Major changes and renovations to the Revit API 2013

.NET 4.0 for compilation

All Revit components including RevitAPI.dll and RevitAPIUI.dll are now built with the .NET 4.0 compiler. Thus, add-in projects which reference Revit DLLs must be set as .NET 4.0 Framework projects as well.

New macro environment

VSTA has been replaced by SharpDevelop. This provides identical capabilities for macro creation, editing, and debugging as well as for deployment (application-level and document-level macros) and a simple upgrade procedure for existing macros.

SharpDevelop also provides expanded IDE capabilities, .NET 4.0, extended refactoring tools, code analysis tools, a regular expression toolkit, and macro IDE add-in support.

Revit.ini registration disabled

Add-ins will no longer be read from Revit.ini. Revit will look only to .addin files in the machine-wide and user-specific locations when deciding what add-ins to load.

Document.Element properties replaced

The indexed properties:

- Document.Element[ElementId]
- Document.Element[String]

have been replaced by methods:

- Document.GetElement(ElementId)
- Document.GetElement(String)

The old indexed properties have been marked obsolete.
The behavior for the new methods is identical to what they replaced.

Replacements for use of old-style collections

ElementArray and ElementSet are now obsolete

With a few exceptions, API methods using ElementArray and ElementSet have been replaced with equivalents using .NET generic collections of ElementIds (ICollection<ElementId>).

Some APIs have been obsoleted with no replacement; these methods are batch element creation methods which dated to the time when regeneration was automatic after every database change. Now that regeneration is manual, having separate batch creation methods provides no extra benefit as multiple element creation calls can be chained together with no delay caused by regeneration. Thus these methods are obsolete:

- ItemFactoryBase.NewTextNotes()
- Creation.Document.NewRooms(List<RoomCreationData>)
- Creation.Document.NewWalls() (all overloads)

When replacing calls to these methods, you should use the corresponding creation method which creates a single element in a loop, while delaying calls to Regenerate() until the loop has concluded. Because Regenerate() is delayed, you should not make inquiries to the newly created elements' geometry as a part of the loop.

Definitions and DefinitionGroups

These classes used to inherit from base classes consisting of old-style Revit API collections. The new versions of the classes offer the same functionality, but implement System.Collections.Generic.IEnumerable<T>.

Code which referred to the older base class or its associated iterator will need to be changed to access members through iteration or the now-available LINQ methods.
In addition, code which called ForwardIterator() or ReverseIterator() on these classes will need to be changed as the new version of the class does not offer these two legacy methods.

GeometryElement.Objects

The property

- GeometryElement.Objects

has been obsoleted. GeometryElement now inherits directly from IEnumerable<GeometryObject>, and it possible to directly iterate the contents of the GeometryElement using .NET and LINQ, if needed.

Replacements for wall creation methods

New wall creation APIs have been provided as a part of the Wall class:

- static Wall Create(Document document, Curve curve, ElementId levelId, bool structural) - Creates a new rectangular profile wall within the project using the default wall style.
- static Wall Create(Document document, Curve curve, ElementId wallTypeId, ElementId levelId, double height, double offset, bool flip, bool structural) - Creates a new rectangular profile wall within the project using the specified wall type, height, and offset.
- static Wall Create(Document document, IList<Curve> profile, bool structural) - Creates a non rectangular profile wall within the project using the default wall type:
- static Wall Create(Document document, IList<Curve > profile, ElementId wallTypeId, ElementId levelId, bool structural) - Creates a non rectangular profile wall within the project using the specified wall type.

- static Wall Create(Document document, IList<Curve > profile, ElementId wallTypeId, ElementId levelId, bool structural, XYZ normal) - Creates a non rectangular profile wall within the project using the specified wall type and normal vector.

These methods replace the obsolete methods on Autodesk.Revit.Creation.Document (NewWall() (all overloads)). The NewWalls() overloads are obsoleted without dedicated replacements, as described here: ElementArray and ElementSet are now obsolete

Idling event and External Events

Idling event frequency

The Idling event has been changed to offer two different modes of behavior. The default behavior has changed; pre-existing code that needs the original behavior will need to opt-in to the non-default behavior.

In the default mode, a single raise of the event will be made each time Revit begins an idle session. Note that when the user is active in the Revit user interface, idle sessions begin whenever the mouse stops moving for a moment or when a command completes. However, if the user is not active in the user interface at all, Revit may not invoke additional idling sessions for quite some time; this means that your application may not be able to take advantage of time when the user leaves the machine completely idle for a period of time.

In the non-default mode, your application forces Revit to keep the idling session open and to make repeated calls to your event subscriber. In this mode even if the user is totally inactive the Revit session will continue to make Idling calls to your application. However, this can result in performance degradation for the system on which Revit is running because the CPU remains fully engaged in serving Idling events during the Revit application's downtime.

You can indicate the preference for the non-default Idling frequency by calling

- IdlingEventArgs.SetRaiseWithoutDelay()

each time the Idling event callback is made. Revit will revert to the default Idling frequency if this method is not called every time in your callback.

Idling event with no active document

The Idling event now is invoked when there is no document active in Revit.

External Events framework

The External Events framework is a new API framework tailored for asynchronous processing such as that which is needed to support modeless dialogs. It operates similarly to the Idling event with default frequency.
With this framework, instead of implementing an event handler and implementing the handler to do nothing when there is no activity in your dialog, instead you do the following:

1. Implement an external event by deriving from the interface:
    - IExternalEvent
2. Create an ExternalEvent with
    - ExternalEvent.Create()

3. When an event happens in your modeless UI where a Revit action needs to be taken, call
    - ExternalEvent.Raise()
4. Revit will wait for an available Idling timecycle and call the implementation of your event:
    - IExternalEvent.Execute()

This framework is more useful for modeless dialog developers because you can skip the default no-op implementation when nothing has happened, and tell Revit when there is some work you need to do in Revit. It also saves on the time required for Revit to make repeated calls to the Idling event when there is no actual work to be done.

Calling OpenAndActivateDocument during events

The UIApplication.OpenAndActivateDocument can now be invoked during events under the following conditions:

1. When invoked from an External Event, not restrictions apply except for the original active document to be free of transactions and transaction groups.
2. When invoked from regular event handlers, the following restrictions apply:
    - There must be no active document open yet in Revit
    - The event must not be nested in other events or in execution external command

Document & worksharing changes

Worksharing properties

The information required to identify a workshared file on the central or local locations have changed due to changes to RevitServer. As a result, the members

- Application.GetRevitServerNetworkHosts()
- Application.CurrentRevitServerAccelerator
- Document.WorksharingCentralFilename

replace the properties

- Application.CentralServerName
- Application.LocalServerName
- Document.GetWorksharingCentralModelPath()

The new members:

- Application.GetWorksharingCentralGUID(ServerPath serverModelPath)
- Document.WorksharingCentralGUID

provides read access to the worksharing central GUID of the given server-based model. This is applicable only to workshared models saved in Revit 2013 or later.

New overloads for Application.OpenDocumentFile() and UIApplication.OpenAndActivateDocument()

The new overloads support parameters OpenOptions and OpenOptionsForUI, respectively, to specify how a Revit document should be opened. Both options classes currently offer the ability to detach the opened document from central if applicable.

The property:

- OpenOptions.DetachFromCentralOption
  can be set to DoNotDetach (the default) or DetachAndPreserve.

The property:

- OpenOptionsForUI.DetachFromCentralOption
  can be set to DoNotDetach (the default), DetachAndPreserveWorksets or DetachAndPrompt.

BasicFileInfo

The new class

- BasicFileInfo
  offers basic information about a Revit file, including worksharing status, Revit version, username and central path. The information can be extracted without fully opening the file.

Point cloud engine API changes

Because these are new methods in a pre-existing interface, existing code will have to be modified to implement the methods.

Free()

Unlike other Point Cloud interfaces in the Revit 2012 release, this interface did not offer a method to be called when Revit is finished with it.

When Revit calls this function it indicates that the IPointCloudAccess interface is no longer going to be used and the provider of the interface can dispose of all allocated resources.

GetOffset()

Implement this method to return the offset stored in the point cloud. All points are assumed to be offset by the same offset vector. The offset should be expressed in the same units as used by the point coordinates (the scale conversion factor is not applied). The offset will be used by Revit if the user choose to place an instance relative to another point cloud (the "Auto - Origin To Last Placed" placement option).

CreatePointSetIterator()

A second overload with an additional argument – density – the desired number of points per unit area. Area is computed in native units of the point cloud.

IFC export APIs

The IFC export implementation in Revit has switched to a more generic toolkit. As a result, many of the APIs introduced in Revit 2012 to support customized IFC export have been removed and replaced, and new interfaces introduced that allow more flexibility in the data which is written to the IFC file. Autodesk recommends that any customization of IFC export begin with the implementation of the default Revit IFC exporter client, and that customizations written for 2012 be ported incrementally to support Revit 2013.

Construction Modeling changes

The following members are now obsolete:

- Part.GetDividedParents() - replaced with Part.GetSourceElementIds
- PartMaker.GetDividedElementIds() - replaced with PartMaker.GetSourceElementIds
- PartMaker.IsElementDivided() - replaced with PartMaker.IsSourceElement
- PartMaker.SetDividedElementIds() - replaced with PartMaker.SetSourceElementIds
- PartUtils.ArePartsValidForDivide -> replaced with PartUtils.AreElementsValidForDivide()

The following members are now obsolete - they are not being replaced with new members because the information they returned can be obtained by applying GetSourceElementIds recursively to source elements that are Parts.

- Part.OriginalDividedElementId
- Part.ParentDividedElementId
- PartUtils.GetDividedParent()

The following members are obsolete. Their replacements live on the PartMakerMethodToDivideVolumes class, which you can obtain for a given PartMaker using PartUtils.GetPartMakerMethodToDivideVolume():

- PartUtils.AreElementsValidIntersectingReferences()
- PartUtils.IsValidSketchPlane()
- PartUtils.SetOffsetForIntersectingReference()
- PartUtils.GetOffsetForIntersectingReference()
- PartUtils.PartMakerUsesReference()

The methods:

- AssemblyViewUtils.CreatePartList()
- AssemblyViewUtils.CreateMaterialTakeoff()
  now return ViewSchedule instead of View.

Structure changes

Rebar.GetCenterlineCurves()

The overload taking one argument is now obsolete.

The new three-argument Rebar.GetCenterlineCurves(Boolean, Boolean, Boolean) has arguments to control whether or not to:

- adjust for self-intersection
- include hooks

- include bend radius curves (fillets)

Rebar member changes

These members are now obsolete:

- Rebar.DistributionPath is replaced by Rebar.GetDistributionPath()
- Rebar.Host is replaced by Rebar.GetHostId() and SetHostId()
- Rebar.RebarShape is replaced by Rebar.RebarShapeId property
- Rebar.BarType is obsoleted. Use the inherited GetTypeId() and ChangeTypeId()

AreaReinforcement and PathReinforcement

BarDescription class

The new RebarInSystem element replaces the **BarDescription** class. This class and related methods:

- AreaReinforcement.NumBarDescriptions()
- AreaReinforcement.GetBarDescription()
- PathReinforcement.NumBarDescriptions()
- PathReinforcement.GetBarDescription()

are obsolete in 2013.

When **ReinforcementSettings.HostStructuralRebar** is true, Area and Path Reinforcement elements create individual rebar elements that participate in views and schedules. The individual **RebarInSystem** elements have the same graphical representation and parameters as **Rebar** elements, and have the same API methods for inquiry. But the **RebarInSystem** elements are almost entirely controlled by their system element (Area or Path). Therefore they are missing most of the modification methods of the Rebar class.

New methods:

- AreaReinforcement.GetRebarInSystemIds()
- AreaReinforcement.RemoveAreaReinforcementSystem()
- PathReinforcement.GetRebarInSystemIds()
- PathReinforcement.RemovePathReinforcementSystem()

MEP changes

ConnectorProfileType, PartType

ConnectorProfileType enums are renamed to:
Invalid, Round, Rectangular, Oval

PartType::UndefinedPartType is renamed to PartType::Undefined.

ConnectorElement

Subclasses of ConnectorElement: PipeConnector, DuctConnector, and ElectricalConnector, are obsolete.

- IsPrimary - moved to ConnectorElement
- AssignAsPrimary - moved to ConnectorElement
- LinkedConnector - moved to ConnectorElement as GetLinkedConnectorElement and SetLinkedConnectorElement
- Please query the parameters RBS_PIPE_CONNECTOR_SYSTEM_CLASSIFICATION_PARAM, RBS_DUCT_CONNECTOR_SYSTEM_CLASSIFICATION_PARAM, or RBS_ELEC_CIRCUIT_TYPE on ConnectorElement to obtain the PipeSystemType, DuctSystemType, and ElectricalSystemType values on the now obsolete PipeConnector, DuctConnector, and ElectricalConnector elements.

Connector element types can now be created using new methods:

- ConnectorElement.CreatePipeConnector()
- ConnectorElement.CreateDuctConnector()
- ConnectorElement.CreateElectricalConnector()
- ConnectorElement.CreateConduitConnector()
- ConnectorElement.CreateCableTrayConnector()

These methods replace the methods on Autodesk.Revit.Creation.FamilyItemFactory (NewDuctConnector, NewPipeConnector, NewElectricalConnector). Those methods are now obsolete.

Obsolete API Cleanup

Many API classes and methods previously marked Obsolete in Revit 2012 and earlier have been removed from the API.

Classes removed

- MaterialConcrete, MaterialGeneric, MaterialOther, MaterialSteel, MaterialWood - all data is now accessed via the Material class
- Autodesk.Revit.DB.ConstructionType - replaced by Autodesk.DB.Analysis.ConstructionType
- SuspendUpdating - no longer needed because Automatic regeneration mode has been removed.Autodesk.Revit.UI.Macros.ApplicationEntryPoint
- Autodesk.Revit.VSTA.ApplicationEntryPoint - replaced by Autodesk.Revit.UI.Macros.ApplicationEntryPoint
- Autodesk.Revit.VSTA.DocumentEntryPoint - replaced by Autodesk.Revit.UI.Macros.DocumentEntryPoint
- Autodesk.Revit.VSTA.IEntryPoint - replaced by Autodesk.Revit.UI.Macros.IEntryPoint
- Autodesk.Revit.VSTA.AddInIdAttribute - replaced by Autodesk.Revit.UI.Macros.AddInIdAttribute
- Autodesk.Revit.VSTA.VendorIdAttribute - replaced by Autodesk.Revit.UI.Macros.VendorIdAttribute

Methods and properties removed

- AnalysisDisplayLegend.Width - replaced by legend settings of AnalysisDisplayStyle.
- AnalysisDisplayLegendSettings.SetTextTypeId(ElementId , Document)
- AnalysisDisplayMarkersAndTextSettings.SetTextTypeId(ElementId , Document)
- AnalyticalModel methods
    - IsValidAnalyticalProjectionType(AnalyticalDirection , AnalyticalProjectionType) replaced by IsValidProjectionType

        - GetAnalyticalProjectionDatumPlane(AnalyticalDirection) replaced by GetAnalyticalProjectionDatumPlane(AnalyticalElementSelector , AnalyticalDirection)
        - IsValidDatumPlaneForProjection(AnalyticalDirection, ElementId) replaced by IsValidProjectionDatumPlane
        - SetAnalyticalProjectionDatumPlane(AnalyticalDirection , ElementId) replaced by setAnalyticalProjectionDatumPlane(AnalyticalElementSelector, AnalyticalDirection, ElementId)
        - SetAnalyticalOffset(XYZ) replaced by SetOffset
        - GetAnalyticalOffset() replaced by GetOffset
        - GetAnalyticalProjectionType(AnalyticalDirection) replaced by GetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection)
        - SetAnalyticalProjectionType(AnalyticalDirection , AnalyticalProjectionType) replaced by SetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection, AnalyticalProjectionType)
- IFCExportOptions.CurrentViewOnly
- PointCloudFilter.GetFilterOutline() - replaced by PointCloudFilterUtils.GetFilteredOutline()
- RebarHostData.HasCoverTypeForReference(Reference) - replaced by IsFaceExposed()
- SpatialFieldManager members
    - CurrentUnits replaced by AnalysisResultSchema.CurrentUnits and SpatialFieldManager.SetResultSchema()
    - UpdateSpatialFieldPrimitive(int , Autodesk.Revit.DB.Analysis.FieldDomainPoints, Autodesk.Revit.DB.Analysis.FieldValues)
    - SetUnits(System.Collections.Generic.IList<System.String> , System.Collections.Generic.IList<double>) replaced by AnalysisResultSchema.SetUnits() and SpatialFieldManager.SetResultSchema()
    - GetDescription() replaced by SpatialFieldManager.SetResultSchema() and AnalysisResultSchema.Description
    - SetDescription(System.String) replaced by AnalysisResultSchema.Description and SpatialFieldManager.SetResultSchema()
- AnnotationSymbol.AsFamilyInstance
- Area.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
- Element members
    - ObjectType replaced by GetTypeId() and ChangeTypeId()
    - SimilarObjectTypes replaced by GetValidTypes()
- Materials members
    - AddOther(System.String) replaced by Material.Create
    - AddWood(System.String) replaced by Material.Create
    - AddConcrete(System.String) replaced by Material.Create
    - AddSteel(System.String) replaced by Material.Create
    - AddGeneric(System.String) replaced by Material.Create
- PropertySetElement.Create(Document, MaterialAspect) replaced by PropertySetElement.Create(Document, StructuralAsset) and PropertySetElement.Create(Document, ThermalAsset)
- Mullion.AsFamilyInstance
- Panel.AsFamilyInstance
- Rebar.IsShownInFrontOfGeometryInView[View] - replaced by IsUnobscuredInView() and SetUnobscuredInView()
- PointOnEdge members
    - PointOnEdge(Autodesk.Revit.DB.Reference, double) replaced by PointLocationOnCurve

        - CurveParameter replaced by LocationOnCurve
    - Room.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
    - Face.MaterialElement - replaced by MaterialElementId
    - Reference members
        - Element replaced by Document.GetElement(Reference)
        - GeometryObject replaced by Element.GetGeometryObjectFromReference(Reference)
        - Transform replaced by ReferenceByContext.GetInstanceTransform() after obtaining ReferenceWithContext from Document.FindReferencesWithContextByDirection()
        - ProximityParameter replaced by ReferenceByContext.ProximityParameter after obtaining ReferenceWithContext from Document.FindReferencesWithContextByDirection()
    - PipingSystem members
        - FluidType, FluidType is now a property of the system type. Get the system type, and set the parameter RBS_PIPE_FLUID_TYPE_PARAM directly
        - FluidTemperature, FluidTemperature is now a property of the system type. Get the system type, and set the parameter RBS_PIPE_FLUID_TEMPERATURE_PARAM directly
        - FluidDensity, FluidDensity is now a property of the system type. Get the system type, and set the parameter RBS_PIPE_FLUID_DENSITY_PARAM directly
        - FluidViscosity, FluidViscosity is now a property of the system type. Get the system type, and set the parameter RBS_PIPE_FLUID_VISCOSITY_PARAM directly
    - Space.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
    - Application.LibraryPaths - replaced by GetLibraryPaths() and SetLibraryPaths()
    - ControlledApplication.LibraryPaths - replaced by GetLibraryPaths() and SetLibraryPaths()
    - Application.NewPointOnEdge(Autodesk.Revit.DB.Reference , double edgeParam) - replaced by other constructor that uses PointLocationOnCurve
    - Document members
        - Move() - all overloads - replaced by methods of ElementTransformUtils
        - Rotate() - all overloads - replaced by methods of ElementTransformUtils
        - Mirror() - all overloads - replaced by methods of ElementTransformUtils
        - Array() - all overloads - replaced by methods of LinearArray and RadialArray
        - ArrayWithoutAssociate() - all overloads - replaced by methods of LinearArray and RadialArray
        - SaveAs(System.String, bool) - replaced by SaveAs(String, SaveAsOptions)
        - FindReferencesByDirection() - replaced by FindReferencesWithContextByDirection() and ReferenceIntersector methods
        - ExternalFileReference & ImportFileData - all methods and enums related to material paths
        - AbbreviationItem - empty class removed completely
        - WorksharingCentralFilename - replaced by GetWorksharingCentralModelPath()
    - ModelPathUtils.GetRevitServerPrefix() - RSN:// is the prefix string for Revit Server path.

Major enhancements to the Revit API

Stairs and railings API

Stairs and stairs components

The new classes in the Autodesk.Revit.DB.Architecture namespace:

- Stairs
- StairsType

provide access to new stairs elements in the Revit database. Note that these API classes only provide access to the Stairs created "by component" introduced in this release. Stairs elements created by sketch cannot be accessed as a Stairs object in the API. It is possible to identify both type of stairs using the BuiltInCategory OST_Stairs and the static method Stairs.IsByComponent() can identify an element id represents a by component stairs.

The classes:

- StairsLanding
- StairsLandingType
- StairsRun
- StairsRunType

provide access to the subcomponents and subcomponent types of the new Stairs elements.

Railings and railing components

The new classes in the Autodesk.Revit.DB.Architecture namespace:

- Railing
- RailingType

provide access to the top level railing elements in the Revit database. Railings can be associated to a host, such as a stairs or a floor, or placed in space. Railings can be continuous or non-continuous. If non-continuous, only a limited level of access is provided.

Stairs annotations

The new class

- CutMarkType

provides access to the properties and capabilities of a CutMark annotation which can be associated to stairs.

The new classes

- StairsPath
- StairsPathType

provide access to the stairs path annotation.

