameters is an ElementId representing an ImageType element.

The new ImageType class is a subclass of ElementType representing a type containing an image. Instances of this type may also be displayed on 2D views or sheets directly.

The new method:

* ImageType.Create()

provides the ability to create a new ImageType element and loads a copy of the image into it.

The new members:

* ImageType.Reload()
* ImageType.ReloadFrom()
* ImageType.IsLoadedFromFile()
* ImageType.Path

provide the ability to manage the contents of the image, and reload it from its original path location or a new location.

The new members

* ViewSchedule.ImageRowHeight
* ViewSchedule.RestoreImageSize()
* ViewSchedule.HasImageField()

provide the ability to affect the size and display characteristics of schedules that contain images.

## IFC API additions

### IFC import options and operations

The new method:

* RevitLinkType.CreateFromIFC()

creates a new linked IFC type representing an IFC model opened for reference.  Once created, you can place instance(s) of this type using regular RevitLinkInstance methods.

The new method:

* Application.OpenIFCDocument(string, IFCImportOptions)

supports different options for import and create of a new document based on an IFC file:

* IFCImportOptions.Action(open or link)
* IFCImportOptions.Intent (parametric or reference)
* IFCImportOptions.AutoJoin (applies to parametric import only)

### ImporterIFC new properties and functions

ImporterIFC offers new members to assist with the implementation of custom IFC importers:

* ImporterIFC.Document  - returns the document associated with the IFC file.
* ImporterIFC.GetOptions() - returns the options names and values set for the current IFC import.

### Built-in parameter changes

BuiltInParameter.IFC\_\*\_GUID values (IFC\_GUID, IFC\_TYPE\_GUID, IFC\_PROJECT\_GUID, IFC\_BUILDING\_GUID, and IFC\_SITE\_GUID) are no longer required to be unique, and are schedulable.

BuiltInParameter.IFC\_TYPE\_GUID has a default English value of "Type IfcGUID" to distinguish it from BuiltInParameter.IFC\_GUID.

## Import API

### DirectShape

The new classes:

* DirectShape
* DirectShapeType
* DirectShapeLibrary

offer the ability to create imported geometry elements directly in the project document.  The geometry can include closed solids or meshes.   The geometry will be validated to ensure that it is valid for Revit use.

The created elements must be assigned to a category.  This grants the elements a collection of available parameters and some limited behaviors.

### TessellatedShapeBuilder

The new classes:

* TessellatedShapeBuilder
* TessellatedFaces
* TessellatedShapeBuilderResult

can be used create solid, shell, or polymeshes bounded by a set of connected planar facets, created by adding TessellatedFace objects one by one.  The utility includes some ability to heal imprecisions and discontinuities in the inputs, and offers some diagnostics regarding geometry that is too imprecise to be used.

## External Resources Service API

This new framework allows add-ins to provide Revit with external content obtained from anywhere.  Add-ins implementing a server for this type can obtain their external content from the web, an external database, or another application. The Revit user can browse the external content locally and select appropriate content to use in their models.

In this release, only some types of Revit external content are supported as External Resources:

* Keynotes
* Assembly classification codes
* Revit links

### IExternalResourceServer

The new interface:

* IExternalResourceServer

allows developers to provide resources from an external source. Revit will call IExternalResourceServer.LoadResource(), and the server will provide the data for the requested resource.

### IExternalResourceUIServer

The new interface:

* IExternalResourceUIServer

gives Revit a list of the resources handled by the server. The resources will appear in Revit's UI when the Revit user browses for the appropriate link type. Servers can also provide custom error-handling UI. This allows servers to handle cases Revit could not know about. For example, if the network is down and the server cannot access its resources, the server can put up a detailed error message explaining the problem.

### ExternalResourceReference

The new class:

* ExternalResourceReference

contains identifying information for resources which come from external servers. Each external server link will contain an ExternalResourceReference. The ExternalResourceReference contains the id of the server which provided the resource. The class also contains a string-to-string map which contains the actual identity information for the reference. Servers can define their own conventions for naming and identifying resources.

## Keynote and Assembly Code API

Revit 2015 introduces a number of new classes to give applications access to the keynote and assembly code data used within a Revit model.