View API

View Creation

- Plan Views can now be created with the new ViewPlan.Create(document, viewFamilyTypeId, levelId) method. For example, a ceiling plan can be created as follows:

```
Element level = doc.GetElement(uidoc.Selection.PickObject(ObjectType.Element)) as Level;

IEnumerable<ViewFamilyType> viewFamilyTypes = from elem in new
FilteredElementCollector(doc)
         .OfClass(typeof(ViewFamilyType))
          let type = elem as ViewFamilyType
          where type.ViewFamily == ViewFamily.CeilingPlan
          select type;

ViewPlan ceilingPlan = ViewPlan.Create(doc, viewFamilyTypes.First().Id, level.Id);
ceilingPlan.Name = "New Ceiling Plan for " + level.Name;
ceilingPlan.DetailLevel = ViewDetailLevel.Fine;

//3D views can be created with View3D.CreateIsometric and View3D.CreatePerspective.
//The new ViewOrientation3D object is used to get or set the orientation of 3D views.
View3D view = View3D.CreateIsometric(doc, viewFamily3d);
XYZ eyePosition = new XYZ(10, 10, 10);
XYZ upDirection = new XYZ(-1, 0, 1);
XYZ forwardDirection = new XYZ(1, 0, 1);
view.SetOrientation(new ViewOrientation3D(eyePosition, upDirection, forwardDirection));
```

- The ViewSection class now has methods that support the creation of Callout, Reference Callout, and Reference Section views.
- Rendering views containing images imported from disk can be created using the new ImageView class and the ImageView.Create method.

DisplayStyle

The enum DisplayStyle (with values including Wireframe, HLR, and Shading) has been added so the API can be used to get/set the new DisplayStyle property.

ViewDetailLevel

The enum ViewDetailLevel (with values Coarse, Medium, and Fine) has been added with get/set access to the new DetailLevel property.

ViewRange

The view range for plan views can now be retrieved via ViewPlan.GetViewRange().

3D View Locking

- 3D Views can now be locked and unlocked via the API with the methods SaveOrientationAndLock, RestoreOrientationAndLock, and Unlock.
- The property IsLocked indicates if a view is currently locked.
- NewTag can now be used in locked 3D Views.

View.Duplicate

Views can now be duplicated via the API with the View.Duplicate(ViewDuplicateOption) method.

UIView class and UIDocument.GetOpenUIViews()

The UIView class has been added to represent a view window in the Revit user interface. GetOpenUIViews provides a list of all open views. UIView.GetWindowRectangle() returns a rectangle that describes the size and placement of the UIView window.

Pan and Zoom

The method

- UIView.ZoomAndCenterRectangle()

offers the ability to zoom and pan the active view to center on the input region of the model.

The method

- UIView.GetZoomCorners()

returns the two points that define the corners of the view's rectangle in model coordinates.

PlanViewDirection

The enum PlanViewDirection has been added so API can be used to get/set the view direction to either Up or Down for StructuralPlan views.

ViewFamilyType class and ViewFamily enum

The ViewFamilyType class has been created to correspond to the Types seen in the UI's Properties dialog for each view. The ViewFamily enum (FloorPlan, Section, Legend, etc) specifies the family of the ViewFamilyType.

Temporary view modes

View.EnableRevealHiddenMode, View.EnableTemporaryAnalyticalDisplayMode, View.DisableTemporaryViewMode, and the TemporaryViewMode Enumeration have been added to allow control of temporary view modes.

Schedules

Schedule API

Several new classses have been added to allow schedule views to be created, modified, and added to drawing sheets. Major new classes include:

- The ViewSchedule class that represents the schedule view. Its create methods are used to create new schedules.
- The ScheduleField class for the inidivual fields in a schedule
- The ScheduleSheetInstance class represents schedules placed on sheets. The create method creates an instance of a schedule on a sheet.
- The ScheduleDefinition class defines the contents of a schedule view, including:

1. Basic properties that determine the kind of schedule, such as the schedule's category.
2. A set of fields that become the columns of the schedule.
3. Filters that restrict the set of elements visible in the schedule.
4. Sorting and grouping criteria.

Running API commands in Schedule Views

API commands may now be active when a Schedule is active in the current document.

Schedule export

The ViewSchedule class has been added to the API and the method ViewSchedule.Export method exports an existing schedule to a text file.

Discipline controls

The properties:

- Application.IsArchitectureEnabled
- Application.IsStructureEnabled
- Application.IsStructuralAnalysisEnabled
- application.IsMassingEnabled
- Application.IsEnergyAnalysisEnabled
- Application.IsSystemsEnabled
- Application.IsMechanicalEnabled
- Application.IsMechanicalAnalysisEnabled
- Application.IsElectricalEnabled
- Application.IsElectricalAnalysisEnabled
- Application.IsPipingEnabled
- Application.IsPipingAnalysisEnabled

provide read and modify access to the available disciplines. Your application can read the properties to determine when to enable or disable aspects of the UI.
Enabling and disabling disciplines is available only in Autodesk Revit (and not in any other Revit product). When a discipline's status is toggled, Revit's UI will be adjusted, and certain operations and features will be enabled or disabled as appropriate.

Rendering Options

The settings shown in the Rendering Options Dialog are exposed, allowing an API user can get and set Rendering Options for a 3d view.

The new methods:

- View3D.GetRenderingSettings
- View3D.SetRenderingSettings

gets or changes the rendering settings for the 3d view.

The class

- RenderingSettings

offers properties and methods which correspond to the options in the UI dialog:

- RenderingSettings.UsesRegionRendering - Boolean value that indicates whether to use region rendering.
- RenderingSettings.BackgroundStyle - The enum value that controls the background style for rendering.
- RenderingSettings.ResolutionTarget - The resolution target.
- RenderingSettings.PrinterResolution - The resolution level when using printer.
- RenderingSettings.LightingSource - The lighting scheme type.
- RenderingSettings.ResolutionValue - The rendering resolution in dots per inch (DPI).
- RenderingSettings.GetBackgroundSettings() - Returns an object that represents the rendering background settings.
- RenderingSettings.SetBackgroundSettings() - Changes the rendering background settings details for the current background style.
- RenderingSettings.GetRenderingImageExposureSettings() - Returns an object that represents the rendering image exposure settings.
- RenderingSettings.SetRenderingImageExposureSettings() - Changes the rendering image exposure settings.
- RenderingSettings.GetRenderingQualitySettings() - Returns an object that represents the rendering quality settings.
- RenderingSettings.SetRenderingQualitySettings() - Changes rendering quality settings.
- RenderingSettings.GetRenderingRegionOutline() - Returns the outline of the rendering region.

The rendering properties are exposed with the following supplementary classes and enums:

- enum BackgroundImageFit - Represents the type of background image fit.
- enum ResolutionTarget - Represents the resolution target (Screen or Printer) used for rendering.
- enum PrinterResolution - Controls the resolution level for a printer target.
- enum LightingSource - Represents the lighting scheme type.
- enum RenderingQuality - Represents the rendering quality.
- enum BackgroundStyle- Represents the background style.
- class SkyBackgroundSettings - Represents the rendering sky background settings.
- class ColorBackgroundSettings - Represents the rendering color background settings.
- class ImageBackgroundSettings - Represents the rendering image background settings.
- class RenderingImageExposureSettings - Represents the exposure settings of rendering.

Example: checking rendering options for a 3D view

```
RenderingSettings renderingSettings = view3D.GetRenderingSettings();
Transaction newTran = new Transaction(doc, "Change Rendering Settings");
newTran.Start();
renderingSettings.BackgroundStyle = BackgroundStyle.Image;
view3D.SetRenderingSettings(renderingSettings);
newTran.Commit();
```

Construction Modeling

Linked element parents for parts

Parts can be created from linked elements. There are new overloads:

- PartUtils.CreateParts()
- PartUtils.IsValidForCreateParts()
- PartUtils.HasAssociatedParts()
- PartUtils.GetAssociatedParts()
- PartUtils.GetAssociatedPartMaker()

which accept LinkElementIds (representing the id of the element in the link). The existing Part utilities accepting ElementId have not been changed.

Merged parts

The methods:

- PartUtils.ArePartsValidForMerge()
- PartUtils.CreateMergedPart()
- PartUtils.FindMergeableClusters()
- PartUtils.GetChainLengthToOriginal()
- PartUtils.GetMergedParts()
- PartUtils.IsMergedPart()

provide support for merging of parts.

Excluded parts

The new property

- Part.Excluded

allows identification and assignment of parts to be excluded.

Part category

The new members:

- Part.OriginalCategoryId
- Part.GetSourceElementOriginalCategoryIds()

provide the ability to get and set the original category id for a given part.

Part division

Part divisions can now be assigned custom geometry divisions. The members:

- PartMakerMethodToDivideVolumes.ProfileMatch
- PartMakerMethodToDivideVolumes.DivisionGap
- PartMakerMethodToDivideVolumes.ProfileOffset
- PartMakerMethodToDivideVolumes.SplitterProfileType
- PartMakerMethodToDivideVolumes.ProfileFlipAcross
- PartMakerMethodToDivideVolumes.ProfileFlipAlong
- PartMakerMethodToDivideVolumes.GetSketchCurves()
- PartMakerMethodToDivideVolumes.GetPlaneOfSketch()
- PartMakerMethodToDivideVolumes.GetSplitRefsOffsets()
- PartMakerMethodToDivideVolumes.CanBeDivisionProfile()

offer the ability to examine and control properties of the divisions.

Assembly views

The method

- AssemblyViewUtils.AcquireAssemblyViews()

transfers the assembly views owned by a source assembly instance to a target sibling assembly instance of the same assembly type.

Assembly instance transform and location

The methods

- AssemblyInstance.GetTransform()
- AssemblyInstance.SetTransform()

provide read and write access to the transformation origin of the assembly. Changing this value causes the origin point to change relative to the assembly members (and does not move the assembly members). All assemblies of the same type should have changed their transforms by the same amount. For example, if you changed the transform of the assembly by a rotation of 90 degrees, then other assemblies should also have rotated their transform by 90 degrees.

The inherited property

- AssemblyInstance.Location

also has been modified to return a the location for an AssemblyInstance.

Structure improvements

New Rebar members

- RebarShapeMatchesCurvesAndHooks() checks if rebarShape matches curves and hooks

- CreateFromCurvesAndShape() creates a new instance of a Rebar element with default shape parameters from the RebarShape
- GetBarPositionTransform()
- GetBarExistsAtPosition()
- GetHookTypeId() and SetHookTypeId()
- ScheduleMark property
- TotalLength property
- Volume property

ReinforcementSettings

The new class **ReinforcementSettings** controls project settings that are accessible from the Reinforcement Settings dialog in the Revit UI. For 2013 the only supported setting is the **HostStructuralRebar** property, which is new for 2013 and affects the **AreaReinforcement** and **PathReinforcement** elements.

AreaReinforcement/PathReinforcement

AreaReinforcement and PathReinforcement now have types.
The types - AreaReinforcementType and PathReinforcementType - contain only identity data.

There is no change to the interface for creating Area and Path Reinforcement elements. A default, or most-recently-used type, is automatically assigned on element creation.

New Area- and PathReinforcementTypes can be created using the following methods:

- (static) AreaReinforcementType AreaReinforcementType.Create(Document ADoc, AString Name)
- (static) PathReinforcementType PathReinforcementType.Create(Document ADoc, AString Name)

Analytical model

The new element type

- AnalyticalLink

can be created between two other Analytical Elements, e.g. an analytical beam and analytical column. There are two kinds of Links: user defined and automatically generated. Links have properties like "fixity state" that can be accessed via the API.

MEP improvements

Routing Preferences

Revit MEP now supports a system for selecting preferred pipe and duct sizes, materials, and fitting types for use in an MEP project. The routing preferences API can be used to set routing preference policies for end users as well as to query which fittings and segments will be used given size criteria.

- property MEPCurveType.RoutingPreferenceManager - accesses the main routing preferences object for a give MEPCurve Type. Currently, PipeType and DuctType are the only MEPCurve types that support Routing Preferences.

- class RoutingPreferenceManager - manages all routing preference rules for segments and fittings and allows the user to query which fitting or segment will be chosen by Revit, given a size condition.
- classes RoutingCriterionBase and PrimarySizeCriterion - These classes manage criteria for selecting fittings and segments based on minimum and maximum size constraints.
- class RoutingPreferenceRule - Manages one segment or fitting preference
- classes RoutingCondition and RoutingConditions - Inputs to RoutingPreferenceManager.GetMEPPartId() to select fittings and segments based on their selection criteria.
- class Segment - A class representing a length of MEPCurve that contains a material and set available sizes.
- class PipeSegment - a subclass of Segment representing a length of pipe
    - PipeSegment.Create(ElementId materialId, ElementId scheduleId) - Creates a new PipeSegment of given material, schedule, and size list.
- enum RoutingPreferenceRuleGroupType - Each routing preference rule is grouped according to what type of routing item it manages
    - Elbows
    - Junctions
    - Crosses
    - Transitions
    - Unions
    - MechanicalJoints
    - Segments
    - TransitionsRectangularToRound
    - TransitionsRectangularToOval
    - TransitionsOvalToRound

MEP Sections

The new class

- MEPSection

is the base class for duct and pipe section. It is a series of connected elements (segments - ducts or pipes, fittings, terminals and accessories). All section members should have same flow analysis properties: Flow, Size, Velocity, Friction and Roughness. One section member element which contains more than one connector can belongs to multiple section.
For example: One Tee fitting that has 3 connectors usually belongs to 3 sections. One segment that connects to a tap will be divided into 2 sections

FluidType and FluidTemperature

The new class

- FluidTemperature

represents the viscosity and density properties as defined at a certain temperature.

The class

- FluidType

has been extended to provide read and write access to a collection of FluidTemperature objects which represent the fluid's properties at various temperatures.

Panel schedule - spare values

The new functions

- PanelScheduleView.GetSpareLoadValue()
- PanelScheduleView.SetSpareLoadValue()
- PanelScheduleView.GetSpareCurrentValue()
- PanelScheduleViwe.SetSpareCurrentValue()

provide access to the parameter values of spare circuits.

LabelUtils

New overloads of

- LabelUtils.GetLabelFor()

have been added to supply the user-visible string matching members of the enums DuctLossMethodType, PipeLossMethodType, and PipeFlowState.

Physical Properties

Thermal properties

Thermal properties of various elements have been enhanced in Revit 2013.

- class ThermalProperties
    - Absorptance
    - HeatTransferCoefficient
    - Roughness
    - ThermalMass
    - ThermalResistance

This class is a property on the following types:

- ThermalProperties WallType.ThermalProperties
- ThermalProperties FloorType.ThermalProperties
- ThermalProperties CeilingType.ThermalProperties
- ThermalProperties RoofType.ThermalProperties
- ThermalProperties BuildingPadType.ThermalProperties
- class FamilyThermalProperties
    - AnalyticConstructionTypeId - This value corresponds to the 'id' property of a constructionType node in Constructions.xml, a supplied data library of thermal properties of common construction types.
    - HeatTransferCoefficient
    - SolarHeatGainCoefficient
    - ThermalResistance
    - VisualLightTransmittance

    - Find() - returns a FamilyThermalProperties object from an "id" string in Constructions.xml
- FamilySymbol.GetThermalProperties() – Gets the thermal properties of a family symbol of a door, window, or curtain panel.
- FamilySymbol.SetThermalProperties()-- Sets the thermal properties of a family symbol of a door, window, or curtain panel.

This class is a property on Door, Window, and Curtain Panel family symbols

- FamilyThermalProperties FamilySymbol.ThermalProperties

Structural properties

- Beam, Column, and Foundation family instances now support an ElementId of an additional structural Material element
    ElementId FamilyInstance.StructuralMaterialId (Foundation, Beam, and Column instances)

All compound structurals can now elect one layer in the structural to be used as the "structural" layer.

- int CompoundStructure.StructuralMaterialIndex
- PropertySetElement now supports a StructuralAsset property via the GetStructuralAsset and SetStructuralAsset methods.
  StructuralAsset contains named properties for various structural values, such as PoissonModulus and YoungModulus, that offer enhanced ease of use over the previous parameter-api based access to these same values.

Material assets

- StructuralAsset – contains structural properties that can be set on materials via PropertySetElement and SetMaterialAspectByPropertySet()
- ThermalAsset – contains thermal properties that can be set on materials via PropertySetElement and SetMaterialAspectByPropertySet()
- PropertySetElement.Create() - now takes a StructuralAsset or ThermalAsset as a parameter instead of a MaterialAspect.

GBXML Export

- Thermal properties of building construction element types can now be set to use calculated values or pre-defined values from
  Constructions.xml with:
    - MEPBuildingConstruction.GetBuildingConstructionOverride
    - MEPBuildingConstruction.SetBuildingConstructionOverride
- EnergyDataSettings.IncludeThermalProperties
    - Indicates if thermal information from model assemblies and components is included in GreenBuildingXML export of the detailed model.

User interface API improvements

Contextual help support

Items placed on the Ribbon can now be assigned an action for contextual help using

- RibbonItem.SetContextualHelp()

The options supported include linking to an external URL, launching a locally installed help file, or linking to a topic on the Autodesk help wiki.

You can also invoke the same set of contextual help options at any arbitrary time within your add-in by creating a ContextualHelp object with the appropriate target location, and invoking

- ContextualHelp.Launch()

Support for Keyboard Shortcuts and Quick Access Toolbar

API commands may now be assigned keyboard shortcuts, and those assignments will be preserved even if add-ins are added, removed or changed in subsequent sessions.

API commands may also be moved to the Quick Access Toolbar, and that assignment will also be preserved even if add-ins are added, removed or changed in subsequent sessions.

Replace implementation of commands

It provides ability to replace the existing Revit command (located in any tab, application menu and right-click menus) implementation with add-in routine.

- RevitCommandId

The new class RevitCommandId provides ability to look up and retrieve an object representing Revit Command id given an id string. The id string of the command is the string which appears in the journal when the command is clicked, e.g. "ID_APP_EXIT".

- AddInCommandBinding

The new class AddInCommandBinding provides ability to override the Revit command implementation with Executed and CanExecute events.

- UIApplication.CreateAddInCommandBinding()
- UIControlledApplication.CreateAddInCommandBinding()

These 2 new methods provides ability to create the command binding with given Revit command id.

Preview control

The new class

- PreviewControl

provides the ability to embed an interactive Revit view as a WPF control inside a dialog.

Options dialog customization

Subscribe to the new event

- UIApplication.DisplayingOptionsDialog

to be notified when Revit is preparing the Options dialog for display. During the event callback, your application can call

- DisplayingOptionsDialogEventArgs.AddTab()

to add a new tab to the dialog. The contents of the panel shown when this new tab is selected are determined by the members assigned to the input class:

- TabbedDialogExtension

Drag & Drop support

The new methods

- UIApplication.DoDragDrop(ICollection<String>)
- UIApplication.DoDragDrop(Object, IDropHandler)

initiate a drag and drop operation on the Revit user interface.
The first method accepts a list of file names, which can be Revit files, import CAD and images, or some combination of the two. The second method accepts arbitrary data, and provides an IDropHandler callback to handle the drop of the data when the user completes it.

Translation API enhancements

Several updates have been made to the APIs related to data exchange and translation of models.

DGN import

The new method

- Document.Import(string, DGNImportOptions, View)

supports import of DGN files into Revit views.

DXF import

The method

- Document.Import(string, DWGImportOptions)

now also supports DXF files for import.

DWG and DXF export settings changes

The options supported for DWG and DXF export have been expanded. A new base class

- BaseExportOptions

supports all properties shared by DWG/DXF & DGN export tasks. New properties include:

- BaseExportOptions.HideScopeBox
- BaseExportOptions.HideReferencePlane
- BaseExportOptions.HideUnreferenceViewTag

DGN export

The options supported for DGN export have changed to match the new implementation of DGN export based on DWG supporting DGN V8. The previously existing class

- DGNExportOptions

thus offers many new properties inherited from the same base classes as DWGExportOptions.

The property

- DGNExportOptions.TemplateFile

has been replaced with:

- DGNExportOptions.SeedName

FBX Export

Three new options have been added:

- FBXExportOptions.UseLevelsOfDetail - true to use levels of detail, false otherwise
- FBXExportOptions.LevelsOfDetailValue - the value of the levels of detail
- FBXExportOptions.WithoutBoundaryEdges - true to export without boundary edges, false otherwise

DividedPath

A DividedPath is an element supported in the massing environment that consists of a set of points distributed along a connected set of curves and edges. The points can be the result of a uniform distribution along the curves. The type of the distribution is determined by a selected 'layout'. The distance between the layout points depends on the curves, the layout, and layout specific settings. In addition, points can also be the result of intersecting the curves with other elements.

The class

- DividedPath

exposes the interface for this element type. It permits creation of a new DividedPath from input curves and edges, as well as optionally, intersecting elements. It allows control over the layout type and parameters, as well as other miscellaneous settings.

Analysis Visualization Framework

The following additions have been made to support deformed shapes in analysis visualization display:

- AnalysisDisplayStyleDeformedShapeTextLabelType class
- AnalysisDisplayDeformedShapeSettings class

A new override to CreateAnalysisDisplayStyle accepts an input argument of AnalysisDisplayDeformedShapeSettings

Revit Link creation

The API now contains two creation methods for Revit links.

- RevitLinkType.Create(Document, ModelPath, RevitLinkOptions) will create a new Revit link type and load the associated linked file into the document. This returns a RevitLinkLoadResult, which stores the ElementId of the newly-created RevitLinkType and contains any errors which occurred when trying to load the linked file (The RevitLinkLoadResultType enumeration contains the full list.)
- RevitLinkInstance.Create(Document, ElementId) will create a new instance of an already-loaded RevitLinkType.

FilledRegion

The class

- FilledRegion

has been extended to offer the ability to create Filled regions, to get the boundaries of the region, and to apply a linestyle to all boundary segments.

The class

- FilledRegionType

has been added providing access to the visible properties of a filled region type.

Light API

The classes

- LightType
- LightFamily

have been added to offer the ability to get and set photometric data and all other lighting parameters from both family instances in project documents and all family types in family documents. The LightType and LightFamily classes expose the same data except for light shape and distribution types which are only available from the LightFamily class.

Some examples of lighting parameters exposed are;

- initial color
- initial intensity
- loss factor
- color filter
- dimming color

Light Group API

The classes

- LightGroupManager

- LightGroup

have been added to offer the ability to manage light groups to allow control over which lights are used when rendering the scene.

The LightGroupManager class gives the user the ability to

- create a new light group
- delete an existing light group
- turn on or off all the lights in a light group
- turn on or off individual lights
- set the dimmer value for individual lights

The LightGroup class gives the user the ability to

- get the name of a light group
- rename light group
- add a light to a light group
- remove a light from a light group

Light groups are used in rendering options to turn lights on or off when rendering.

ReferenceIntersector

The new class ReferenceIntersector allows ray-cast selection of elements, given a point and direction, similar to FindReferencesWithContextByDirection(), but with support for filtering the output based on element or reference type.
The enum FindReferenceTarget is used with ReferenceIntersector to filter selection of elements, meshes, edges, curves, and faces.

Key members of ReferenceIntersector

- ReferenceIntersector(ElementId targetElementId, FindReferenceTarget targetType, View3d view3d) - constructor specifying a single element to search for in the intersection test.
- ReferenceIntersector(ElementIdSet targetElementIds, FindReferenceTarget targetType, View3d view3d) - constructor specifying a set of ElementIds to search for in the intersection test.
- ReferenceIntersector(ElementFilter filter, FindReferenceTarget targetType, View3d view3d) - constructor specifying an ElementFilter for the intersection test.
- Find(XYZ origin, XYZ direction) - Finds all references intersecting the origin-direction ray given the selection criteria set up in the ReferenceIntersector constructor
- FindNearest (XYZ origin, XYZ direction) - Finds the reference closest to the origin in the origin-direction ray given the selection criteria set up in the ReferenceIntersector constructor

Small enhancements & API interface changes

Elements & filtering

Element.PhaseCreated and Element.PhaseDemolished

These readonly properties have been obsoleted and replaced with read/write properties:

- Element.CreatedPhaseId
- Element.DemolishedPhaseId

The new method Element.AllowsPhases() indicates whether these phase id properties can be modified for any given element.

Phase filter

The new class

- PhaseFilter

and related types support read/write and create access to phase filters.

SelectionFilterElement

The new class

- SelectionFilterElement

provides access to a filter type which contains a selected set of element ids, rather than criteria based on parameters.

Ceiling, Floor and CeilingAndFloor

The newly exposed Ceiling class represents a ceiling in Revit. It inherits from CeilingAndFloor. The Floor class has been modified to also inherit from CeilingAndFloor.

Geometry & sketching

Split volumes

The new method:

- SolidUtils.SplitVolumes()

takes a solid which includes multiple disjoint enclosed volumes, and returns newly allocated Solids representing each volume. If no splitting was necessary, a copy of the input Solid is returned.

Solid tessellation

The new method:

- SolidUtils.TessellateSolidOrShell()

generates a single triangulated structure for the given input Solid (which could one or more fully closed volumes, or a shell of faces which do not form a closed volume). The function returns:

- TriangulatedSolidOrShell

which allows access to the stored triangulated components and triangles.

Use the method:

- FacetingUtils.ConvertTrianglesToQuads()

with a TriangulationInterface object constructed from the TessellateSolidOrShell results to convert pairs of adjacent, coplanar triangles into quadrilaterals.

Face.Triangulate()

A new overload to Face.Triangulate() accepts a level of detail as an argument. Levels of detail close to 0 (the minimum) will result in coarser triangulations than levels of detail close to 1 (the maximum).

Ellipse - axes

The new properties:

- Ellipse.XDirection
- Ellipse.YDirection

provide the unit vectors of the ellipse for the X and Y directions.

GeometryElement.GetBoundingBox()

This new method provides read access to the bounding box of a given geometry element.

ItemFactoryBase.NewAlignment()

This method can now align a line and a reference plane as well as all other geometry types from previous versions.

CylindricalHelix curve type

A new subclass of Curve has been introduced: CylindricalHelix. The class offers methods useful for reading and creating such curves.

In this release, CylindricalHelix curves are used only in specific applications in stairs and railings, and should not be used or encountered when accessing curves of other Revit elements and geometry.

CurveLoop as IEnumerable<Curve>

The members of a CurveLoop can now be traversed, as CurveLoop implements IEnumerable<Curve>.

CurveElement - center point reference

The new property:

- CurveElement.CenterPointReference
  returns a reference to the center of a curve if that curve is one that has a center (e.g. arcs, circles, elliptical arcs or ellipses). This reference can be used for dimensioning to the center point.

ModelCurve.ChangeToReferenceLine

ModelCurve.ChangeToReferenceLine is now supported in all family types, not just conceptual modeling.

NewSketchPlane(Document, Reference)

This new overload for NewSketchPlane in the ItemFactoryBase class supports creation of a sketch plane directly from a Reference representing a planar face.

Detailing & Annotations

MultiSegmentGrid class

This class has been added to the API to support multi-segment grids.

Grid geometry

Element.Geometry now supports Grids and MultiSegmentGrids, including the ability to see gaps and extents applied to the grid lines to form different segments.

Diameter dimensions

Revit now supports Diameter dimensions. The DimensionShape enumeration contains a new Diameter member. And there is a new creation method for diameter dimensions:

- FamilyItemFactory.NewDiameterDimension()

DimensionSegment overrides

There are several new properties on DimensionSegment to override text properties.

- DimensionSegment.Prefix
- DimensionSegment.Suffix
- DimensionSegment.Above
- DimensionSegment.Below
- DimensionSegment.ValueOverride

These properties are also available on Dimension, for use with dimensions with just one segment.

Detail element draw order

The class

- DetailElementOrderUtils

has been added providing the ability to affect the draw order of detail elements (including FilledRegions)

Families & content

SpatialElementCalculationPoint

This class provides access to the "Room or Space Calculation Point" seen in families. The Spatial Element Calculation Point is graphically showed as a location point marker with a "snake" line from the origin of the family. If the Spatial Element Calculation Point is turned on, this point will be used as a search point for room and space relations for all instances of this family. The API offer creation, read and modification options for this object type.

RPC content assets

The AssetType enum now contains a value "Content", which can be used to enumerate the loaded RPC content assets in the Revit session.

NewFamilyInstance validation

Some validation has been added to overloads of NewFamilyInstance(). This validation is intended to prevent use of the incorrect overload for a given family symbol input. For specific details on the exceptional conditions which
will be validated, consult the documentation.

Family.PlacementType

This new property provides information about the placement type for instances of the given family. PlacementType roughly maps to the overloads of NewFamilyInstance.

DataStorage element

The new DataStorage class represents an element which API applications can create logically organize different sets of ExtensibleStorage entities into separate elements. This allows an application to update one set of data in a local workshared project without locking other elements.

ApplicationInitialized event

The ControlledApplication object now supports an ApplicationInitialized event, which is called once the Revit application is fully initialized, after all external applications have been started and when the application is ready to work with documents.

ProgressChanged application event and ProgressChangedEventArgs

The application object now supports a ProgressChanged event and ProgressChangedEventsArgs object that returns progress bar data from time-consuming transactions.

- ProgressChangedEventArgs
    - Caption – The name of the current transaction or subtransaction in progress
    - Stage – The type of progress event (Started, RangeChanged, PositionChanged, CaptionChanged, UserCancelled, Finished)
    - Position – The numerical index between zero and UpperRange showing the number of completed steps of the current transaction or subtransaction

- UpperRange – The total number of steps in the current transaction or subtransaction
- Cancel - Cancels the current transaction in progress, if possible.
- Cancellable - indicates if the current transaction can be cancelled.

SiteLocation - PlaceName

The new property:

- SiteLocation.PlaceName

provides the place name of the site location.

Application.DefaultProjectTemplate

Provides the full path to the default template file for new project documents

UnitType, DisplayUnit and FamilyParameter updates

Many new units have been added to UnitType, DisplayUnitType and ParameterType.

One UnitType was renamed: UT_TemperalExp was renamed to UT_ThermalExpansion.
The corresponding ParameterType value (TemperalExp) was also renamed to ThermalExpansion.

BasePoint and BASEPOINT_ANGLETON_PARAM "True North" Parameter

The parameter to return the "True North" angle parameter "BASEPOINT_ANGLETON_PARAM" on the BasePoint object is now a double instead of a string.

Journaling of Add-in user interface

Revit now activates a tool to collect and journal certain actions which take place in the user interface provided by an add-in. The tool supports actions in both WPF and Windows Forms dialogs.
Some actions and some components may not be fully recorded.

This capability is useful to Autodesk in the event that a journal can be supplied from a Revit user indicating the presence of a software problem. Replay of add-in UI via journals is not supported.

Major changes and renovations to the Revit API 2014

Document APIs

Document.Save()

Several modifications have been made to the Document.Save() methods.

- The methods now return void instead of boolean.
- Failures are signaled with specified documented exceptions.
- The new property SaveOptions.Compact allows the caller to specify if the OS should eliminate all dead data from the file on disk.

Document.SaveAs()

Several modifications have been made to the Document.SaveAs() functions.

- The methods now return void instead of boolean.
- Failures are signaled with specified documented exceptions.
- The new property SaveAsOptions.MaximumBackups identifies the maximum number of backups to keep on disk.
- The new property SaveAsOptions.Compact allows the caller to specify if the OS should eliminate all dead data from the file on disk.
- The new property SaveAsOptions.WorksharingOptions offers options specific to workshared files:

- WorksharingSaveAsOptions.SaveAsCentral
- WorksharingSaveAsOptions.OpenWorksetsDefault
- WorksharingSaveAsOptions.ClearTransmitted
- The property SaveAsOptions.Rename has been obsoleted. The property previously had no effect, use of SaveAs will always modify the document in memory.

In addition, a new version of SaveAs() taking a ModelPath and SaveAsOptions has been introduced. This offers capabilities similar in behavior to SaveAs(string, SaveAsOptions).

OpenOptions

New options are available in this class:

- OpenOptions.Audit

This boolean option specifies whether to expand all elements, to check for corruption. Defaults to false.

- OpenOptions.AllowOpeningLocalByWrongUser

  This boolean option specifies whether a local file is allowed to be opened ReadOnly by a user other than its owner.

- OpenOptions.GetOpenWorksetsConfiguration()
- OpenOptions.SetOpenWorksetsConfiguration()

  These methods access the WorksetConfiguration. This class specifies which user-created worksets should be opened/closed when the document is opened. Once an instance of this class is created, it can be further modified by calling available methods in any order. It is a specification of a setting for model open; the methods of this class just adjust the specification, and do not themselves open or close worksets.

  Only user-created worksets can be specified to be opened or closed. All system worksets are automatically open. An open workset allows its elements can be expanded and displayed. For a closed workset, Revit tries to not expand its elements, and to that end, does not display them. This is intended to help with performance by reducing Revit's memory footprint.

  An application can determine how to populate this class by looking at workset information from a closed document. This is done by calling

- WorksharingUtil.GetUserWorksetInfo()

  This method returns a collection of WorksetPreview classes containing the workset id, unique id, owner, name and whether or not the workset is default.

Application.OpenDocumentFile(ModelPath, OpenOptions)

This method has been reimplemented. While the signature has not changed, there are new options available to be set in OpenOptions.

Failures are now signaled with specific documented exceptions.

Application.OpenDocumentFile(String)

This method has been reimplemented. While the signature has not changed, failures are now signaled with specific documented exceptions.

UIApplication.OpenAndActivateDocument(ModelPath, OpenOptionsForUI)

The signature and the implementation of this method has changed.

- The class OpenOptionsForUI has been removed.
- The new signature is (ModelPath, OpenOptions, Boolean detachAndPrompt).

If you wish to let the user answer interactively the prompt of whether to preserve or discard worksets when opening the documented detached from central, set the following values:

- OpenOptions.DetachFromCentralOption = DoNotDetach
- Boolean argment "detachAndPrompt" = true.

Failures are now signaled with specific documented exceptions.

UIApplication.OpenAndActivateDocument(String)

This method has been reimplemented. Failures are now signaled with specific documented exceptions.

FilteredElementCollector

Iteration and element deletion

When an element is deleted active iterations over the document (via FilteredElementCollector) are now stopped. A new InvalidOperationException is thrown. The iterator cannot proceed due to changes made to the Element table in Revit's database (typically, this can be the result of an Element deletion). This affects the use of FilteredElementIterator, FilteredElementIdIterator, and foreach loops over a FilteredElementCollector.

The exception can be triggered by direct calls to Document.Delete(), but also by other Revit APIs which change the document resulting in an element being deleted. In general it is best not to make any changes to the document while an iterator is running over the document.

The simplest workaround to fix existing code which encounters this error is to use one of:

- FilteredElementCollector.ToElements()
- FilteredElementCollector.ToElementIds()
- FilteredElementCollector.ToList<Type>() (LINQ method)

to get a standalone collection, then iterate that collection. Even if elements are deleted from the document, the iteration of the already fetched collection can proceed.

Geometry APIs

Curve creation

New curve creation methods have been added as statics on the associated curve type:

- Curve.CreateTransformed()
- Line.CreateBound()
- Line.CreateUnbound()
- Arc.Create()
- Ellipse.Create()
- NurbSpline.Create()
- HermiteSpline.Create()

The older curve creation properties and methods are now obsolete:

- Curve.Transformed
- Line.Bound
- Line.Unbound
- Autodesk.Revit.Creation.Application.NewLine()
- Autodesk.Revit.Creation.Application.NewLineBound()
- Autodesk.Revit.Creation.Application.NewLineUnbound()
- Autodesk.Revit.Creation.Application.NewArc()
- Autodesk.Revit.Creation.Application.NewEllipse()
- Autodesk.Revit.Creation.Application.NewNurbSpline()
- Autodesk.Revit.Creation.Application.NewHermiteSpline()

Both the old and new curve creation routines are updated to consistently prevent creation of curves smaller than Revit's tolerance. This value is now exposed via:

- Application.ShortCurveTolerance

Some other adjustments have been made to the validation on specific curve creation routines, consult the documentation for details.

Curve utilities

The new methods:

- GetEndPoint()
- GetEndParameter()
- GetEndPointReference()

replace the indexed property utilities on Curve:

- EndPoint
- EndParameter
- EndPointReference

The setter for EndParameter is deprecated and code should be adjusted to call MakeBound(double, double) instead.

Edge utilities

The new methods:

- Edge.GetFace()
- Edge.GetEndPointReference()

replace:

- Edge.Face (property)
- Edge.EndPointReference (property)

Transform initialization

The new methods:

- Transform.CreateTranslation()
- Transform.CreateRotation()
- Transform.CreateRotationAtPoint()
- Transform.CreateReflection()

replace:

- Transform.Translation (property)
- Transform.Rotation (property)
- Transform.Reflection (property)

SketchPlane creation

The SketchPlane creation methods of Autodesk.Revit.Creation.ItemFactoryBase have been replaced.

- SketchPlane.Create(Document, Plane) replaces ItemFactoryBase.NewSketchPlane(Plane)
- SketchPlane.Create(Document, Reference) replaces ItemFactoryBase.NewSketchPlane(Reference)
- Either method may be appropriate to replace calls to ItemFactoryBase.NewSketchPlane(PlanarFace), depending on whether the goal is a sketch plane tied to a Revit geometric reference, or a sketch plane fixed in space to a given plane.

A new SketchPlane creation has been added:

- SketchPlane.Create(ElementId)

This creates a SketchPlane from a grid, reference plane, or level.

SketchPlane also now has two more methods to get related properties of the SketchPlane element.

- SketchPlane.GetPlane() Returns the corresponding Plane.
- SketchPlane.GetPlaneReference() Returns a reference to this element as a plane.

BeamSystem creation

The BeamSystem creation methods of Autodesk.Revit.Creation.Document have been replaced:

- BeamSystem.Create(Document, IList<Curve>, SketchPlane, int) replaces Document.NewBeamSystem(CurveArray, SketchPlane)
- BeamSystem.Create(Document, IList<Curve>, SketchPlane, XYZ, bool) replaces Document.NewBeamSystem(CurveArray, SketchPlane, XYZ, bool)
- BeamSystem.Create(Document, IList<Curve>, Level, int, bool) replaces Document.NewBeamSystem(CurveArray, Level)

- BeamSystem.Create(Document, IList<Curve>, Level, XYZ, bool) replaces Document.NewBeamSystem(CurveArray, Level, XYZ, bool)

Truss creation

The Truss creation method of Autodesk.Revit.Creation.Document have been replaced.

- Truss.Create(Document, ElementId, ElementId, Curve) replaces Document.NewTruss(TrussType, SketchPlane, Curve)

Family Symbol API

Family symbols that are not used in the document are now inactive until they are used. A symbol's geometry will be empty and should not be accessed until it is active. To test if a symbol is active, use

- FamilySymbol.IsActive()

To activate an inactive family symbol, use

- FamilySymbol.Activate()

Units API

The API for Units in Revit has been expanded and changed. The methods

- Document.GetUnits()
- Document.SetUnits()

allow interaction with the units of a document. The Units class provides access to data such as

- DecimalSymbol
- DigitGroupingAmount
- DigitGroupingSymbol
- FormatOptions
  The FormatOptions class provides access to data including:

- Rounding
- Accuracy
- DisplayUnits
- SuppressLeadingZeros
- SuppressTrailingZeros
- SuppressSpaces

LabelUtils.GetLabelFor() has been enhanced so that it can now return the user-visible name for a UnitSymbolType.

Unit Formatting and Parsing

The methods:

- UnitFormatUtils.FormatValueToString()
- UnitFormatUtils.TryParse()

provide the ability to format a value into a string based on formatting options and to parse a formatted string (including units) into a value if possible.

Unit Conversion

The new class UnitUtils contains methods to convert between unit types:

- UnitUtils.Convert() - Converts a value from one unit type to another, such as square feet to square meters.
- UnitUtils.ConvertFromInternalUnits() - Converts a value from Revit's internal unit type.
- UnitUtils.ConvertToInternalUnits() - Converts a value to Revit's internal unit type.

View API changes

Viewport.Create behavioral change

The method

- Viewport.Create()

previously did not consistently align the center of the Viewport with the point supplied. This has been changed, and now the center will be aligned with the input point. This should allow easier alignment of multiple viewports on the same sheet.

View.ViewName obsolete

The property

- View.ViewName

is now obsolete. View.Name can be used.

View.SetVisibility()

The name of this method has now been correctly capitalized.

ViewSchedule changes

ViewSchedule is now a child of TableView. All previously existing ViewSchedule API is still valid, but TableView also brings in a set of APIs related to:

- Table sections (header and body)
- Table formatting
- The contents of individual table cells

There are methods on TableView (and its constituent TableData and TableSectionData class) that are useful for Electrical Panel Schedules and some varieties of specialized schedules but forbidden for use with standard Revit tabular schedules generated from Revit content (e.g. InsertRow(), RemoveRow()). Use of these functions on standard Revit schedules will result in an exception.

Some new members were introduced on schedule related classes:

- ScheduleField.SetStyle()
- ScheduleField.GetStyle()
- ScheduleField.IsOverridden
- ScheduleField.ResetOverrides()
- ScheduleField.GetFormatOptions()
- ScheduleField.SetFormatOptions()

relate to the style and formatting applied to schedule columns, and:

- ScheduleField.IsCalculatedField
- ScheduleField.IsCombinedParameterField
- ScheduleField.HasSchedulableField

relate to information about the type of a field.

The new members:

- ViewSchedule.GetFontId()
- ViewSchedule.AddFontId()

provide access to fonts stored in the table and applied to cells.

The new members:

- ViewSchedule.GroupHeaders()
- ViewSchedule.UngroupHeaders()
- ViewSchedule.CanGroupHeaders()
- ViewSchedule.CanUngroupHeaders()

affect header grouping in the schedule.

The new method:

- ViewSchedule.GetTableData()

returns the object which provides access to the sections of the table.

Materials API changes

Applying visual materials

The method

- Material.SetRenderAppearance()

has been deprecated. The Render appearance properties should be set via the related ApperanceAssetElement.

Use the new property:

- Material.AppearanceAssetId

to assign the element to the material.

AppearanceAssetElements can be found via element filtering - they expose the following members:

- AppearanceAssetElement.Create() - creates a new asset element for a given rendering Asset and name.
- AppearanceAssetElement.GetAppearanceAssetElementByName() - gets an asset element handle given the name.
- AppearanceAssetElement.SetRenderingAsset() - Sets the rendering Asset to the element

AssetProperty changes

AssetProperty no longer inherits from APIObject.

New subclasses of AssetProperty are exposed:

- AssetPropertyList - a property consisting of a list of AssetProperty members.
- AssetPropertyFloatArray - a property consisting of an array of float values.
- AssetPropertyUInt64 - a property of UInt64 value.
- AssetPropertyInt64 - a property of Int64 value.

Some of the return values of AssetProperty.GetTypeName() have been changed as shown in the following form:

Input argument	old return	new return
APT_Double	"Double1"	"Double"
APT_DoubleArray2d	"Double2"	"DoubleArray2d"
APT_DoubleArray3d	"Double3"	"DoubleArray3d"
APT_DoubleArray4d	"Double4"	"DoubleArray4d"
APT_Asset	"RenderingAsset"	"Asset"
APT_FloatArray	"Float3"	"FloatArray"

UI API changes

External commands now supported from Project Browser as active view

API commands and macros are now enabled when the Revit active view is the Project Browser.

- If there are actively selected elements in the Project Browser, these will be returned from UIDocument.Selection
- However, add-ins cannot prompt for interactive selection when the Project Browser is the active view.
- The enumerated type ViewType now has separate entries for ProjectBrowser and SystemBrowser to allow applications to deal with situations where the Project Browser is active. These view types used to be returned as ViewType.Internal, so code which keys of off ViewType.Internal may need to be updated to also deal with these new types.

Note that API commands are still disabled when the active view is the MEP system browser.

Beam and Brace Parameters changes

Revit 2014 includes several changes to control the position of structural framing members like beams and braces. These changes do not affect the API members but can be accessed via parameters.

Start/End Extension & Cutback

There are new extension and cutback parameters assigned to Beam and Brace elements.