Because there are a number of similarities in the way the keynote data and the assembly code data are structured, many of the operations can be accessed through base classes that provide common functionality for key-based tree-structured data.

### KeyBasedTreeEntryTable

The new class:

* KeyBasedTreeEntryTable

represents a collection of key-based tree entries stored within the Revit model, such as the keynote or assembly code table. There are two subclasses - **KeynoteTable** for the keynote table, and **AssemblyCodeTable** for the assembly code table.

The methods:

* KeynoteTable.GetKeynoteTable()
* AssemblyCodeTable.GetAssemblyCodeTable()

are static methods which allow access to the current table.

The methods:

* KeyBasedTreeEntryTable.LoadFrom()
* KeyBasedTreeEntryTable.Reload()

allow the user to reload the keynote or assembly code table. LoadFrom() allows the table to be reloaded from a new location.

The method:

* GetKeyBasedTreeEntries()

allows access to the data in the table.

### KeyBasedTreeEntry

The class:

* KeyBasedTreeEntry

represents an individual entry within a key-based tree. It provides properties for the key and parent key. The subclasses, KeynoteEntry and ClassificationEntry, provide properties and methods specific to the keynote table and the assembly code table. Keynotes have access to the keynote text, while ClassificationEntries have access to the level, category Id, and description.

## Energy analysis API additions

### gbXML export options

The new property:

* GBXMLExportOptions.ExportEnergyModelType

determines the type of analysis used when producing the export gbXML file for the document.  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 2014.
* BuildingElement - Energy model based on analysis of building element volumes.

### BuildingEnvelopeAnalyzer Class

The new class:

* BuildingEnvelopeAnalyzer

analyzes which elements are part of the building envelope (the building elements exposed to the outside). This class uses a combination of ray-casting and flood-fill algorithms in order to find the building elements that are exposed to the outside of the building. This method can also look for the bounding building elements for enclosed space volumes inside the building. Options for the analysis include:

* AnalyzeEnclosedSpaceVolumes - Whether or not to analyze interior connected regions inside the building forming enclosed space volumes.
* GridCellSize - The cell size for the uniform cubical grid used when analyzing the building envelope.
* OptimizeGridCellSize - Whether or not to use the exact value for the cell size or let the analyzer optimize the cell size based on the specified grid size

## BrowserOrganization API

The new class

* BrowserOrganization

contains settings for grouping, sorting, and filtering of items in the project browser.

New methods

* BrowserOrganization.AreFiltersSatisfied() - Determines if the given element satisfies the filters defined by the browser organization.
* BrowserOrganization.GetFolderItems() -  Returns a collection of leaf FolderItemInfo objects each containing the given element Id.

New static Methods

* GetCurrentBrowserOrganizationForViews() - Gets the BrowserOrganization that applies to the Views section of the project browser.
* GetCurrentBrowserOrganizationForSheets() - Gets the BrowserOrganization that applies to the Sheets section of the project browser.

New properties

* SortingOrder - The sorting order if sorting of items is applicable in the browser.
* SortingParameterId - The id of the parameter used to determine the sorting order of items in the browser.

The new class

* FolderItemInfo

contains data for each folder item in the organization settings of the project browser including folder parameter Id and folder name.

New properties

* ElementId -The folder parameter Id.
* Name - The folder name .

# Minor API additions

## Application API additions

3 new properties have been added to Application to retrieve file paths from revit.ini:

1. DefaultIFCProjectTemplate: the template set in the IFC import options to override the default project template.
2. ExportIFCCategoryTable: the path and file name to the Revit category to IFC entity mapping table for export.
3. ImportIFCCategoryTable: the path and file name to the IFC entity to Revit category mapping table for import.

## Document API additions

### Document.IsDetached

The new property

* Document.IsDetached

identifies if a workshared document is opened as detached.

### Document.DocumentWorksharingEnabled

This event is raised when Revit has just enabled worksharing in the document.   Handlers of this event are permitted to make modifications to any document (including the active document), except for documents that are currently in read-only mode.

### UIDocument operations and additions

The new method

* UIDocument.RequestViewChange()

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.