- BuiltInParameter.START_EXTENSION
- BuiltInParameter.END_EXTENSION
- BuiltInParameter.START_JOIN_CUTBACK
- BuiltInParameter.END_JOIN_CUTBACK

Note: In some families “Start Extension” and “End Extension” family parameters may also exist but it is recommended to use instead new the new built-in parameters.

These parameters work as follows:

- If the Beam or Brace element end doesn’t belong to a join Revit uses the new parameters “Start Extension" or "End Extension”. Assigning positive values to these parameters lengthens the element).
- If the Beam or Brace element belongs to a join Revit uses the parameters “Start Join Cutback" or "End Join Cutback”. Assigning positive values to these parameters shortens the element).

In certain cases it may be difficult to detect if an element node belongs to join or not. Therefore, it may be advisable to set both groups of parameters via the API. With both groups of parameters will be set for element, Revit automatically detects which should be applied to the structural element.

Justifications

There are new justification parameters assigned to Beam and Brace elements. The new set of parameters provides more options to manipulate the physical element in relation to its analytical curve.

Parameter	Permitted values
BuiltInParameter.YZ_JUSTIFICATION	YZJustificationOption.Uniform YZJustificationOption.Independent
BuiltInParameter.Y_JUSTIFICATION	YJustification.Left YJustification.Center YJustification.Origin YJustification.Right
BuiltInParameter.Y_OFFSET_VALUE	double

BuiltInParameter.Z_JUSTIFICATION	ZJustification.Top ZJustification.Center ZJustification.Origin ZJustification.Bottom
BuiltInParameter.Z_OFFSET_VALUE	double
BuiltInParameter.START_Y_JUSTIFICATION	YJustification.Left YJustification.Center YJustification.Origin YJustification.Right
BuiltInParameter.START_Y_OFFSET_VALUE	double
BuiltInParameter.START_Z_JUSTIFICATION	ZJustification.Top ZJustification.Center ZJustification.Origin ZJustification.Bottom
BuiltInParameter.START_Z_OFFSET_VALUE	double
BuiltInParameter.END_Y_JUSTIFICATION	YJustification.Left YJustification.Center YJustification.Origin YJustification.Right
BuiltInParameter.END_Y_OFFSET_VALUE	double
BuiltInParameter.END_Z_JUSTIFICATION	ZJustification.Top ZJustification.Center ZJustification.Origin ZJustification.Bottom
BuiltInParameter.END_Z_OFFSET_VALUE	double

Previously only Beam elements had any justification parameters.

In Revit 2014, Beams as well as Braces share this set of built-in justification parameters.

The following table shows a mapping from old justification parameters to new ones:

Deprecated parameter	New parameter
BEAM_H_JUSTIFICATION	Y_JUSTIFICATION
BEAM_V_JUSTIFICATION	Z_JUSTIFICATION
BEAM_V_JUSTIFICATION_OTHER_VALUE (valid only for Other value set to BEAM_V_JUSTIFICATION)	Z_OFFSET_VALUE (valid for all Z_JUSTIFICATION values)

Old beam parameters can still be used. When set, they will be used to recalculate the new parameters. It is strongly recommended to use new justification parameters instead of old ones, because in some cases the old parameters may be not sufficient to correctly describe the element position (for example, when justification is set independently on either end of the member).

Divided surface changes

Divided Surface API

The class DividedSurfaceData and the associated method Element.GetDividedSurfaceData() have been replaced. The new methods to access DividedSurfaces applied to elements are:

- DividedSurface.GetReferencesWithDividedSurface()
- DividedSurface.GetDividedSurfaceForReference()

The method:

- Autodesk.Revit.Creation.Document.NewDividedSurface()

has been replaced by:

- DividedSurface.Create()

Point clouds

Revit 2014 includes a new point cloud engine. This new engine supports .rcp and .rcs file formats. The introduction of this new engine has led to several changes in the client API around Point Cloud elements.

PointCloudType.Create()

The method

- PointCloudType.Create()

no longer supports input of .pcg files to directly create a new PointCloudType. .pcg files can be indexed to create matching .rcs files, but this does not happen automatically when using this method.

This method does support creation of PointCloudTypes from .rcp or .rcs files, or from custom formats supplied by the Point Cloud Engine API.

PointCloudInstance.GetPoints()

The method

- PointCloudInstance.GetPoints(PointCloudFilter filter, int numPoints)

has been deprecated and replaced by

- PointCloudInstance.GetPoints(PointCloudFilter filter, double averageDistance, int numPoints)

The new averageDistance argument is the desired average distance between "adjacent" cloud points (in Revit units of length). The smaller the averageDistance the larger number of points will be returned up to the numPoints limit. Specifying this parameter makes the actual number of points returned for a given filter independent of the density of coverage produced by the scanner. This average distance parameter is only used when extracting points from .rcs or .rcp point clouds, and is not used when extracting from .pcg point clouds or custom point clouds from the Point Cloud Engine API.

Point cloud overrides

The classes

- PointCloudOverrides
- PointCloudOverrideSettings
- PointCloudColorSettings

allow read and write access to point cloud override settings assigned to a given view. Overrides can be applied to an entire point cloud instance, or to specific scans within that instance. Options for the overrides include making the cloud non-visible, setting it to a fixed color, or to color gradients based on elevation, normals, or intensity.

The property

- PointCloudInstance.SupportsOverrides

identifies point clouds which support override settings (clouds which are based on .rcp or .rcs files).

Point cloud scans

The method

- PointCloudInstance.GetScans()

returns a list of scans contained within the .rcp point cloud. Scan names can be used to set visibility and fixed color overrides independently for each scan in the PointCloudInstance.

IFC API changes

IFC export now External Service

The capability to override IFC export is now managed as an External Service. As such, the explicit interfaces in ExporterIFCRegistry are no longer needed and have been marked obsolete. It is now possible to register more than one external IFC export implementation in the same session, and manage the active implementation using the methods of the SingleServerService wrapper to IFC export.

IFC APIs moved to new assembly

Most IFC API classes have been moved from RevitAPI.dll to a new assembly: RevitAPIIFC.dll. The classes moved include all members of the Autodesk.Revit.DB.IFC namespace which enable development of a custom IFC exporter.

Any Add-In using any of the migrated APIs needs to reference the new DLL and rebuild to work in Revit 2014.

Note that the APIs to invoke an IFC export and import:

- Document.Export(String, String, IFCExportOptions)
- Application.OpenIFCDocument(String)

and the related options class have not moved. Applications which export or import IFC files but do not provide custom implementations do not need to make changes.

PrintParameters

The property HideUnreferencedViewTages was renamed to HideUnreferencedViewTags.

Obsolete API Removal

API classes and methods previously marked Obsolete in Revit 2013 or earlier have been removed from the API.

Classes removed

- CurtainCellSetIterator and CurtainCellSet - Use generic .NET collection classes instead.
- PipeMaterialType - replaced by Material
- Batch creation argument class types - batch creation is not needed with changes to Revit API regeneration
- ProfiledWallCreationData
- RectangularWallCreationData
- RoomCreationData
- TextNoteCreationData
- Subclasses of ConnectorElement have been removed, access these elements through the parent class
- ElectricalConnector
- ElectricalConnector.SystemType - query the parameter RBS_ELEC_CIRCUIT_TYPE on ConnectorElement and cast to ElectricalSystemType
- DuctConnector
- DuctConnector.LinkedConnector - replaced by ConnectorElement.GetLinkedConnectorElement() and ConnectorElement.SetLinkedConnectorElement()
- DuctConnector.SystemType - query the parameter RBS_DUCT_CONNECTOR_SYSTEM_CLASSIFICATION_PARAM on ConnectorElement and cast to DuctSystemType
- PipeConnector
- PipeConnector.LinkedConnector replaced by ConnectorElement.GetLinkedConnectorElement() and ConnectorElement.SetLinkedConnectorElement()
- PipeConnector.SystemType - query the parameter RBS_PIPE_CONNECTOR_SYSTEM_CLASSIFICATION_PARAM on ConnectorElement and cast to PipeSystemType

Methods and Properties removed

Autodesk.Revit.Creation namespace

Application

- NewMaterialSet() - replaced by .net Generic collection classes
- NewElementArray() - replaced by .net Generic collection classes

Document

- NewAnnotationSymbol(XYZ ,AnnotationSymbolType ,View) - replaced by NewFamilyInstance(XYZ, FamilySymbol, View)
- NewAreaViewPlan(String,Level,AreaElemType) - replaced by ViewPlan.CreateAreaPlan()
- NewCurtainSystem(ReferenceArray,CurtainSystemType) - replaced by NewCurtainSystem2(ReferenceArray, CurtainSystemType)
- NewElectricalSystem(ElementSet,ElectricalSystemType) - replaced by NewElectricalSystem(ICollection<ElementId>, ElectricalSystemType)
- NewFamilyInstances(List<FamilyInstanceCreationData>) - replaced by NewFamilyInstances2(List<FamilyInstanceCreationData>)

- NewGroup(ElementSet) - replaced by NewGroup(System.Collections.Generic.ICollection<Autodesk.Revit.DB.ElementId>)
- NewRooms(Phase, Int32) - replaced by NewRooms2(Phase, Int32)
- NewRooms(Level,Phase) - replaced by NewRooms2(Level, Phase)
- NewRooms(Level) - replaced by NewRooms2(Level)
- NewRooms(List<RoomCreationData>) - there is no single equivalent that creates multiple rooms, this is not needed with Revit API control over regeneration
- NewSpaces(Phase,Int32) - replaced by NewSpaces2(Phase, Int32)
- NewSpaces(Level,Phase,View) - replaced by NewSpaces2(Level, Phase, View)
- NewTextNotes(List<TextNoteCreationData>) - replaced by NewTextNote()
- NewViewPlan(String,Level,ViewPlanType) - replaced by ViewPlan.Create(Document, ElementId, ElementId)
- NewView3D(XYZ) - replaced by View3D.CreateIsometric(Document, ElementId)
- NewViewSection(BoundingBoxXYZ) - replaced by ViewSection.CreateDetail()
- All Wall creation methods replaced by equivalent Wall.Create() methods:
- NewWall(CurveArray,WallType,Level,Boolean,XYZ)
- NewWall(CurveArray,WallType,Level,Boolean)
- NewWall(CurveArray,Boolean)
- NewWall(Curve,WallType,Level,Double,Double,Boolean,Boolean)
- NewWall(Curve,Level,Boolean)
- NewWalls(List<ProfiledWallCreationData> dataList) - there is no single equivalent that creates multiple walls, this is not needed with Revit API control over regeneration
- NewWalls(List<RectangularWallCreationData> dataList) - there is no single equivalent that creates multiple walls, this is not needed with Revit API control over regeneration

FamilyItemFactory

- NewDuctConnector(Reference,DuctSystemType) - replaced by ConnectorElement.CreateDuctConnector()
- NewPipeConnector(Reference,PipeSystemType) - replaced by ConnectorElement.CreatePipeConnector()
- NewElectricalConnector(Reference,ElectricalSystemType) - replaced by ConnectorElement.CreateElectricalConnector()

Autodesk.Revit.DB namespace

BaseArray

- CopyMembers - replaced by GetCopiedMemberIds()
- OrigMembers - replaced by GetOriginalMemberIds()

CurtainGrid

- UnlockedMullions - replaced by GetUnlockedMullionIds()
- Mullions - replaced by GetMullionIds()
- Cells - replaced by GetCurtainCells()
- UnlockedPanels - replaced by GetUnlockedPanelIds()
- Panels - replaced by GetPanelIds()
- VGridLines - replaced by GetVGridLineIds()
- UGridLines - replaced by GetUGridLineIds()

CurveElement

- LineStyles - replaced by GetLineStyleIds()

Document

- Delete(ElementSet elements) - replaced by Delete(System.Collections.Generic.ICollection<Autodesk.Revit.DB.ElementId> elementIds)
- WorksharingCentralFilename - replaced by ModelPathUtils.ConvertModelPathToUserVisiblePath(Document.GetWorksharingCentralModelPath())
- PrintSettings - replaced by GetPrintSettingIds()
- Element/get_Element - replaced by GetElement method

Element

- PhaseDemolished - replaced by DemolishedPhaseId
- PhaseCreated - replaced by CreatedPhaseId

FamilyInstance

- GetCopings() - replaced by GetCopingIds()
- SetCopings(ElementSet) - replaced by SetCopingIds(ICollection<ElementId> cutters)
- SubComponents - replaced by GetSubComponentIds()

Floor

- SpanDirectionSymbols - replaced by GetSpanDirectionSymbolIds()

GeometryElement

- Objects - replaced by GetEnumerator()

Group

- Ungroup() - replaced by UngroupMembers()
- Members - replaced by GetMemberIds()

LinearArray

- CopyMembers - replaced by GetCopiedMemberIds()
- OrigMembers - replaced by GetOriginalMemberIds()

Material

- GetMaterialAspectPropertySet(MaterialAspect) - replaced by GetStructuralAssetId() and GetThermalAssetId()
- SetMaterialAspect(MaterialAspect,ElementId,Boolean) - replaced by SetStructuralAssetId() and SetThermalAssetId()
- SetMaterialAspectToIndependent(MaterialAspect) - replaced by SetStructuralAssetId() and SetThermalAssetId()

MEPSystem

- Remove(ElementSet) - replaced by Remove(ICollection<ElementId>)

Part

- ParentDividedElementId - no replacement, concept is removed from Revit
- OriginalDividedElementId - no replacement, concept is removed from Revit
- GetDividedParents() - no replacement, concept is removed from Revit

PartMaker

- IsElementDivided(ElementId elemId) - replaced by IsSourceElement(ElementId)
- GetDividedElementIds() - replaced by GetSourceElementIds()
- SetDividedElementIds(ICollection<ElementId>) - replaced by SetSourceElementIds(ICollection<ElementId>)

PartUtils

- AreElementsValidForDivide(Document, ICollection<ElementId>) - replaced by ArePartsValidForDivide(Document, ICollection<ElementId>)
- AreElementsValidIntersectingReferences(Document, ICollection<ElementId>) - replaced by PartMakerMethodToDivideVolumes.AreElementsValidIntersectingReferences(Document, ICollection<ElementId>)
- IsValidSketchPlane(Document, ElementId) - replaced by PartMakerMethodToDivideVolumes.IsValidSketchPlane(Document, ElementId)

- SetOffsetForIntersectingReference() - replaced by PartMakerMethodToDivideVolumes.SetOffsetForIntersectingReference()
- GetOffsetForIntersectingReference() - replaced by PartMakerMethodToDivideVolumes.GetOffsetForIntersectingReference()
- PartMakerUsesReference() - replaced by PartMakerMethodToDivideVolumes.PartMakerUsesReference()
- IsMaxDivisionDepthReached(Document, ElementId) - no replacement, concept is removed from Revit
- GetDividedParents(Part) - no replacement, concept is removed from Revit
- PlanTopology
- Rooms - replaced by GetRoomIds()

PropertySetElement

- Create(Document, MaterialAspect) - replaced by Create(Document, StructuralAsset)

RadialArray

- CopyMembers - replaced by GetCopiedMemberIds()
- OrigMembers - replaced by GetOriginalMemberIds()

SpatialFieldManager

- UpdateSpatialFieldPrimitive(Int32,FieldDomainPoints,FieldValues) - replaced by UpdateSpatialFieldPrimitive(Int32, FieldDomainPoints, FieldValues, Int32)
- SetUnits(IList<string>, IList<double>) - replaced by AnalysisResultSchema.SetUnits() and SetResultSchema()

View

- ApplyTemplate(View viewTemplate) - replaced by ViewTemplateId/ApplyViewTemplateParameters(View viewTemplate)
- Hide(ElementSet elemSet) - replaced by HideElements(System::Collections::Generic::ICollection<Autodesk::Revit::DB::ElementId^>^ elementIdSet)
- Unhide(ElementSet elemSet) - replaced by UnhideElements(System::Collections::Generic::ICollection<Autodesk::Revit::DB::ElementId^>^ elementIdSet)

View3D

- EyePosition - replaced by ViewOrientation3D.EyePosition/View.Origin
- ViewSheet
- AddView(View,UV) - replaced by Viewport.Create(Document, ElementId, ElementId, XYZ)

Autodesk.Revit.DB.Plumbing namespace

PipeType

- Roughness - replaced by Segment.Roughness
- Material - replaced by PipeSegment.MaterialId

Autodesk.Revit.DB.Structure namespace

AnalyticalModel

- CanDisable() - no replacement, concept removed from Revit
- IsValidAnalyticalProjectionType(AnalyticalDirection,AnalyticalProjectionType) - replaced by IsValidProjectionType(AnalyticalElementSelector, AnalyticalDirection, AnalyticalProjectionType)

AreaReinforcement

- NumBarDescriptions - replaced by GetRebarInSystemIds()
- BarDescription - replaced GetRebarInSystemIds()

- Curves - replaced by GetCurveElementIds()

BeamSystem

- GetAllBeams() - replaced by GetBeamIds()

PathReinforcement

- BarDescription - replaced by GetRebarInSystemIds()
- Curves - replaced by GetCurveElementIds()

Rebar

- GetCenterlineCurves(Boolean) - replaced by GetCenterlineCurves(Boolean,Boolean,Boolean)
- DistributionPath - replaced by GetDistributionPath()
- RebarShape - replaced by RebarShapeId
- Host - replaced by Rebar.GetHostId() and SetHostId(Document, ElementId)
- BarType - replaced by Element.GetTypeId() and Element.ChangeTypeId(ElementId)

Major enhancements to the Revit API

Worksharing API enhancements

Reload Latest

The method:

- Document.ReloadLatest()

Fetches changes from central (due to one or more synchronizations with central and merges them into the current session. After this call finishes, use

- Document.HasAllChangesFromCentral()

to confirm that there were no Synchronizations with Central performed during execution of ReloadLatest.

Synchronize with Central

The method:

- Document.SynchronizeWithCentral()

performs a reload latest until the model in the current session is up to date and then saves changes back to central. A save to central is performed even if no changes were made.

Element ownership

The following methods affect element and workset ownership:

- WorksharingUtils.CheckoutElements - Obtains ownership for the current user of as many specified elements as possible.
- WorksharingUtils.CheckoutWorksets - Obtains ownership for the current user of as many specified worksets as possible.
- WorksharingUtils.RelinquishOwnership - Relinquishes ownership by the current user of as many specified elements and worksets as possible, and grants element ownership requested by other users on a first come, first served basis.

Create new local

The new method:

- WorksharingUtil.CreateNewLocal()

takes a path to a central model and copies the model into a new local file for the current user.

Enable Worksharing

The new method Document.EnableWorksharing will enable worksharing in a project.

Enhancements to interactions with links

Several improvements have been made to increase API functionality when working with RVT links.

Identifying links

The property:

- Document.IsLinked

identifies if a document is in session because it is a linked Revit file.

Obtaining linked documents

The method:

- RevitLinkInstance.GetLinkedDocument()

gets the document that corresponds to an Revit Link instance.

Link creation

Two functions have been added to RevitLinkOptions, used to specify options for RevitLinkType.Create.

- RevitLinkOptions.GetWorksetConfiguration()
- RevitLinkOptions.SetWorksetConfiguration()

This allows the user to specify which worksets should be opened when creating and loading a new Revit link.

Link load and unload

The methods

- RevitLinkType.Load()
- RevitLinkType.LoadFrom()
- RevitLinkType.Unload()

allow a link to be loaded or unloaded, or to be loaded from a new location. These methods regenerate the document. The document's Undo history will be cleared by these methods. As a result, this function and others executed before it cannot be undone. All transaction phases (e.g. transactions, transaction groups and sub-transactions) that were explicitly started must be finished prior to calling this method.

Link shared coordinates

The methods:

- RevitLinkType.SavePositions()
- RevitLinkType.HasSharedCoordinatesChanges()

support save of shared coordinates changes back to the linked document. While this operation does not clear the document's undo history, you will not be able to undo this specific action, since it saves the link's shared coordinates changes to disk.

Link path type

The property:

- RevitLinkType.PathType

allows read and modification of the path type associated with a link.

Conversion of geometric references

The new APIs:

- Reference.LinkedElementId
- Reference.CreateLinkReference(RevitLinkInstance)
- Reference.CreateReferenceInLink()

allow conversion between Reference objects which reference only the contents of the link and Reference objects which reference the host. This allows an application, for example, to look at the geometry in the link, find the needed face, and convert the reference to that face into a reference in the host suitable for use to place a face-based instance. Also, they would allow you to obtain a reference in the host (e.g. from a dimension or family) and convert it to a reference in the link, suitable for use in Element.GetGeometryObjectFromReference().

Room tag creation from linked rooms

The new method:

- NewRoomTag(LinkElementId roomId, UV point, ElementId viewId)

can create room tags in Revit Links.

Picking in links

PickObject() and PickObjects() now allow selection of elements in RVT Links.

Views & Display

Graphic Display options

These new members expose read and write of graphic display options:

- View.GetBackground()
- View.SetBackground()
- View.ShadowIntensity
- View.SunlightIntensity
- View.SurfaceTransparency
- View.ShowEdges
- View.ShowSilhouettes
- View.SilhouetteLineStyleId

Category classes override

Display of category classes may be overridden by the user. The new properties:

- View.AreModelCategoriesHidden
- View.AreAnnotationCategoriesHidden
- View.AreAnalyticalModelCategoriesHidden
- View.AreImportCategoriesHidden

allow an application to determine if a class of categories has been completely hidden. Note that Category.Visible[View] and View.GetVisibility(Category) does not look to the category classes when it returns the individual visibility status.

Category override

Display of categories can be overridden. This can be done with the new class OverrideGraphicSettings and the new View methods:

- SetCategoryOverrides
- GetCategoryOverrides
- IsOverrideValidForCategory
- IsCategoryOverridable

Element Override

Display of elements can be overridden with the new methods:

- View.GetElementOverrides
- View.SetElementOverrides.

View Filters

A new set of methods on the View class allow getting, setting, adding, and removing filters. Filters can be created using the ParameterFilterElement class and its Create method which existed in previous versions of the Revit API.

Non-rectangular crop region

Views can now be assigned a crop region which is non-rectangular. The new View members:

- View.GetCropRegionShapeManager()
- View.GetCropRegionShapeManagerForReferenceCallout()
- View.GetReferenceCallouts()

provide access to a ViewCropRegionShapeManager for the view or for a displayed reference callout.

This class allows access and modification of the crop region shape:

- ViewCropRegionShapeManager.GetCropRegionShape()
- ViewCropRegionShapeManager.SetCropRegionShape()
- ViewCropRegionShapeManager.IsCropRegionShapeValid()
- ViewCropRegionShapeManager.RemoveCropRegionShape()
- ViewCropRegionShapeManager.ShapeSet
- ViewCropRegionShapeManager.Valid

The properties:

- View.CropBoxActive
- View.CropBoxVisible

also apply to non-rectangular crop regions which may be assigned to a given view.

Viewport

The new property

- Viewport.Rotation

controls the rotation of the viewport on the sheet.

The new method

- Viewport.MoveCenterTo()

moves the viewport so that the center of the box outline (excluding the viewport label) is at a given point.

The new method

- Viewport.GetBoxCenter()

returns the center of the outline of the viewport on the sheet, excluding the viewport label.

Exploded Views

The new DisplacementElement class can be used to cause elements to appear displaced from their actual location to create views such as this one where the roof has been moved in the positive Z direction. The DisplacementPath class creates an annotation that depicts the movement of the element from its actual location to its displaced location.

Revisions on sheets

The new methods:

- ViewSheet.GetAllProjectRevisionIds()
- ViewSheet.GetAdditionalProjectRevisionIds()
- ViewSheet.SetAdditionalProjectRevisionIds()

provide access to the ids of project revision elements associated to a particular sheet. GetAllProjectRevisionIds() returns project revisions ordered from lowest to highest by project revision sequence number. The results include revisions associated to a revision cloud visible on the sheet and revisions that have been additionally included using the Revisions On Sheet parameter. GetAdditionalProjectRevisionIds() and SetAdditionalProjectRevisionIds() access specifically the additional revisions added using the Revisions On Sheet parameter.

Note that there is no special class for project revision elements, but that they can be accessed as Elements by filtering on category BuiltInCategory.OST_Revisions.

User interface customization

UIView

Zoom operations

The new methods:

- UIView.ZoomToFit()
- UIView.ZoomSheetSize()
- UIView.Zoom(double zoomFactor)

provide shortcuts to quickly adjust the zoom of the graphical view window.

Close view

The new method:

- UIView.Close()

  closes a visible view window. Note that the last open view for a given document cannot be closed by this method.

PreviewControl

The new property:

- PreviewControl.UIView

  returns a UIView handle to the preview view. This allows an application to manipulate the zoom and pan settings of the preview view.

The new property:

- PreviewControl.ScrollbarVisibility

  accesses the visibility setting for the preview view scrollbars.

Command API

Command event

The event

- AddInCommand.BeforeExecuted

  occurs before the command associated with an AddInCommandBinding executes. This event is read-only, an application can react to this event but cannot make changes to documents, or affect the invocation of the command in any way.

Command posting

The method

- UIApplication.PostCommand()

  posts a command to Revit. Revit will invoke it when control returns from the current API context. Only certain commands can be posted using this method:

1. Commands listed in the Autodesk.Revit.UI.PostableCommand enumerated type
2. External commands created by any add-in

   This restriction prevents posting of contextual commands (because Revit's command framework cannot directly access the accessibility of some contextual commands).

   Note that only one command may be posted to Revit at a given time. If a second command is posted from any API application, the method throws an InvalidOperationException.

The command must be accessible to be executed. This is determined only at the point where Revit returns from the API context, and therefore a failure to execute the command because the command is not currently accessible will not be reported directly back to the application that posted the command.

To use this API, the application must obtain a RevitCommandId handle for the command. This can be done using either

- RevitAddInCommand.LookupCommandId(String)
- RevitAddInCommand.LookupPostableCommandId(PostableCommand)

The method

- UIApplication.CanPostCommand

identifies if the given command can ever be posted (that is, it is a member of PostableCommand or an external command). It does not identify the command can be posted at the given timeframe (that is, it will not attempt to determine if the command is currently accessible).

Dockable Dialog Panes

Revit now allows addins to register WPF dialogs to dock into the top, left, right, and bottom of the main Revit window, as well as to be added as a tab to an existing system pane, such as the project browser. Dialogs can be registered with UIApplication.RegisterDockablePane and a user-implementation of the IDockablePaneProvider interface. Dockable panes can later be retrieved, shown, and hidden through UIApplication.GetDockablePane() and DockablePane.Show() and Hide().

Dimensions & annotations API

Multi-reference annotations for rebar

Revit now supports annotations pointing to more than one reference, consisting of a dimension and associated tag These annotations can be used to label and dimension Rebar elements, and are labeled in the user interface as “Multi-rebar annotations.”

The API exposes this through:

- MultiReferenceAnnotation - the annotation object instance. This class includes a reference to the associated dimension and tag element.
- MultiReferenceAnnotationType - the annotation type. The type specifies the tag and dimension types to be used in the multi-reference annotation, as well as associated display settings.
- MultiReferenceAnnotationOptions - an options class used to create a new MultiReferenceAnnotation, including specification of the associated elements and options for the dimension and tag placement.
- IndependentTag.MultiReferenceAnnotationId - returns the ElementId of the MultiReferenceAnnotation that owns a tag.

- Dimension.MultiReferenceAnnotationId - returns the ElementId of the MultiReferenceAnnotation that owns a dimension.

Dimension alternate units

New properties of DimensionStyle provide access to alternate units settings:

- DimensionStyle.AlternateUnits
- DimensionStyle.GetAlternateUnitsFormatOptions()
- DimensionStyle.SetAlternateUnitsFormatOptions()
- DimensionStyle.AlternateUnitsPrefix
- DimensionStyle.AlternateUnitsSuffix

Dimension unit type

The property:

- DimensionStyle.UnitType

returns the type of units supported by this dimension style (length, angle, or slope).

Add-ins and macros

Automatic load of add-ins without restarting Revit

Revit now automatically loads addins from newly added .addin manifest files without restarting Revit.

After placing a new .addin manifest file into one of the dedicated addins folders, the running Revit session will attempt to load the corresponding addin. Loaded ExternalCommands will be added to the External Tools pulldown menu. ExternalApplication and ExternalDBApplication OnStartup methods will be executed upon loading. An addin's installer may leverage this feature - but it must do so by creating the new .addin file at the end of installation to the dedicated Revit addin folders (the installer must ensure that the addin's assembly is already deployed to the target machine and the assembly path can be resolved in the add-in manifest file).

Add-ins may decline the ability for Revit to load the .addin in the middle of a session. To decline this options, add an <AllowLoadingIntoExistingSession> tag (set to “NO”) in the add-in manifest file.

Note that when Revit starts an add-in in the middle of the session, some add-in logic may not function identically because of the different interactions with the session. Specifically:

- If the application's goal is to prevent something from happening, the application may not be able to handle the fact that this activity has already happened in the existing session.
- If the application's goal is to manage external information in synch with documents loaded in the session, the application may not be able to handle documents that were loaded before the application started.
- If the application's logic depends on the ApplicationInitialized event, this event has already been called before the add-in was loaded.

Also, some add-ins may not be able to fully initialize when loading in the middle of the session. This is because some activities must take place at the start of the Revit session:

- Creation of custom failure definitions
- Establishment of a system-wide IFailureProcessor to handle all failures.
- Registering ExternalServices.

Revit also offers a new UI API method

- UIApplication.LoadAddIn()

to programmatically load the add-in(s) listed in the provided add-in manifest file. Typically, this API would not be needed because Revit is already loading new .addin files automatically.

MacroManager API

Revit now support an API for listing, creating, removing, editing, debugging, and running macros through several supporting classes and enums

- MacroManager

- Available at the DB.Document or ApplicationServices.Application level; manages the querying, creation, and removal of macro modules
- UIMacroManager

- Available at the UIDocument or UIAppliciation level, manages adding, removing, and editing UI-level modules.
- MacroModule

- Manages the querying, creation, removal, and running of specific macro methods
- ModuleSettings
- A collection of top-level properties of a MacroModule
- Macro

- Represents a single, runnable macro method.
- UIDocumentMacroOptions

- Represents security options for UIDocument-level macros.
- DocumentMacroOptions

- Represents security options for DB.Document-level macros.
- ApplicationMacroOptions

- Represents security options for Application-level macros
- MacroLanguageType

- Represents the language of a given macro, C#, VB.NET, Ruby, or Python
- MacroLevel

- Represents whether a macro is associated with a document the Revit application

- ModuleStatus
- Represents the compiled, loaded, or built status of a MacroModule.

The MacroManager APIs are all in the Autodesk.Revit.DB.Macros namespace and are available in RevitAPIMacros.dll and RevitAPIUIMacros.dll

Macro Attributes

The attributes

- Autodesk.Revit.UI.Macros.AddinAttribute
- Autodesk.Revit.UI.Macros.VendorIdAttribute

and the interface

- Autodesk.Revit.UI.Macros.IEntryPoint

have been moved to the namespace Autodesk.Revit.DB.Macros.

Parameters

Shared parameter - create with specified GUID

The new method

- Definitions.Create(String, ParameterType, bool, GUID)

supports creation of new ExternalDefinition objects (shared parameter definitions) which use the designated GUID instead of a random GUID. This allows an application to programmatically create consistent definitions for shared parameters without a shared parameter file or copying elements from one project to another.

Dimension.Label

The property

- Dimension.Label

Has been replaced by a new property

- Dimension.FamilyLabel

As this label applies only to dimension in families and their corresponding family parameter.

Family Parameters

The new property

- FamilyParameter.IsShared

identifies if the family parameter is a shared parameter.

Geometry APIs

JoinGeometryUtils

Revit now supports APIs for joining, unjoining, querying join state, and changing join order of elements in a model through the JoinGeometryUtils class.

Extensible Storage

ExtensibleStorage API changes

The method

- Element.GetEntitySchemaGuids()

returns the Schema Guids of any Entities present on an Element.

The methods

- Schema.GetField()
- Schema.ListFields()

are now restricted based on the read permission defined in the Schema.

ExtensibleStorageFilter

An ExtensibleStorageFilter is a new ElementQuickFilter for finding elements that contain an extensible storage entity of a given Schema Guid.

Translation

Export to Navisworks

The new function:

- Document.Export(String, String, NavisworksExportOptions)

exports a Revit project to the Navisworks .nwc format. Note that in order to use this function,you must have a compatible Navisworks exporter add-in registered with your session of Revit. If there is no compatible exporter registered, the function will throw OptionalFunctionalityNotAvailableException. Use

- OptionalFunctionalityUtils.IsNavisworksExporterAvailable()

to check if there is an exporter registered.

Import/Link SAT

The functions:

- Document.Import(String, SATImportOptions, View)
- Document.Link(String, SATImportOptions, View)

import or link an SAT file into the document.

Import/Link SketchUp

The functions:

- Document.Import(String, SKPImportOptions, View)
- Document.Link(String, SKPImportOptions, View)

import or link an SKP file into the document.

Import DWF Markups

The function:

- Document.Import(String, DWFImportOptions)

imports DWF markups into the document.

Export tables

The new classes

- ExportLayerTable
- ExportLinetypeTable
- ExportPatternTable
- ExportFontTable
- ExportLineweightTable

expose read and write access to the tables used for mapping on export to various formats such as DWG, IFC and DGN.

Site

Editing a TopographySurface

Editing the points in a TopographySurface now requires establishment of an edit scope. The class

- TopographyEditScope

allows an application to create and maintain an editing session for a TopographySurface. Start and end of a TopographyEditScope will also start and end a transaction group. After the TopographyEditScope is started, an application can start transactions and edit the topography surface. Individual transactions the application creates inside TopographyEditScope will not appear in the undo menu. All transactions committed during the edit mode will be merged into a single one which will bear the given name passed into TopographyEditScope constructor.

The method:

- TopographySurface.AddPoints()

has been modified to operate with the edit scope, and thus cannot be called outside of an edit scope. The function no longer regenerates the document.

The new methods:

- TopographySurface.DeletePoints()
- TopographySurface.MovePoint()
- TopographySurface.MovePoints()
- TopographySurface.ChangePointElevation()
- TopographySurface.ChangePointsElevation()

provide the ability to modify and delete existing points in the TopographySurface. Multiple calls to these functions can be included in the same edit scope.

Reading points from a TopographySurface

The new method:

- TopographySurface.ReadPoints()

returns a list of the points stored in the surface.

The method:

- TopographySurface.FindPoints()

filters and returns only the points of the topography surface which lie within the input bounding box.

The method:

- TopographySurface.ContainsPoint()

identifies if the input point is found in points stored in the surface.

The methods:

- TopographySurface.GetBoundaryPoints()
- TopographySurface.IsBoundaryPoint()

identify points which are a part of the boundary of the surface.

Validation

Several new validation methods have been added to the TopographySurface class:

- IsValidRegion() - Identifies whether the points can construct a valid region for a topography surface.
- ArePointsDistinct() - Identifies whether the input points are distinct in XY location.

These methods are newly used in AddPoints() to prevent problematic inputs from causing issues.

SiteSubRegion

The class SiteSubRegion represents a proxy class exposing the interfaces needed to access details of a subregion. In the Revit database, both TopographySurface elements and subregion elements are represented by the same TopographySurface element subclass, however, in the Revit API, this SiteSubRegion class exists to separate the interfaces for subregions from those of topography surfaces. The SiteSubRegion class offers these APIs:

- SiteSubRegion.Create() - creates a new subregion given a boundary consisting of one or more curve loops.
- SiteSubRegion.IsValidBoundary() - validates the input boundary for a new subregion
- SiteSubRegion.GetBoundary() - returns the boundary curves
- SiteSubRegion.SetBoundary() - sets the boundary for the subregion
- SiteSubRegion.TopographySurface - the TopographySurface element which this subregion represents
- SiteSubRegion.HostId - the identifier of the topography surface hosting this subregion

BuildingPad

The class BuildingPad represents a building pad element in Revit. It inherits from CeilingAndFloor and thus offers inherited capabilities from HostObject as well. The API exposes the following specific capabilities around BuildingPads:

- BuildingPad.Create() - Creates a new BuildingPad given a boundary, type and level.
- BuildingPad.GetBoundary() - Returns the sketched boundary curves.
- BuildingPad.SetBoundary() - Assigns a new boundary to the BuildingPad.
- BuildingPad.HostId - The element id of the topography surface hosting this BuildingPad.

The type of the building pad is represented by BuildingPadType. This inherits from HostObjAttributes and offers inherited capabilities from this parent, including access to the CompoundStructure of the type. The API exposes these new capabilities for BuildingPadTypes:

- BuildingPadType.CreateDefault() - Creates a new default BuildingPadType in the document.

MEP

Externalized Calculations

Several new interfaces and classes based on external services are supported for pipe and duct calculations.

- Several new interfaces deriving from IExternalServer to support friction, flow, and pressure drop for pipes and ducts have been added.
- IPipePlumbingFixtureFlowServer
- IPipePressureDropServer
- IDuctPressureDropServer
- IDuctFittingAndAccessoryPressureDropServer

- IPipeFittingAndAccessoryPressureDropServer
- IDuctFittingAndAccessoryPressureDropUIServer
- IPipeFittingAndAccessoryPressureDropUIServer
- Several new classes implementing IExternalData to support friction, flow, and pressure drop for pipes and ducts have been added.
- PipePlumbingFixtureFlowData
- PipePressureDropData
- DuctPressureDropData
- DuctFittingAndAccessoryConnectorData
- PipeFittingAndAccessoryConnectorData
- Several new classes implementing ISingleServerService to support friction, flow, and pressure drop for pipes and ducts have been added.

- PipePlumbingFixtureFlowService
- PipePressureDropService
- DuctPressureDropService
- DuctFittingAndAccessoryPressureDropService
- PipeFittingAndAccessoryPressureDropService
- DuctFittingAndAccessoryPressureDropUIService
- PipeFittingAndAccessoryPressureDropUIService

ElectricalLoadClassificationData

ElectricalLoadClassificationData has several new string properties corresponding to the load classification section of the electrical panel schedule.

- LoadSummaryDemandFactorLabel
- PanelConnectedLabel
- PanelEstimatedLabel
- PanelConnectedCurrentLabel
- PanelEstimatedCurrentLabel
- ActualElecricalLoadNameLabel

CSV Fitting Parameter Removal

Because CSV files are no longer used to drive MEP fitting parameters, Revit supports a new set of APIs to manage fitting parameters through several classes

- FamilySizeTableManager - manages importing and exporting of legacy CSV data and size tables.
- FamilySizeTable - manages specific sizes of fittings.
- FamilySizeTableColumn - manages a specific dimension of a given size in a size table
- FamilySizeTableErrorInfo - reports any errors when importing an file with CSV size table into a FamilySizeTable

Fitting Angle Settings

The members:

- DuctSettings.FittingAngleUsage
- DuctSettings.GetSpecificFittingAngles()
- DuctSettings.GetSpecificFittingAngleStatus()
- DuctSettings.SetSpecificFittingAngleStatus()
- PipeSettings.FittingAngleUsage
- PipeSettings.GetSpecificFittingAngles()
- PipeSettings.GetSpecificFittingAngleStatus()
- PipeSettings.SetSpecificFittingAngleStatus()
- ElectricalSetting.FittingAngleUsage
- ElectricalSetting.GetSpecificFittingAngles()
- ElectricalSetting.GetSpecificFittingAngleStatus()
- ElectricalSetting.SetSpecificFittingAngleStatus()

provide access to fitting angle usage settings for ducts, pipes, cable trays and conduits.

Duct Settings

Duct settings for size prefixes and suffixes, annotations, and air density and viscosity may now be set through the DuctSettings class.

Curve Creation

Pipes, Ducts, FlexPipes, and FlexDucts can now be created with a Pipe, Duct, FlexPipe, or FlexDuct SystemTypeId as a parameter on their respective static Create() methods.

ConnectorElement

The methods

- ConnectorElement.CreatePipeConnector(Document, Document, PipeSystemType, Reference, Edge)
- ConnectorElement.CreateDuctConnector(Document, DuctSystemType, ConnectorProfileType, Reference, Edge)
- ConnectorElement.CreateElectricalConnector(Document, ElectricalSystemType, Reference, Edge)
- ConnectorElement.CreateConduitConnector(Document, Reference, Edge)
- ConnectorElement.CreateCableTrayConnector(Document, Reference, Edge)

allow creation of connector elements in families on centered on internal loops of a given face.

The property

- ConnectorElement.SystemType

accesses the MEPSystemType of the connector

The members:

- ConnectorElement.Direction
- ConnectorElement.FlipDirection()

access the direction of the connector.

Connect Air Terminal to Duct

The new method:

- MechanicalUtils.ConnectAirTerminalOnDuct()

connects an air terminal to a duct directly (without the need for a tee or takeoff). The current location of the air terminal will be projected to the duct centerline, and if the point can be successfully projected, the air terminal will be placed on the most suitable face of the duct.

General

- CableTray.CurveNormal returns the "up" direction vector of a cable tray segment.
- RoutingPreferenceGroup now supports a new rule type for Cap fittings.
- Caps can be automatically be placed on all open connections of a pipe with PlumbingUtils.PlaceCapOnOpenEnds(Document, ElementId, ElementId)

Reinforcement API

There are many new reinforcement and rebar-related methods and classes in Revit 2014.

AreaReinforcement has several new methods and properties.

- AreaReinforcement.Create – creates a new AreaReinforcement object from curves.
- AreaReinforcement.IsUnobscuredInView - checks if Area Reinforcement is shown unobscured in a view.
- AreaReinforcement.SetUnobscuredInView - sets Area Reinforcement to be shown unobscured in a view.

PathReinforcement has several new methods and properties.

- PathReinforcement .Create – creates a new PathReinforcement object from curves.
- PathReinforcement .AdditionalOffset - gets and sets length offset.
- PathReinforcement .IsUnobscuredInView - checks if PathReinforcement is shown unobscured in a view.
- PathReinforcement .SetUnobscuredInView - sets PathReinforcement to be shown unobscured in a view.

FabricArea has several new methods and properties.

- FabricArea.Create - now supports an origin point of direction parameter.
- FabricArea.GetReinforcementRoundingManager - returns an object for managing reinforcement rounding override settings.

FabricSheet has several new methods and properties.

- FabricSheet.Create() - new static factory method for creation.

- FabricSheet.GetLocalCoordinateSystem()
- FabricSheet.SetLocalCoordinateSystem()
- FabricSheet.PlaceInHost()
- FabricSheet.GetReinforcementRoundingManager() - returns an object for managing reinforcement rounding override settings.
- FabricSheet.HostId
- FabricSheet.FabricLocation - the FabricSheet location in the host.
- FabricSheet.CoverOffset - the additional cover offset of the FabricSheet.
- FabricSheet.IsCoverOffsetValid() - identifies if the specified value is valid for use as a cover offset.

FabricSheetType

- FabricSheetType.GetReinforcementRoundingManager() - returns an object for managing reinforcement rounding override settings.

Rebar has several new methods and properties.

- Rebar.ComputeDrivingCurves() - compute the driving curves.
- Rebar.GetRebarConstraintsManager() - returns an object for managing the external constraints on the Rebar element.
- Rebar.GetReinforcementRoundingManager() - returns an object for managing reinforcement rounding override settings.
- Rebar.HookAngleMatchesRebarShapeDefinition() - checks that the hook angle of the specified RebarHookType matches the hook angle used in the Rebar's RebarShape at the specified end of the bar.
- Rebar.CanUseHookType - checks if the specified RebarHookType id is of a valid RebarHookType for the Rebar's RebarBarType.
- Rebar.ConstraintsCanBeEdited - returns true, if the Rebar element's external constraints are available for editing.
- Rebar.GetHookOrientation() - returns the orientation of the hook plane at the start or at the end of the rebar with respect to the orientation of the first or the last curve and the plane normal.
- Rebar.SetHookOrientation() - defines the orientation of the hook plane at the start or at the end of the rebar with respect to the orientation of the first or the last curve and the plane normal.