The new methods

* UIDocument.PostRequestForElementTypePlacement()
* UIDocument.PromptToPlaceElementTypeOnLegendView()

places a request on Revit's command queue for the user to place instances of the specified ElementType.   The former is for general use, the latter is specifically for legend views. This does not execute immediately,  but instead 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.

The new method

* UIDocument.PromptToPlaceViewOnSheet()

Prompts the user to place a specified view onto a sheet. Set  allowReplaceExistingSheetViewport to true to allow the user to replace the existing viewport.

The new method

* UIDocument.PromptToMatchElementType()

prompts the user to select instance elements to change them to the input type.

## Link API additions

### External Resource compatibility

The Revit Link API has been updated - all methods which took a ModelPath argument now have new versions which take an ExternalResourceReference argument. These methods can also work with files from disk. ExternalResourceReference.CreateLocalResource() can be used to create an ExternalResourceReference corresponding to a local file.

The older Revit link methods have not been deprecated; they are still available for local links.

### RevitLinkType.AttachmentType

This property is now writable, and can be toggled between Attachment and Overlay.

## Category API additions

### Category.CategoryType

The new property

* Category.CategoryType

determines if the category is shown in the Visibility/Graphics settings grouped with the model, annotation, or analytical model categories.   Note that import categories are also "model" but will be shown separately in the dialog. Some categories not shown in the dialog and will return Internal for the category type.

## ElementType API additions

### ElementType.FamilyName

The new property

* ElementType.FamilyName

contains the localized string describing the family in which this ElementType belongs. For family symbols, this will be the name of the associated Family. For system family types, this will be the name used to group related types, such as "Oval Duct" or "Curtain Wall".

### ElementType duplicating events

The events:

* Application.ElementTypeDuplicating
* Application.ElementTypeDuplicated

allow you to subscribe to an event to be notified when Revit is just about to duplicate an element type, and after Revit has finished duplicating an element type.

## Material API additions

The new static method

* Material.IsNameUnique(Document document, String name)

validates whether a proposed material name is unique in document.  This will be used to confirm the validity of the name before creating a new material via Material.Create() or Material.Duplicate().

## View API additions

### View.IsAssemblyView

The new property

* View.IsAssemblyView

identifies if the view is an assembly view.

### View.Title

The new property

* View.Title

 returns the view title. This consists of the view name plus other modifiers, such as the view type, sheet number, area scheme, and/or assembly type, depending on the specifics of the view.

### Categories hidden status

The following properties are now settable:

* View.AreModelCategoriesHidden
* View.AreAnnotationCategoriesHidden
* View.AreAnalyticalModelCategoriesHidden
* View.AreImportCategoriesHidden

Setting these properties allow an application to toggle Model Categories, Annotation Categories, Analytical Model Categories, or Import Categories visibility.

### Drafting view creation

The new method

* ViewDrafting.Create()

allows an application to create a new drafting view in the model with a specified ViewFamilyType.

### Orient 3D view

The new method

* View3D.OrientTo(XYZ forwardDirection)

supports reorienting the 3D view to align with the forward direction.  This is an alternate method to reorient the view using typical Revit calculations for other related parameters in ViewOrientation3D.

### Reference callouts and sections

The new class ReferenceableViewUtils provides utility methods that allow an application to manage reference views such as reference sections or reference callouts.

The new methods

* ReferenceableViewUtils.ChangeReferencedView
* ReferenceableViewUtils.GetReferencedViewId

allows an application to read and change the view referenced by a reference view (such as a reference section or reference callout).

## Family API additions

### FamilyManager.IsUserAssignableParameterGroup

The new method

* FamilyManager.IsUserAssignableParameterGroup()

provide the ability to identify the given built-in parameter group is user assignable for family parameter or not.

### LocationPoint.Rotation for view-specific family instances

The property:

* LocationPoint.Rotation

now returns the rotation angle in the plane of the view, for view-specific family instances such as detail components.  Previously the angle was measured from an axis outside the plane of the view.

### PromptForFamilyInstancePlacement() option for Air Terminals on Ducts

The new overload method

* UIDocument.PromptForFamilyInstancePlacement(Autodesk.Revit.DB.FamilySymbol, Autodesk.Revit.UI.PromptForFamilyInstancePlacementOptions)

allows an application to prompt the user to place instances of a specified family symbol interactively.  The options class supports a setting requiring the mode to be set to place an air terminal family instance directly on a duct (the option is either set to be on or off, the user cannot toggle this during placement).