RebarType has several new methods and properties.

- RebarType.SetHookTangentLength() - identifies the hook tangent length for a hook type.
- RebarType.GetReinforcementRoundingManager() - returns an object for managing reinforcement rounding override settings.

RebarBendData

- RebarBendData – new constructor to specify hook orientation.

RebarHandleType - a new enum to represent the various types of handles on a Rebar instance that can be joined to References.

RebarConstrainedHandle - a new class to represent a handle on a Rebar that can be joined to a reference.

RebarConstraintType - a new enum to represent the various types of constraints that can be applied to a RebarConstrainedHandle.

RebarConstraintTargetHostFaceType - a new enum to identify the individual face on a host element to which a Rebar handle is constrained.

RebarConstraint - a new class representing a constraint on the position of a Rebar Element or one of its handles.

RebarConstraintsManager -a new class used to obtain information about the constrained shape handles (RebarConstrainedHandles) on a Rebar element.

RebarDeformationType - new enum representing the type of deformation of rebar.

RebarHookType.HookAngleInDegrees - a new property representing the angle of a rebar hook.

RebarInSystem

- RebarInSystem.getReinforcementRoundingManager - returns an object for managing reinforcement rounding override settings.
- RebarInSystem.setUnobscuredInView - sets the element to be shown unobscured in a view.

RebarShape

- RebarShape.GetDefaultHookAngle - get the hook angle, expressed as an integral number of degrees (common values are 0, 90, 135, and 180).
- RebarShape.GetDefaultHookOrientation - gets the default rebar hook orientation.

ReinforcementRoundingSource - a new enum listing the possible sources for reinforcement rounding overrides.

ReinforcementRoundingManager - a new class providing access to reinforcement rounding overrides for structural elements.

RebarRoundingManager - new class providing access to element reinforcement roundings overrides.

FabricRoundingManager - a new class providing access to element reinforcement roundings overrides.

ReinforcementSettings

- ReinforcementSettings.GetRebarRoundingManager - returns an object for managing reinforcement rounding override settings used by RebarBarTypes, Rebar and RebarInSystem elements.
- ReinforcementSettings.GetFabricRoundingManager -returns an object for managing reinforcement rounding override settings used by FabricSheetType and FabricSheet elements.

Custom Export

The Custom Export API provides access to the rendering output pipeline through which Revit sends a processed model (its graphics 3D representation) to an output device. In the case of Custom Export, the

"device" is represented by a context object that could be any kind of a device, even a file (in the most common case, actually.) A client of Custom Export provides a context and invokes rendering of a model, upon which Revit starts processing the model and sends graphic data out via methods of the context. The data describes the model exactly as it would have appeared in Revit when the model is rendered. The data includes all geometry and material properties.

The following classes have been made available so far:

CustomExporter

A class that allows exporting 3D views via a custom export context. The Export method of this class triggers standard rendering process in Revit, but instead of displaying the result on screen or printer, the output is channeled through the given custom context that handles processing the geometric as well as non-geometric information.

IExportContext

An interface of which interface is used in a custom export of a Revit model. The instance of this class is passed in as a parameter of a CustomExporter. The methods are then called by Revit at times of exporting entities of the model.

Render Node Classes

Classes of which instance are sent to an export context during a custom export.

- RenderNode - base class for all output nodes
- ViewNode - represents a View
- GroupNode - base class for family and link instances
- InstanceNode - represents an instance of a family
- LinkNode - represents an external link
- ContentNode - base class from RPC, Light, and Daylight nodes
- RPCNode - represents an RPC object
- DaylightPortalNode - represents a daylight portal
- LightNode - represents a light
- MaterialNode - represents a material change

CameraInfo

A class that describes information about projection mapping of a 3D view to a rendered image. An instance of this class can be obtained via a property of ViewNode.

Paint API

The paint tool is now supported in the API

- Document.Paint
- Document.RemovePaint
- Document.IsPainted
- Document.GetPaintedMaterial

Component repeater API

Component repeaters can be used to replicate (repeat) elements hosted on repeating references. The result of the repeating operation is a collection of slots. Each slot contains one repeated component. The ComponentRepeater class provides the repeating functionality and access to the slots.

Each repeating reference is capable of hosting one point of an adaptive component. An initial pattern can be created by populating one or more repeating references with such points. Component repeaters can then be used to replicate the pattern to fill the rest of the repeating references in the particular repeating reference source.

The repeating references in repeating reference source are arranged in one or two dimensional arrays, allowing for different kinds of repeating:

- One dimensional source allows for repeating along a path.
- Two dimensional source allows for repeating across a grid.
- It is also possible to host a point on a zero dimensional reference (a point). This point will be shared by all slots. A zero dimensional source allows for repeating around a single point. It should not be used alone, but together with at least one other repeating reference source (typically one dimensional.) The point hosted on the zero dimensional source serves as a central point around which other points can be repeated on their respective repeating reference sources.

Multiple adaptive components may be hosted on one repeating reference source, and different points of one adaptive component may be hosted on different repeating reference sources, effectively allowing different points of an adaptive component to be repeated using different patterns.

The following classes provide access to the functionality of Component repeaters:

- Autodesk.Revit.DB.RepeatingReferenceSource
- Autodesk.Revit.DB.RepeaterBounds
- Autodesk.Revit.DB.RepeaterCoordinates
- Autodesk.Revit.DB.ComponentRepeater
- Autodesk.Revit.DB.ComponentRepeaterSlot

Small enhancements & API interface changes

API validation

No transactions from outside threads

Calling into the Revit API from outside threads and outside modeless dialogs has never been supported, but it was not strictly prohibited, meaning there would be no immediate exceptions when someone tries to modify model from outside of the supported API workflows. That has been changed. It is no longer possible to start a transaction unless the caller is inside a legitimate API call, such as an external command, event, updater, call-back, etc. An exception will be thrown if such attempt is made.

IsValidObject property

If a Revit native object contained within an API wrapper is destroyed, or creation of the corresponding native object is undone, the managed API object containing it is no longer valid. API methods cannot be called on invalidated wrapper objects. The property IsValidObject (added to many API classes) can be used to identify if the corresponding native object has gone out of scope.

Enumerated type validation

Automatic validation has been introduced for enumerated type inputs to API methods and properties. If an value is improperly cast to an enum and is not a valid member of that enum, an ArgumentOutOfRangeException will be thrown.

Elements

Copy & paste elements

The new methods:

- ElementTransformUtils.CopyElements(Document, ICollection<ElementId>, Document, Transform)
- ElementTransformUtils.CopyElements(View, ICollection<ElementId>, View, Transform)
- Transform ElementTransformUtils.GetTransformFromViewToView(View, View)

support copy and paste of arbitrary elements. The first overload supports copy within documents, or from document to document. The second overload also support copying within one document or between two documents, but specifically supports copy and paste of view-specific elements.

Materials

Materials applied with the Paint tool and their areas can be found by specifying "true" for the 'usePaintMaterial' argumement in Element.GetMaterialIds and Element.GetMaterialArea

Geometry

FreeForm element

A FreeFormElement is a form sub-type that contains non-parametric geometry created from an input solid outline. The element can be added to families, and can participate in joins and void cuts with other combinable elements. Planar faces of the element can be offset interactively and programmatically in the face normal direction.

- FreeFormElement.Create() - creates a new FreeForm element.
- FreeFormElement.SetFaceOffset() - offsets a planar face a certain distance in the face normal direction.

Solid & curve intersection

The new method

- Solid.IntersectWithCurve()

calculates the intersection between a closed volume Solid and a curve. There is an option to return details about the segments inside the volume, or outside. Both the curve segments and the parameters of the segments are available in the results.

Face/Face Intersection

The method

- Face.Intersect()

calculates the intersection of the input face with this face and returns the intersection results. The method can output the intersection geometry if the intersection consists of a single curve.

ReferenceIntersector & RVT Links

The new option findReferencesInRevitLinks allows ReferenceIntersector to find geometry in RVT links.

Rulings of RuledFace

The new method

- RuledFace.RulingsAreParallel

determines if the rulings of the ruled surface are parallel.

Detail elements

Detail element draw order

The class

- DetailElementOrderUtils

now includes routines to BringToFront, BringForward, SendToBack, SendBackward multiple detail members. The draw order of the members does not change relative to one another.

Stairs

StairsRunJustification

New values LeftExterior and RightExterior have been added to this enum to support jusitification to the left and right supports.

StairsLanding

The new members:

- CreateAutomaticLanding(Document, ElementId, ElementId)
- CanCreateAutomaticLanding(Document, ElementId, ElementId)

provide automatic landing(s) creation and creation validation between two stairs runs.

StairsRun

The new properties:

- StairsRun.ExtensionBelowRiserBase
- StairsRun.ExtensionBelowTreadBase

represent the extension/trim value for the run, depending upon whether the run begins with a riser or tread.

These replace the deprecated property:

- StairsRun.ExtensionBelowBase

which now accesses whichever property listed above depending on the run's configuration.

StairsComponentConnection

The new methods:

- StairsRun.GetConnections()
- StairsLanding.GetConnections()

provide information about connections among stairs components (run to run, or run to landing).

Parameters & filters

Parameter.AsValueString()

This method previously was implemented for only Integer and Double values. It now can also be used with Enums and ElementIds. Optionally it can accept a FormatOptions object to specify units formatting of the returned string.

Parameter.Definition.UnitType

This new property provides access to the UnitType of a parameter definition.

Parameter variance among group instances

The new members:

- InternalDefinition.VariesAmongGroups
- InternalDefinition.SetVariesAmongGroups(Document)

support read and write to the option that the parameter value can vary across groups. This can be changed only for non-built-in parameters.

FilterCategoryRule

The new class FilterCategoryRule can be used in the definition of a ParameterFilterElement. It represents a filter rule that matches elements of a set of categories.

The related method:

- ParameterFilterElement.AllCategoriesFilterable()

has been replaced by

- FilterCategoryRule.AllCategoriesFilterable()

Miscellaneous

ThermalAsset.SpecificHeat

This new property provides the specific heat value of a thermal asset in feet per Kelvin, squared-second.

AreaVolumeSettings

This new class provides access to settings related to volume and area computations. The old VolumeCalculationSetting class is obsolete.

Document.Delete()

This method previously returned null if the element or elements could not be deleted. It now will throw an exception in this situation.

Document level updaters

The new methods:

- UpdaterRegistry.IsUpdaterRegister(UpdaterId, Document)
- UpdaterRegistry.UnregisterUpdater(UpdaterId, Document)

provide the ability to inquire about and to unregister updaters that are associated to specific documents.

UIThemeManager

The static properties of UIThemeManager provide access to the current UI theme and the default theme.

Families & content

Family category

The property

- FamilyBase.FamilyCategory

can now be set. This allows the category of an family being edited to be changed.

SpatialElementCalculationLocation

The class SpatialElementCalculationLocation is used to specify the room or space where an element should be considered as placed.

This is a base class with currently subtypes of calculation location:

- SpatialElementCalculationPoint
- SpatialElementFromToCalculationPoints

A SpatialElementCalculationLocation can be added to the family by turning on the family's ROOM_CALCULATION_POINT parameter. Once the location has been added, you can access the marker position using the MarkerPosition property.

Note that the MarkerPosition property is no longer settable - this position is determined automatically.

SpatialElementFromToCalculationPoints

SpatialElementFromToCalculationPoints is a subclass of SpatialElementCalculationLocation used to specify the search points for a family instance that 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". The properties ToPosition and FromPosition govern the coordinates for these points.

Arc through points

The method

- CurveByPointsUtils.CreateArcThroughPoints()

creates an arc curve through three input ReferencePoints.

Events

DocumentChangedEventArgs

For the methods GetAddedElementIds()/GetModifiedElementIds() - internal Revit element ids that are not visible to UI and API are now excluded from the return set.

GetAddElementIds(ElementFilter)/GetModifiedElementIds(ElementFilter)

The new methods:

- DocumentChangedEventArgs.GetAddedElementIds(ElementFilter)
- DocumentChangedEventArgs.GetModifiedElementIds(ElementFilter)

only return ElementIds that pass the input element filter. This helps applications detect only specific changes of interest.

Reinforcement API

Reinforcement Length Tolerance

The new classes:

- RebarRoundingManager
- FabricRoundingManager

contain settings for rounding tolerance for rebar and fabric reinforcement elements. They can be assigned at the element instance level (Rebar, RebarInSystem, FabricArea, FabricSheet), at the type level (RebarType, FabricSheetType), or at the ReinforcementSettings level. Lower level settings override the setting of higher levels.

Major changes and renovations to the Revit API 2015

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.

Major changes and renovations to the Revit API - Subscription Release 2015

API changes

None

Major API additions

ComponentRepeater

New Methods

- ComponetRepeater.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.

Dimension API improvements

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.

DWFExportOptions

New Property

- ExportTexture **-** sets the option to export textures in 3D DWF files (default true)

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.

View3D

New methods:

- 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.

Minor API additions

BaseExportOptions

New Property:

- BaseExportOptions.PreserveCoincidentLines - Allows export to preserve coincident lines

ElectricalSetting

New Enum:

- ElectricalSetting.CircuitSequence - Gets and sets the circuit sequence numbering schema

New Properties:

- ElectricalSetting.CircuitNamePhaseA - Accesses the Phase A circuit name.
- ElectricalSetting.CircuitNamePhaseB - Accesses the Phase B circuit name.
- ElectricalSetting.CircuitNamePhaseC - Accesses the Phase C circuit name.

Workset

The new static method:

- Workset.Create()

creates a new Workset.

WorksetTable

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.

Application

The new property:

- Application.BackgroundColor

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

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.

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.

Major changes and renovations to the Revit API - 2016

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.StartPoint
LineLoad.Point[]	LineLoad.EndPoint
	LineLoad.SetPoints()
LineLoad.Force[]	
	LineLoad.ForceVector1
LineLoad.Force1	
	LineLoad.ForceVector2
LineLoad.Force2	
LineLoad.Moment[]	
	LineLoad.MomentVector1
LineLoad.Moment1	
	LineLoad.MomentVector2
LineLoad.Moment2	
LineLoad.UniformLoad	LineLoad.IsUniform
N/A	LineLoad.IsProjected
Document.NewLineLoad()	LineLoad.Create()
AreaLoad.NumLoops	
	AreaLoad.GetLoops()
AreaLoad.NumCurves[]	
	AreaLoad.SetLoops()
AreaLoad.Curve[]	
AreaLoad.Force[]	AreaLoad.ForceVector1
AreaLoad.Force1	AreaLoad.ForceVector2
	AreaLoad.ForceVector3

AreaLoad.Force2	
AreaLoad.Force3	
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.GetComponents()
LoadCombination.NumberOfComponents	LoadCombination.SetComponents()
LoadCombintation.UsageName[]	LoadCombination.GetUsageIds()
LoadCombination.NumberOfUsages	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.AssociatedLoadId
BoundaryConditions.associateWithLoad()	N/A
BoundaryConditions.Curve[]	BoundaryConditions.GetCurve()
BoundaryConditions.NumCurves	BoundaryConditions.GetLoops()
BoundaryConditions.Curve[]	

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.GetAlphanumericRevisionSettings()
RevisionSettings.GetRevisionAlphabet()	For 2016, GetRevisionAlphabet() returns information if the revision can be represented with alphabetic settings; otherwise, an exception is thrown.
	RevisionSettings.SetAlphanumericRevisionSettings()
RevisionSettings.SetRevisionAlphabet()	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.

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.

Major changes and renovations to the Revit API 2016 R2

API additions

Parameter API additions

Global Parameters

Global Parameters support controlling geometry constraints through special parameters defined in a project document. Global Parameters can be used for both labeling and reporting to/from dimensions, as well as setting values of instance parameters.

The new class

- GlobalParametersManager

provides the main access point to managing global parameters in project document. It offers the following members:

- AreGlobalParametersAllowed() - Tests whether global parameters are allowed in a document.
- GetAllGlobalParameters() - Returns all global parameters in a document.
- FindByName() - Find a global parameter by its name.
- IsUniqueName() - Test uniqueness of the name of a prospective global parameter.
- IsValidGlobalParameter() - Test if an Id is of a valid global parameter element.

The new class:

- GlobalParameter

contains methods to control and manipulate a single global parameter. Its most important members include:

- [static] Create() - Creates a new Global Parameter in the given document.
- GetAffectedElements() - Returns all elements of which properties are driven by this global parameter.
- GetAffectedGlobalParameters() - Returns all other global parameters which refer to this global parameter in their formulas.
- GetLabeledDimensions() - Returns all dimension elements that are currently labeled by this global parameter.
- CanLabelDimension() - Tests whether a dimension can be labeled by the global parameter.
- LabelDimension() - Labels a dimension with this global parameter.

- UnlabelDimension() - Un-labels a dimension that is currently labeled by this global parameter.
- GetLabelName() - Returns the name of this parameter's label, which is used to label dimension elements.
- SetDrivingDimension() - Set a dimension to drive the value of this parameter.
- IsValidFormula() - Tests that the given expression is a valid as formula for this parameter.
- GetFormula() - Returns the parameter's expression in form of a string.
- SetFormula() - Sets a formula expression for this parameter.
- GetValue() - Obtains the current value of the global parameter.
- SetValue() - Sets a new value of the global parameter.
- HasValidTypeForReporting() - Tests that the global parameter has data of a type that supports reporting.
- [static] IsValidDataType() - Tests whether the input Data Type is valid as a type of a global parameter.

- IsDrivenByDimension - Indicates whether this parameter is driven by a dimension or not.
- IsDrivenByFormula - Indicates whether this parameter is driven by a formula or not.
- IsReporting - Indicates whether this is a reporting global parameter or not.

The new class:

- ParameterValue

contains a value of a corresponding global parameter. It is a base class for derived concrete classes, one per each type of a parameter value:

- IntegerParameterValue
- DoubleParameterValue
- StringParameterValue
- ElementIdParameterValue
- NullParameterValue

All the derived classes have only one property:

- Value - gets or sets the value as the corresponding type.

New methods added the the Parameter class:

- CanBeAssociatedWithGlobalParameter() - Tests whether a parameter can be associated with the given global parameter.
- CanBeAssociatedWithGlobalParameters() - Tests whether a parameter can be associated with any global parameter.
- AssociateWithGlobalParameter() - Associates a parameter with a global parameter in the same document.
- DissociateFromGlobalParameter() - Dissociates a parameter from a global parameter.
- GetAssociatedGlobalParameter() - Returns a global parameter, if any, currently associated with a parameter.

Multiline Text parameter support

The new enumerated value:

- ParameterType.MultilineText

was added for creation and use of multi-line text parameters.

View API additions

TemporaryViewModes

The new class:

- TemporaryViewModes

carries data related to the state and properties of available temporary view modes. Access to an instance of this class is via the property:

- View.TemporaryViewModes

The class has the following methods and properties:

- TemporaryViewModes.DeactivateAllModes() - Deactivates all temporary modes that are currently active.
- TemporaryViewModes.DeactivateMode() - Deactivates the given temporary mode.
- TemporaryViewModes.GetCaption() - A text caption to use for the given mode.
- TemporaryViewModes.IsModeActive() - Tests whether a given mode is currently active or not.
- TemporaryViewModes.IsModeAvailable() - Tests whether a temporary view mode is currently available in the associated view.
- TemporaryViewModes.IsModeEnabled() - Tests whether a temporary view mode is currently enabled in the associated view.
- TemporaryViewModes.IsValidState() - Tests whether the given state is valid for the associated view and the context the view is currently in.
- TemporaryViewModes.PreviewFamilyVisibility - The current state of the PreviewFamilyVisibility mode in the associated view.
- TemporaryViewModes.RevealConstraints - The current state of the RevealConstraints mode in the associated view.
- TemporaryViewModes.RevealHiddenElements - The current state of the RevealHiddenElements mode in the associated view.
- TemporaryViewModes.WorskaringDisplay - The current state of the WorksharingDisplay mode in the associated view.

Convert dependent view to independent

The new function:

- View.ConvertToIndependent()

converts a dependent view to be independent.

Plan view underlay

The new methods:

- ViewPlan.GetUnderlayBaseLevel()
- ViewPlan.GetUnderlayTopLevel()
- ViewPlan.SetUnderlayBaseLevel()
- ViewPlan.SetUnderlayOrientation()
- ViewPlan.SetUnderlayOrientation()

provide access to the underlay levels and settings for plan views.

MEP API additions

FabricationPart - product list support

To specify a size, some FabricationPart elements, such as purchased duct and pipe fittings, have a Product Entry field in the Properties palette. In the API these FabricationPart elements are identified as having a "product list". The product list entries represent a catalog of available sizes for the selected part.

The following new members are added to support product list FabricationPart elements:

- FabricationPart.ProductListEntry - The product list entry index of the fabrication part. A value of -1 indicates the fabrication part is not a product list.
- FabricationPart.IsProductList()
- FabricationPart.GetProductListEntryName()
- FabricationPart.GetProductListEntryCount()
- FabricationPart.IsProductListEntryCompatibleSize() - Checks to see if this part can be changed to the specified product list entry without altering any connected dimensions.

MEP Fabrication API

The following new members have been added to the MEP fabrication and FabricationPart capabilities:

- FabricationPart.ServiceId
- FabricationPart.RotateConnectedPartByConnector()
- FabricationPart.RotateConnectedTap()
- FabricationServiceButton.ContainsFabricationPartType()

Electrical API additions

The new method:

- Wire.GetMEPSystems()

gets the system(s) to which the wire belongs.

The new property

- ElectricalSetting.CircuitRating

provides access to the default circuit rating for a newly created circuit.

Revit Link API additions

Link instance locations

The new function:

- RevitLinkInstance.MoveBasePointToHostBasePoint()

will move a RevitLinkInstance so that the link's base point and host project's base point are in the same location. This function does not set up a monitoring relationship.

The new method:

- RevitLinkInstance.MoveOriginToHostOrigin()

moves this link instance so that the internal origin of the linked document is aligned to the internal origin of the host document. This is a one-time movement and does not set up any shared coordinates relationship.

Local unload of Revit Links

The new method:

- RevitLinkType.UnloadLocally()

allows unloading a Revit link in a workshared file for the current user only. When another user opens their local model, the link will still be loaded for them. This method accepts an instance of a new interface class:

- ISaveSharedCoordinatesCallbackForUnloadLocally

The response to the method in this interface is used to control Revit when trying to unload locally a Revit link with changes in shared coordinates.

The new method:

- RevitLinkType.RevertLocalUnloadStatus()

turns off a user's local link override. If the link is loaded for other users, this function will reload the link. If the link is unloaded for other users, then the link will remain unloaded, but the local unload override will be cleared.

Major changes and renovations to the Revit API 2017

API changes

.NET 4.6

All Revit API binaries are now built targeting .NET 4.5.2. However, Revit uses the runtime from .NET 4.6. At a minimum, add-ins will need to be set to target .NET 4.5.2, but .NET 4.6 will also work.

Visual C++ Redistributable for Visual Studio 2015

Revit is now built with and installs runtime libraries from the Visual C++ Redistributable for Visual Studio 2015. Specifically, version 14.0.23026.0, which corresponds to vanilla Visual Studio 2015. 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.

Automatic transaction mode obsolete

The transaction mode:

- TransactionMode.Automatic

is now obsolete and this capability will be removed in a future release. The Manual transaction mode should be used for command callbacks that make changes to the Revit model (using the Transaction APIs to start, commit and manage the needed transactions). The ReadOnly transaction mode can be used for commands that do not require to make any changes to the Revit model.

Code signing of Revit Addins

To improve the security of Revit and its addins, and help users to clearly understand the origin of 3rd party code running within the context of Revit to avoid malicious tampering, a new code signing mechanism has been introduced. All API developers should:

- Get their addins signed with the certificate mechanism provided by Microsoft before the addins are released.
- Get the certificates installed to the Trusted Publishers store of Windows.

If this is not done, one or more message dialogs will be shown during Revit startup:

- If an addin has been signed correctly, but the certificate is not installed in Trust Publisher, a dialog with detailed information of the certificate will be shown.

- If the signature of an addin is invalid, a dialog with an error message will be shown to let end users know this.
- If an addin is unsigned, a dialog with the addin's information will be shown.

In each case, the end user can choose whether they want to always trust the addin, load it once, or skip loading.

Please refer to https://msdn.microsoft.com/library/ms537361(v=vs.85).aspx for detailed introduction about the code signing from Microsoft.

Background processes can load DB applications

Revit uses background processes called RevitWorker to perform certain calculations and operations out-of-process. While the situations in which these RevitWorkers are used is currently limited, the RevitWorker process is capable of loading a DB application add-in (<AddIn Type="DBApplication">) and events set in that add-in may be triggered by activities in the RevitWorker.

By default, no add-ins are loaded into RevitWorker. To have an add-in loaded into RevitWorker, add the LoadInRevitWorker flag to the .addin file:

The new property:

- RevitAddInDBApplication.LoadInRevitWorker

indicates whether or not a RevitWorker process will load this add-in. The default is false.

Application API changes

The default constructor for

- Autodesk.Revit.ApplicationServices.Application

has been removed. Use of this constructor in unsupported situations could result in unexpected behavior. The correct way to obtain the handle to the Application is from the arguments passed to the associated Revit API callback being invoked, for example:

- IExternalCommand.Execute() passes ExternalCommandData which contains UIApplication. Application is accessible from UIApplication.Application
- IUpdater.Execute() passes UpdaterData which provides access to the Document. Application is accessible from Document.Application
- Events will often pass the Application or the associated Document as the sender in the event callback. The event arguments may also contain an associated Application or Document.

One location where a full Application is not available is in the callbacks to IExternalApplication (OnStartup() and OnShutdown()). ControlledApplication is supplied instead. This is because Revit is not ready to perform operations with Documents while in this initialization state. If you need to start working with Documents immediately after Revit completes startup, you can subscribe to the ApplicationInitialized event.

Family API changes

The following member has been deprecated and replaced:

Deprecated member	New/replacement member
FamilyPointLocation.Location	FamilyPointLocation.GetLocation()

The following member has been changed:

- SpatialElementFromToCalculationPoints.isAcceptableFromPosition()

Revit now transforms the input position into the SpatialElementFromToCalculationPoints objects's coordinate space before calculating the distance. Previously, the input position was used unchanged, which could lead to Revit incorrectly claiming the point was out of bounds.

View API changes

Category visibility API

The following members related to category visibility in a view have been deprecated and replaced or added:

Deprecated member	New/replacement member
View.GetVisibility(Category category)	View.GetCategoryHidden(ElementId categoryId)
View.SetVisibility(Category category, bool visible)	View.SetCategoryHidden(ElementId categoryId, bool hide)
N/A	View.CanCategoryBeHidden(ElementId categoryId)

Text API changes

Revit text handling has been reworked, addressing long-standing issues and introducing new editing capabilities. Most of the existing API for text access has been maintained, but a few changes have been made.

Text related to CustomExporter contexts

Deprecated member	New/replacement member
TextNode.TextSize	TextNode.FontHeight
TextNode.IsForRightToLeftReading	No replacement
TextNode.GetFormattedTextRuns()	TextNode.GetFormattedText()

Note that the class FormattedTextRun has been marked obsolete. It is replaced by the new class FormattedText.

Event API changes

The following member has been deprecated and replaced:

Deprecated member	New/replacement member
DocumentPrintingEventArgs.Settings	DocumentPrintingEventArgs.GetSettings()

Alignment API changes

The method:

- ItemFactoryBase.NewAlignment()

will now throw exceptions with more informative messages, such as "The two references are not geometrically aligned so the Alignment cannot be created."

Geometry API changes

NurbSpline creation changes

The methods used to create a NurbSpline curve have been replaced. In the replacement methods, the newly created curve may be returned as a NURBSpline or a simpler curve such as line or arc. This is consistent with Revit expectations that the simplest possible representation of curve should be used in Revit elements.

Deprecated member	New/replacement member
	NurbSpline.CreateCurve(int degree, IList<double> knots, IList<XYZ> controlPoints, IList<double> weights)
NurbSpline.Create(IList<XYZ> controlPoints, IList<double> weights, IList<double> knots, int degree, bool closed, bool rational)	Two arguments were removed: • Rationality is now determined by the weights array. • Closed splines are not properly supported in Revit tools hence there is no option to create them (curves will be marked open).
NurbSpline.Create(IList<XYZ> controlPoints, IList<double> weights, IList<double> knots, int degree, bool closed, bool rational)	NurbSpline.CreateCurve(int degree, IList<double> knots, IList<XYZ> controlPoints). This version can be used in the common case when rational = false and all the weights are 1.
NurbSpline.Create(IList<XYZ> controlPoints, IList<double> weights)	NurbSpline.CreateCurve(IList<XYZ> controlPoints, IList<double> weights)

TessellatedShapeBuilder changes

The method Build() has changed from the prior release as follows:

Deprecated member	New/replacement member
TessellatedShapeBuilder.Build(TessellatedShapeBuilderTarget, TessellatedShapeBuilderFallback, ElementId)	TessellatedShapeBuilder.Build()

The results of Build are now stored in the TessellatedShapeBuilder instead of returned from Build(). Access these results through the new method:

- TessellatedShapeBuilder.GetBuildResult()

The options for building are now set as options in the TessellatedShapeBuilder itself. Access these options through the new properties:

- TessellatedShapeBuilder.Target
- TessellatedShapeBuilder.Fallback
- TessellatedShapeBuilder.GraphicsStyleId

Structure API changes

FoundationWall API

The following members have been deprecated and replaced:

Deprecated member	New/replacement member
Document.NewFoundationWall()	WallFoundation.Create()
WallFoundation.GetFoundationType()	WallFoundation.GetTypeId()
WallFoundation.SetFoundationType()	WallFoundation.ChangeTypeId()

Rebar API changes

The following method has been deprecated and replaced:

Deprecated method	New/replacement method
ExporterIFCUtils.GetRebarGeometry(DBView* pView)	Rebar.GetFullGeometryForView(const DBView* pView)

The method generates full geometry for a Rebar for a specific view, before cutting is applied.

The following methods have been deprecated and replaced:

Deprecated method	New/replacement method
Rebar.GetCenterlineCurves(bool adjustForSelfIntersection, bool suppressHooks, bool suppressBendRadius)	Rebar.GetCenterlineCurves(bool adjustForSelfIntersection, bool suppressHooks, bool suppressBendRadius, MultiplanarOption multiplanarOption, int barPositionIndex)
Rebar.GetCenterlineCurves(bool adjustForSelfIntersection, bool suppressHooks, bool suppressBendRadius, MultiplanarOption multiplanarOption)	

FabricSheet API changes

The following property has been deprecated and replaced:

Deprecated property	New/replacement property
FabricSheet.BentFabricWiresOrientation	FabricSheet.BentFabricStraightWiresLocation

FabricSheet.PlaceInHost behavioral change

The behavior of the method:

- Autodesk.Revit.DB.Structure.FabricSheet.PlaceInHost()

has been updated. It now properly supports move of the FabricSheet to a different structure with a modified transformation.

LoadCase API changes

The following types and members have been deprecated and replaced/added:

Deprecated type/member	New/replacement type/member
LoadNatureCategory enum	LoadCaseCategory enum
LoadCase.Create(Document, string, ElementId, LoadNatureCategory)	LoadCase.Create(Document, string, ElementId, LoadCaseCategory)
N/A	LoadCase.Create(Document, string, ElementId, ElementId)
LoadCase.NatureCategory	LoadCase.SubcategoryId

MEP API changes

Duct API

The following duct creation methods have been deprecated and replaced by new methods. Many of the new methods offer additional parameters supporting the assignment of duct system type and reference level:

Deprecated member	New/replacement member
Document.NewDuct(XYZ, XYZ, DuctType)	Duct.Create(Document, ElementId ductSystemTypeId, ElementId ductTypeId, ElementId levelId, XYZ, XYZ)
Document.NewDuct(XYZ, Connector, DuctType)	Duct.Create(Document, ElementId ductTypeId, ElementId levelId, Connector, XYZ)
Document.NewDuct(Connector, Connector, DuctType)	Duct.Create(Document, ElementId ductTypeId, ElementId levelId, Connector, Connector)
DuctFittingAndAccessoryConnectorData.Coordination	DuctFittingAndAccessoryConnectorData.GetCoordination()

Pipe API

The following members have been deprecated and replaced:

Deprecated member	New/replacement member
PipeType.Class	PipeSegment.ScheduleTypeId
PipeFittingAndAccessoryConnectorData.Coordination	PipeFittingAndAccessoryConnectorData.GetCoordination()

The deprecated property returns one pipe schedule type element on the pipe type. In case that the pipe type contains multiple pipe segments and schedule types in its routing preference definition, only the first pipe schedule type is returned in the deprecated property. Instead, the correct usage is to use the new property Pipe.PipeSegment, which provides the correct pipe schedule type and other segment properties, just as Revit property palette shows.

Additionally, the following new methods are available related to PipeScheduleType:

- PipeScheduleType.Create(Document, String) - Creates a new pipe schedule type with the given name.
- PipeScheduleType.GetPipeScheduleId(Document, String) - Returns an existing pipe schedule type with the given name.

MEP System API

The following properties have been deprecated and replaced by methods:

Deprecated member	New/replacement member
MechanicalSystem.Flow	MechanicalSystem.GetFlow()
MechanicalSystem.StaticPressure	MechancalSystem.GetStaticPressure()
PipingSystem.FixtureUnits	PipingSystem.GetFixtureUnits()
PipingSystem.Flow	PipingSystem.GetFlow()
PipingSystem.StaticPressure	PipingSystem.GetStaticPressure()

A new method is also added:

- PipingSystem.GetVolume()

Internally, these MEP system values are now calculated asynchronously on a non-blocking evaluation framework. In order to handle asynchronous calculation results, the caller needs to define callback methods to react on background calculation results (e.g., to refresh the user interface). API developers cannot define callbacks but will still get the correct value. If no callback methods are defined (e.g., in third party applications), the calculation is automatically switched to synchronous calculation.

These values have been exposed via built-in parameters in the past. They are still supported. For example, PipingSystem.get_ParameterValue(BuiltInParameter.RBS_PIPE_FLOW_PARAM) will get the correct flow value synchronously, assuming no callback is detected. The caveat is that, due to the internal support of asynchronous calculation, these parameters no longer support dynamic model update.

EnergyDataSettings API changes

The following members have been deprecated and replaced:

Deprecated member	New/replacement member

EnergyDataSettings.MassZoneCoreOffset	EnergyDataSettings.CoreOffset
EnergyDataSettings.MassZoneDividePerimeter	EnergyDataSettings.DividePerimeter

Rendering API changes

The rendering API has undergone major changes with the switch from NVIDIA mental ray to RapidRT. Several functions and enum values have been removed, as their corresponding functionality no longer exists.

Removed functions

- RenderingQuality.Low
- RenderingImageExposureSettings.MidTones
- SkyBackgroundSettings.VisibilityDistance
- DaylightPortalNode
- IExportContext.OnDaylightPortal

New enum value

- BackgroundStyle.Transparent

RenderQualitySettings changes

RenderingQualitySettings has been completely overhauled. All of the properties and functions have been removed and new functions have been added to be compatible with the RapidRT interface.

New properties and functions:

- RenderingQualitySettings.IsValidRenderLevel
- RenderingQualitySettings.IsValidRenderTime
- RenderingQualitySettings.IsCustomQuality
- RenderingQualitySettings.LightAndMaterialAccuracyMode
- RenderingQualitySettings.RenderDuration
- RenderingQualitySettings.RenderLevel
- RenderingQualitySettings.RenderTime

Plane API changes

The constructors and methods used to construct a Plane have been obsoleted and replaced, for consistency with the other exposed subclasses of Surface.

Deprecated member	New/replacement member
Plane(XYZ, XYZ, XYZ) (constructor)	Plane.CreateByOriginAndBasis()
Plane(XYZ, XYZ) (constructor)	Plane.CreateByNormalAndOrigin()
Plane()	Plane.Create(Frame) or other dedicated creation function
Autodesk.Revit.Creation.Application.NewPlane(XYZ, XYZ, XYZ)	Plane.CreateByOriginAndBasis()

Autodesk.Revit.Creation.Application.NewPlane(XYZ, XYZ)	Plane.CreateByNormalAndOrigin()
Autodesk.Revit.Creation.Application.NewPlane(CurveArray)	CurveLoop.GetPlane()

The new method:

- Plane.CreateByThreePoints()

creates a Plane object passing through three points supplied as arguments.

DirectShape API changes

Functions that treat application id and application data id as mandatory GUIDs are being phased out.

This affects:

- DirectShape.SetGUIDs()
- DirectShape.CreateElement()
- DirectShape.CreateElementInstance()

The DirectShape.ApplicationId and DirectShape.ApplicationDataId properties should be used instead.

DirectShape.IsValidCategoryId() has been reimplemented. The new version lists categories approved for use with DirectShape. The old version listed all top-level built-in model categories.

Point Cloud API changes

PointCloudOverrides API

PointCloudOverrides now supports overrides for point cloud regions. Several functions related to point cloud overrides have been deprecated and replaced:

Deprecated member	New/replacement member
PointCloudOverrides.GetPointCloudOverrideSettings(ElementId)	PointCloudOverrides.GetPointCloudScanOverrideSettings(ElementId)
PointCloudOverrides.GetPointCloudOverrideSettings(ElementId, String scanTag, Document)	PointCloudOverrides.GetPointCloudScanOverrideSettings(ElementId, String scanTag, Document)
PointCloudOverrides.SetPointCloudOverrideSettings(ElementId, PointCloudOverrideSettings)	PointCloudOverrides.SetPointCloudScanOverrideSettings(ElementId, PointCloudOverrideSettings)
PointCloudOverrides.SetPointCloudOverrideSettings(ElementId, PointCloudOverrideSettings, String scanTag, Document)	PointCloudOverrides.SetPointCloudScanOverrideSettings(ElementId, PointCloudOverrideSettings, String scanTag, Document)

The following new functions support region overrides:

- PointCloudOverrides.GetPointCloudRegionOverrideSettings(ElementId)
- PointCloudOverrides.GetPointCloudRegionOverrideSettings(ElementId, String regionTag, Document)

- PointCloudOverrides.SetPointCloudRegionOverrideSettings(ElementId, PointCloudOverrideSettings)
- PointCloudOverrides.SetPointCloudRegionOverrideSettings(ElementId, PointCloudOverrideSettings, String regionTag, Document)

Schedule API changes

ScheduleField

The following member has been deprecated and replaced:

Deprecated member	New/replacement member
ScheduleField.HasTotals	ScheduleField.DisplayType

The new enum:

- ScheduleFieldDisplayType

allows the user to specify the display type for a field. In addition to a standard field, the user can choose to display a total, the maximum value, the minimum value, or both.

The new method:

- ScheduleField.CanDisplayMinMax()

indicates whether the field can display minimum and maximum values.

UI API change

Rectangle class

The Rectangle class has been moved from the namespace Autodesk.Revit.UI to the namespace Autodesk.Revit.DB, and is now found in RevitAPI.dll.

Obsolete API removal

The following API members and classes which had previously been marked Obsolete have been removed in this release. Consult the API documentation from prior releases for information on the replacements to use:

Methods

- Autodesk.Revit.Creation.Application.NewPoint(XYZ)
- Autodesk.Revit.Creation.Document.NewGrid(Arc)
- Autodesk.Revit.Creation.Document.NewGrid(Line)

- Autodesk.Revit.Creation.Document.NewGrids(CurveArray)
- Autodesk.Revit.Creation.ItemFactoryBase.NewLevel(Double)
- Autodesk.Revit.Creation.ItemFactoryBase.NewTextNote(View, XYZ, XYZ, XYZ, Double, TextAlignFlags, TextNoteLeaderTypes, TextNoteLeaderStyles, XYZ, XYZ, String)
- Autodesk.Revit.Creation.ItemFactoryBase.NewTextNote(View, XYZ, XYZ, XYZ, Double, TextAlignFlags, String)
- Autodesk.Revit.Creation.Document.NewPointLoad(Reference, XYZ, XYZ, Boolean, PointLoadType, SketchPlane)
- Autodesk.Revit.Creation.Document.NewPointLoad(XYZ, XYZ, XYZ, Boolean, PointLoadType, SketchPlane)
- Autodesk.Revit.Creation.Document.NewLineLoad(Reference, IList<XYZ>, IList<XYZ>, Boolean, Boolean, Boolean, LineLoadType, SketchPlane) Autodesk.Revit.Creation.Document.NewLineLoad(Element, IList<XYZ>, IList<XYZ>, Boolean, Boolean, Boolean, LineLoadType, SketchPlane)
- Autodesk.Revit.Creation.Document.NewLineLoad(IList<XYZ>, IList<XYZ>, IList<XYZ>, Boolean, Boolean, Boolean, LineLoadType, SketchPlane)
- Autodesk.Revit.Creation.Document.NewLineLoad(XYZ, XYZ, XYZ, XYZ, XYZ, XYZ, Boolean, Boolean, Boolean, LineLoadType, SketchPlane)
- Autodesk.Revit.Creation.Document.NewAreaLoad(Element, XYZ, Boolean, AreaLoadType) Autodesk.Revit.Creation.Document.NewAreaLoad(CurveArray, Int32[], Int32[], XYZ, XYZ, XYZ, Boolean, AreaLoadType)
- Autodesk.Revit.Creation.Document.NewAreaLoad(CurveArray, Int32[], Int32[], IList<XYZ>, Boolean, AreaLoadType) Autodesk.Revit.Creation.Document.NewAreaLoad(IList<XYZ>, XYZ, Boolean, AreaLoadType)
- Autodesk.Revit.Creation.Document.NewLoadNature(String)
- Autodesk.Revit.Creation.Document.NewLoadUsage(String)
- Autodesk.Revit.Creation.Document.NewLoadCase(String, LoadNature, Category)
- Autodesk.Revit.Creation.Document.NewLoadCombination(String, Int32, Int32, Double[], LoadCaseArray, LoadCombinationArray, LoadUsageArray)
- Autodesk.Revit.DB.ElementTransformUtils::MirrorElements(Document,ElementIdSet,Plane)
- Autodesk.Revit.DB.Grid.ExtendToAllLevels()
- Autodesk.Revit.DB.Family.HasStructuralSection()
- Autodesk.Revit.DB.FamilySymbol.HasStructuralSection()
- Autodesk.Revit.DB.TessellatedShapeBuilder.Build(TessellatedShapeBuilderTarget, TessellatedShapeBuilderFallback, ElementId)
- Autodesk.Revit.DB.ViewShapeBuilder.SetShape(DirectShape)
- Autodesk.Revit.DB.ViewShapeBuilder.SetShape(DirectShapeType)
- Autodesk.Revit.DB.ViewCropRegionShapeManager.SetCropRegionShape(CurveLoop)
- Autodesk.Revit.DB.ViewCropRegionShapeManager.SetCropRegionEmptyShape()
- Autodesk.Revit.DB.ViewCropRegionShapeManager.GetCropRegionShape()
- Autodesk.Revit.DB.Structure.ReinforcementSettings.DocumentContainsNoAreaOrPathReinforcement()
- Autodesk.Revit.DB.Structure.ReinforcementSettings.DocumentContainsNoRebar()