### Family loading events

The events:

* Application.FamilyLoadingIntoDocument
* Application.FamilyLoadedIntoDocument

allow you to subscribe to an event to be notified when Revit is just about to load a family into a document, and after Revit has just finished loading the family.

## Geometry API additions

### Curve

The new method:

* Curve.CreateReversed()

creates a new curve that has the same shape, but has its orientation reversed.

The new method:

* Curve.CreateOffset()

creates a new curve that is offset from the original curve by a certain distance.  The offset direction is determined by the normal of the curve at any given point.

### CurveLoop

The new method:

* CurveLoop.CreateViaThicken()

creates a new closed curve loop by thickening the input open curve loop with respect to a given plane.

The new method:

* CurveLoop.CreateViaOffset()

creates a new CurveLoop that is an offset of the original CurveLoop.  This is effectively done by offsetting each Curve in the CurveLoop and trimming the ends to form a new continuous CurveLoop.

### Face

The new method:

* Face.GetEdgesAsCurveLoops()

returns a list of closed curve loops that correspond to the edge loops of the face. Curves in each curve loop correspond to individual edges.

### CurveElement

The new method:

* CurveElement.SetGeometryCurve()

explicitly sets the geometry of the curve element with the option to not affect the geometry of any currently joined curve elements. After the curve geometry is set, other curves may autojoin to the new curve geometry.

The new method:

* CurveElement.SetSketchPlaneAndCurve()

 sets the sketch plane and the curve for the CurveElement simultaneously (allowing the SketchPlane to be successfully modified in a way that would be incompatible if set separately).  This method will not affect the geometry of any current joined curve elements.

### FreeFormElement

The new method:

* FreeFormElement.UpdateSolidGeometry()

updates the geometry of the FreeFormElement to the given shape preserving References to the existing geometry where possible.

## Material API additions

### Material.UseRenderAppearanceForShading

The new property:

* Material.UseRenderAppearanceForShading

determines if the material's appearance in a shaded view should be driven by the settings of the render appearance, or driven by the material's graphics properties.

## Print API additions

### Region edges mask coincident lines

The new property:

* PrintParameters.MaskCoincidentLines

indicates whether to mask coincident lines when printing.

## Connector API additions

### New connector properties

The new properties:

* Connector.AllowsSlopeAdjustments
* Connector.Utility
* Connector.Description

provide project level read access to connector properties which can be set in the family environment.

## Naming utilities

The method

* NamingUtils.IsValidName()

identifies if the input is valid to be used as a name of an object in Revit.

This routine checks only for prohibited characters in the string. When setting the name for an object there are other specific considerations which are checked (for example, the same name cannot be used twice for different elements of the same type). This routine does not check those conditions.

The method

* NamingUtils.CompareNames(string nameA, string nameB)

compares the input two names according to the comparison rules in Revit. The method returns a negative value if nameA comes before nameB, zero if nameA is equivalent to nameB,and a positive value if nameA comes after nameB. The method is similar to String.Compare(), but uses Revit rules for comparison.  This involves breaking the names into alphabetic and numeric tokens and comparing tokens individually.

## Dynamic Model Update additions

The new methods:

* UpdaterRegistry.EnableUpdater()
* UpdaterRegistry.DisableUpdater()
* UpdaterRegistry.IsUpdaterEnabled()

allow temporary enable and disable of Updaters.  This allows an application to control whether an updater is triggered unnecessarily based on changes the application knows about.

## Custom Exporter additions

The new method:

* CustomExporter.IsRenderingSupported()

allows an application to test that libraries necessary to support rendering and 3D exports are installed and available.

## UI API additions

### Drag & drop API

The new interface

* IControllableDropHandler

inherits from IDropHandler.  This includes an extra interface to be executed when custom data is dragged and dropped onto the Revit user interface.  This interface is different from IDropHandler in that it allows the handler to verify whether the drop event can be executed on the given view.

 The new interface method

* IControllableDropHandler.CanExecute(UIDocument document, object data, ElementId dropViewId)

 Implement this method to inform Revit whether the drop event can be executed onto the given view.