Properties

- Autodesk.Revit.ApplicationServices.Application.IsQuiescent
- Autodesk.Revit.ApplicationServices.ControlledApplication.IsQuiescent
- Autodesk.Revit.DB.AnnotationSymbol.Leaders
- Autodesk.Revit.DB.BoundarySegment.Document
- Autodesk.Revit.DB.BoundarySegment.Element
- Autodesk.Revit.DB.BoundarySegment.Curve
- Autodesk.Revit.DB.CustomExporter.IncludeFaces

- Autodesk.Revit.DB.Floor.StructuralUsage
- Autodesk.Revit.DB.Grid.GridType
- Autodesk.Revit.DB.Level.LevelType
- Autodesk.Revit.DB.Level.PlaneReference
- Autodesk.Revit.DB.PlanarFace.Normal
- Autodesk.Revit.DB.PlanarFace.Vector
- Autodesk.Revit.DB.ReferencePlane.Plane
- Autodesk.Revit.DB.ReferencePlane.Reference
- Autodesk.Revit.DB.RevisionSettings.RevisionAlphabet
- Autodesk.Revit.DB.TextElement.Align
- Autodesk.Revit.DB.TextElement.LineWidth
- Autodesk.Revit.DB.TextNote.Leaders
- Autodesk.Revit.DB.ViewCropRegionShapeManager.Valid
- Autodesk.Revit.DB.Structure.AreaLoad.Force
- Autodesk.Revit.DB.Structure.AreaLoad.Force1
- Autodesk.Revit.DB.Structure.AreaLoad.Force2
- Autodesk.Revit.DB.Structure.AreaLoad.Force3
- Autodesk.Revit.DB.Structure.AreaLoad.RefPoint
- Autodesk.Revit.DB.Structure.AreaLoad.NumCurves
- Autodesk.Revit.DB.Structure.AreaLoad.NumLoops
- Autodesk.Revit.DB.Structure.AreaLoad.Curve
- Autodesk.Revit.DB.Structure.LineLoad.Point
- Autodesk.Revit.DB.Structure.LineLoad.Force
- Autodesk.Revit.DB.Structure.LineLoad.Force1
- Autodesk.Revit.DB.Structure.LineLoad.Force2
- Autodesk.Revit.DB.Structure.LineLoad.UniformLoad
- Autodesk.Revit.DB.Structure.LineLoad.ProjectedLoad
- Autodesk.Revit.DB.Structure.LineLoad.Moment
- Autodesk.Revit.DB.Structure.LineLoad.Moment1
- Autodesk.Revit.DB.Structure.LineLoad.Moment2
- Autodesk.Revit.DB.Structure.PointLoad.Force
- Autodesk.Revit.DB.Structure.PointLoad.Moment
- Autodesk.Revit.DB.Structure.LoadCombination.CombinationTypeIndex
- Autodesk.Revit.DB.Structure.LoadCombination.CombinationStateIndex
- Autodesk.Revit.DB.Structure.LoadCombination..CombinationType
- Autodesk.Revit.DB.Structure.LoadCombination..CombinationState
- Autodesk.Revit.DB.Structure.LoadCombination..NumberOfComponents
- Autodesk.Revit.DB.Structure.LoadCombination..Factor
- Autodesk.Revit.DB.Structure.LoadCombination..CombinationCaseName
- Autodesk.Revit.DB.Structure.LoadCombination..CombinationNatureName
- Autodesk.Revit.DB.Structure.LoadCombination..NumberOfUsages
- Autodesk.Revit.DB.Structure.LoadCombination..UsageName
- Autodesk.Revit.DB.Structure.BoundaryConditions.NumCurves
- Autodesk.Revit.DB.Structure.BoundaryConditions.Curve
- Autodesk.Revit.DB.Structure.BoundaryConditions.AssociatedLoad

Enumerated types

- Autodesk.Revit.DB.SlabFoundationType

API additions

Application API additions

Document creation

The new method:

- Application.NewProjectDocument(UnitSystem)

creates a new imperial or metric project document without a specified template.

Family API additions

New PromptForFamilyInstancePlacement() options

The new options:

- PromptForFamilyInstancePlacementOptions.SketchGalleryOptions
- PromptForFamilyInstancePlacementOptions.FaceBasedPlacementType

allow more programmatic control over the options available to govern the placement of a given family. The FaceBasedPlacementType option exposes the available options for placement of a face-based family, while the SketchGalleryOptions offer control over the types of sketched curves which can be used for a curve-based family.

Family and FamilyInstance members

The new property:

- Family.IsParametric

identifies whether the family contains parametric relations between some of its elements.

The new method:

- Family.HasLargeSketches()

determines whether the family contains sketches with a large number of elements.

The new members:

- FamilyInstance.CanSplit
- FamilyInstance.Split() - This method allows to split the family instance element.

provide access to the ability to split a curve-driven family instance (such as beam, column, or brace).

View API additions

TemporaryViewModes

The new class:

- TemporaryViewModes

carries data related to the state and properties of available temporary view modes. Access to an instance of this class is via the property:

- View.TemporaryViewModes

The class has the following methods and properties:

- TemporaryViewModes.DeactivateAllModes() - Deactivates all temporary modes that are currently active.
- TemporaryViewModes.DeactivateMode() - Deactivates the given temporary mode.
- TemporaryViewModes.GetCaption() - A text caption to use for the given mode.
- TemporaryViewModes.IsModeActive() - Tests whether a given mode is currently active or not.
- TemporaryViewModes.IsModeAvailable() - Tests whether a temporary view mode is currently available in the associated view.
- TemporaryViewModes.IsModeEnabled() - Tests whether a temporary view mode is currently enabled in the associated view.
- TemporaryViewModes.IsValidState() - Tests whether the given state is valid for the associated view and the context the view is currently in.
- TemporaryViewModes.PreviewFamilyVisibility - The current state of the PreviewFamilyVisibility mode in the associated view.
- TemporaryViewModes.RevealConstraints - The current state of the RevealConstraints mode in the associated view.
- TemporaryViewModes.RevealHiddenElements - The current state of the RevealHiddenElements mode in the associated view.
- TemporaryViewModes.WorskaringDisplay - The current state of the WorksharingDisplay mode in the associated view.

Convert dependent view to independent

The new function:

- View.ConvertToIndependent()

converts a dependent view to be independent.

Plan view underlay

The new methods:

- ViewPlan.GetUnderlayBaseLevel()
- ViewPlan.GetUnderlayTopLevel()
- ViewPlan.SetUnderlayBaseLevel()
- ViewPlan.SetUnderlayRange()
- ViewPlan.SetUnderlayOrientation()
- ViewPlan.SetUnderlayOrientation()

provide access to the underlay levels and settings for plan views.

Assembly views creation

The creation of assembly views and schedules has been improved to allow the creation of an assembly view or schedule with template information. The new overloads for methods:

- AssemblyViewUtils.Create3DOrthographic()
- AssemblyViewUtils.CreateDetailSection()
- AssemblyViewUtils.CreateSingleCategorySchedule()
- AssemblyViewUtils.CreatePartList()
- AssemblyViewUtils.CreateMaterialTakeoff()

offer two new arguments:

- viewTemplateId - the id of the template from which the view is to be created.
- isAssigned - if true, the template passed in viewTemplateId will be assigned to the view; if false, the template information is only applied to the view.

Depth Cueing

The new class:

- ViewDisplayDepthCueing

allows users to control the display of distant objects in section and elevation views. When depth cueing is active, objects blend into the background color (fade) with increasing distance from the viewer.

The class contains the following methods and properties:

- ViewDisplayDepthCueing.EnableDepthCueing
- ViewDisplayDepthCueing.StartPercentage - Indicates where depth cueing begins. A value of 0 indicates that depth cueing begins at the front clip plane of the view.
- ViewDisplayDepthCueing.EndPercentage - Indicates where depth cueing ends. Objects further than the end plane will fade the same amount as objects at the end plane. A value of 100 indicates the far clip plane.
- ViewDisplayDepthCueing.FadeTo - Indicates the maximum amount to fade objects via depth cueing. A value of 100 indicates complete invisibility.
- ViewDisplayDepthCueing.SetStartEndPercentages()

The new methods:

- DBView.GetDepthCueing()
- DBView.SetDepthCueing()

allow the user to get and set the depth cueing settings for the view.

Text API additions

Revit text handling has been reworked, addressing long-standing issues and introducing new editing capabilities.

Text range

The new class:

- Autodesk.Revit.DB.TextRange

identifies a range of text in a FormattedText via its start index, end index and/or length.

Formatted text

The new methods:

- TextNote.GetFormattedText()
- TextNote.SetFormattedText()

access a FormattedText object which contains the text and its associated formatting. FormattedText allows read and write access to the text and the formatting in the text note.

The text in a FormattedText object is accessible at a whole or at the level of a specific TextRange (overloads exist for both levels of access):

- FormattedText.GetPlainText()
- FormattedText.SetPlainText()

Most formatting can be accessed at the level of the entire text, or at the level of a specific TextRange (overloads exist for both levels of access):

- FormattedText.GetAllCapsStatus()
- FormattedText.SetAllCapsStatus()
- FormattedText.GetBoldStatus()
- FormattedText.SetBoldStatus()
- FormattedText.GetItalicStatus()
- FormattedText.SetItalicStatus()
- FormattedText.GetUnderlineStatus()
- FormattedText.SetUnderlineStatus()
- FormattedText.GetSuperscriptStatus()
- FormattedText.SetSuperscriptStatus()
- FormattedText.GetSubscriptStatus()
- FormattedText.SetSubscriptStatus()
- FormattedText.GetListType()
- FormattedText.SetListType()

When checking the status of a particular range, it is possible that the formatting is applied to all text in the range, or no text in the range, or a mixture. This is reflected in the enumerated type FormatStatus.

The method:

- FormattedText.AsTextRange()

returns a TextRange identifying the entire formatted text.

The method:

- FormattedText.Find()

returns a TextRange identifying the first occurrence of the given string within the text, from a given index.

TextElement additions

The text element now has size restrictions. The new functions:

- TextElement.GetMinimumAllowedWidth()
- TextElement.GetMaximumAllowedWidth()

return the minimum and maximum permitted width for an existing, or newly created, TextElement.

Text editor options

The new class:

- Autodesk.Revit.UI.TextEditorOptions

provides access to settings that control Revit's Text Editor appearance and functionality.

Geometry API additions

ShapeImporter class

The new utility class:

- ShapeImporter

supports conversion of geometry stored in external formats (such as SAT and Rhino) into a collection of Revit geometry objects. Use ShapeImporter.Convert() to generate the geometry objects (and where possible, corresponding materials and graphics styles in the associated document).

Builder for 3D boundary representations

The new builder class:

- BRepBuilder

offers the ability to construct Revit boundary representation geometry (either solids or "open sheets") as a result of inputs of surface, edges, and boundary loops of edges. If the construction of the boundary representation is successful, the resulting geometry objects can be used directly in any other Revit tool that accepts geometry, or the BRepBuilder can directly be passed to populate a DirectShape via:

- DirectShape.SetShape(ShapeBuilder)
- DirectShape.AppendShape(ShapeBuilder)

New Surface subclasses

Several new subclasses of Surface have been introduced:

- CylindricalSurface
- ConicalSurface
- RuledSurface
- RevolvedSurface
- HermiteSurface

These subclasses expose creation methods and read-only properties suitable for use in constructing import geometry.

Frame

New method added

- CanDefineRevitGeometry() - Tests whether the supplied Frame object may be used to define a Revit curve or surface. In order to satisfy the requirements the Frame must be orthonormal and its origin is expected to lie within the Revit design limits.

XYZ

New method added

- IsWithinLengthLimits() - Validates that the input point is within Revit design limits.

Fixed Reference Sweeps

The new static method:

- GeometryCreationUtilities.CreateFixedReferenceSweptGeometry()

allows creation of a solid using the "fixed reference sweep" method, similar to the method defined in the STEP ISO 10303-42 standard.

A typical use of this method is to create a swept solid for which a line in the cross-section of the solid remains horizontal all along the sweep. As an example, this can be used to construct railings to ensure that the top of the railing remains oriented to the horizontal steps of the stairs. In this example, the fixed reference direction would be chosen to be the upward vertical direction. See the function's description for further details.

As with other GeometryCreationUtilities methods, there is a second version of CreateFixedReferenceSweptGeometry that takes a SolidOptions input, allowing the user to assign a material or graphics style to the solid.

Parameter API additions

Global Parameters

Global Parameters support controlling geometry constraints through special parameters defined in a project document. Global Parameters can be used for both labeling and reporting to/from dimensions, as well as setting values of instance parameters.

The new class

- GlobalParametersManager

provides the main access point to managing global parameters in project document. It offers the following members:

- AreGlobalParametersAllowed() - tests whether global parameters are allowed in a document
- GetAllGlobalParameters() - returns all global parameters in a document
- FindByName() - find a global parameter by its name
- IsUniqueName() - test uniqueness of the name of a prospective global parameters
- IsValidGlobalParameter() - test if an Id is of a valid global parameter element

The new class:

- GlobalParameter

contains methods to control and manipulate a single global parameter. It's most important members include:

- [static] Create() - Creates a new Global Parameter in the given document.
- GetAffectedElements() - Returns all elements of which properties are driven by this global parameter.
- GetAffectedGlobalParameters() - Returns all other global parameters which refer to this global parameter in their formulas.
- GetLabeledDimensions() - Returns all dimension elements that are currently labeled by this global parameter.
- CanLabelDimension() - Tests whether a dimension can be labeled by the global parameter.
- LabelDimension() - Labels a dimension with this global parameter.
- UnlabelDimension() - Un-labels a dimension that is currently labeled by this global parameter.
- GetLabelName() - Returns the name of this parameter's label, which is used to label dimension elements.
- SetDrivingDimension() - Set a dimension to drive the value of this parameter.
- IsValidFormula() - Tests that the given expression is a valid as formula for this parameter.
- GetFormula() - Returns the parameter's expression in form of a string.
- SetFormula() - Sets a formula expression for this parameter.
- GetValue() - Obtains the current value of the global parameter.
- SetValue() - Sets a new value of the global parameter.
- HasValidTypeForReporting() - Tests that the global parameter has data of a type that supports reporting.
- [static] IsValidDataType() - Tests whether the input Data Type is valid as a type of a global parameter.

- IsDrivenByDimension - Indicates whether this parameter is driven by a dimension or not.
- IsDrivenByFormula - Indicates whether this parameter is driven by a formula or not.
- IsReporting - Indicates whether this is a reporting global parameter or not.

The new class:

- ParameterValue

contains a value of a corresponding global parameter. It is a base class for derived concrete classes, one per each type of a parameter value:

- IntegerParameterValue
- DoubleParameterValue
- StringParameterValue
- ElementIdParameterValue
- NullParameterValue

All the derived classes have only one property:

- Value - gets or sets the value as the corresponding type.

New methods added the the Parameter class:

- CanBeAssociatedWithGlobalParameter() - Tests whether a parameter can be associated with the given global parameter.
- CanBeAssociatedWithGlobalParameters() - Tests whether a parameter can be associated with any global parameter.
- AssociateWithGlobalParameter() - Associates a parameter with a global parameter in the same document.
- DissociateFromGlobalParameter() - Dissociates a parameter from a global parameter.
- GetAssociatedGlobalParameter() - Returns a global parameter, if any, currently associated with a parameter.

InternalDefinition.Id

The new property:

- InternalDefinition.Id

returns the id for the associated parameter. This is the id of the associated ParameterElement if the parameter is not built-in.

Multiline Text parameter support

The new enumerated value:

- ParameterType.MultilineText

was added for creation and use of multi-line text parameters.

CurveElement API additions

End joins and tangent constraints APIs

CurveElements now support options to read elements that are joined to this element at the given end point, and to apply and lock tangent constraints.

The new methods:

- CurveElement.GetAdjoinedCurveElements()
- CurveElement.IsAdjoinedCurveElement()

support read of elements joined to this curve element.

The new methods:

- CurveElement.SupportsTangentLocks()
- CurveElement.HasTangentJoin()
- CurveElement.HasTangentLocks()
- CurveElement.GetTangentLock()
- CurveElement.SetTangentLock()

support access and modification to tangent constraints on the given curve element.

Railing API additions

Several new methods have been added to BaseRailing:

- BaseRailing.Create() - Creates a new railing by specifying the railing path in the project document.
- BaseRailing.SetPath() - Sets the railing path.
- BaseRailing.RailingCanBeHostedByElement() - Checks whether the specified element can be used as a host for the railing.

Schedule API additions

Combined Parameters

Several methods and properties have been added to support combined parameters in schedules:

- ScheduleDefinition.InsertCombinedParameterField()
- ScheduleDefinition.IsValidCombinedParameters() - Verifies if the input is suitable for a combined parameter field.
- ScheduleField.GetCombinedParameters() - The values from a combined parameter field.
- ScheduleField.SetCombinedParameters()
- ScheduleField.IsCombinedParameterField
- ScheduleField.IsValidCombinedParameters()
- static TableCellCombinedParameterData.Create()

The new enum value:

- ScheduleFieldType.CombinedParameter

indicates a combined parameter field.

Tag API additions

SpatialElementTag API

SpatialElementTag is a base element for Autodesk.Revit.DB.Architecture.RoomTag, Autodesk.Revit.DB.AreaTag and Autodesk.Revit.DB.Mechanical.SpaceTag.

The following new properties have been added:

- SpatialElementTag.IsOrphaned - Identifies if the tag is orphaned or not.
- SpatialElementTag.IsTaggingLink - Identifies if the tag has reference to an object in a linked document or not.
- SpatialElementTag.LeaderElbow - The position of the leader's elbow (middle point).
- SpatialElementTag.LeaderEnd - The position of the leader's end.
- SpatialElementTag.TagHeadPosition - The position of the tag's head.
- SpatialElementTag.TagOrientation - The orientation of the tag.

RoomTag API

The following new properties have been added to RoomTag:

- RoomTag.IsInRoom - Identifies if the tag is located in a room.
- RoomTag.TaggedLocalRoomId - The ElementId of the tagged room in the same document.
- RoomTag.TaggedRoomId - The LinkElementId of the tagged room. This property works for both rooms in the main model and rooms in linked models.

UI API additions

ColorSelectionDialog

The new class:

- ColorSelectionDialog

provides the option to launch the Revit Color dialog to prompt the user to select a color. The original color can be set as well as the selected color before the user changes it. The method:

- ColorSelectionDialog.Show()

returns a status indicating if a color was selected and the dialog confirmed, or if the user canceled the selection.

FileOpenDialog and FileSaveDialog

The new classes:

- FileOpenDialog
- FileSaveDialog

allow an add-in to prompt the user with the Revit dialog used to navigate to and select an existing file path. FileOpenDialog is typically used to select a file for opening or importing. FileSaveDialog is typically used to enter a file name for saving or exporting.

The behavior and appearance of this dialog matches the Revit "Open" dialog. This is a general-purpose dialog for opening any given file type, and options to configure settings like worksharing options will not be included. Use of this dialog does not actually open an existing file, but it will provide the selected file path back to the caller to take any action necessary.

These dialogs inherit from:

- FileDialog

which exposes the shared options and operations needed for prompting with either an open or a save dialog. The method:

- FileDialog.Show()

returns a status indicating if a file was selected and the dialog confirmed, or if the user canceled the selection.

TaskDialog API additions

The new members:

- TaskDialog.ExtraCheckBoxText
- TaskDialog.WasExtraCheckBoxChecked()

provide access to an extra checkbox shown the user in the TaskDialog. If it is set, a checkbox with the text will be shown in the task dialog. The caller can get the user setting for the checkbox by checking the return value of the WasExtraCheckBoxChecked() method

Support for journal data in overridden commands

The new members:

- BeforeExecutingEventArgs.UsingCommandData
- ExecutedEventArgs.GetJournalData()
- ExecutedEventArgs.SetJournalData()

support the ability for the add-in to store journal data associated to an overridden command, similar to the capability offered for External Commands.

Dockable Pane API additions

The new members:

- DockablePaneProviderData.VisibleByDefault

support the ability for the add-in to control the whether or not any Dockable Panes they register should be visible by default or not. Default is to true.

Structure API additions

FabricSheet

The new property:

- FabricSheet.FabricNumber

returns the reinforcement numbering value for the fabric sheet element.

The new method:

- FabricSheet.GetSegmentParameterIdsAndLengths()

returns the set of parameter ID and length pairs that correspond to segments of a bent fabric sheet (like A, B, C, D etc.).

The new method:

- FabricSheet.SetSegmentLength()

sets the length of the bent fabric sheet segment (like A, B, C, D etc.)

Quantitative FabricSheet layout

The new enum value:

- FabricSheetLayoutPattern.QuantitativeSpacing

indicates a pattern containing multiple groups of wires with a specific spacing and diameter.

Several new methods have been added to support this layout pattern:

- FabricSheetType.SetLayoutAsCustomPattern() - Sets the major and minor layout patterns to Custom, and specifies the FabricWireItems and overhang distances to be used.
- FabricSheetType.IsCustom() - Determines whether the type is Custom Fabric Sheet.
- FabricSheettype.GetWireItem() - Gets the wire stored in the FabricSheet at the associated index.

The new class:

- Autodesk.Revit.DB.Structure.FabricWireItem

represents a single fabric wire.
It has the following methods and properties:

- FabricWireItem.Create()
- FabricWireItem.Distance - The distance to the next FabricWireItem.
- FabricWireItem.WireLength
- FabricWireItem.WireType

LoadCase API additions

The property:

- LoadCase.Number

can now be set.

The new method:

- LoadCase.isNumberUnique()

allows users to check if the proposed number is unique.

RebarContainer

The new members:

- RebarContainer.SetItemHiddenStatus()
- RebarContainer.IsItemHidden()

provide access to the option to hide an individual RebarContainerItem in the given view.

The new property:

- RebarContainer.PresentItemsAsSubelements

identifies if Items should be presented in schedules and tags as separate subelements.

Structural Connection API additions

Structural Connection API

The new class:

- Autodesk.Revit.DB.Structure.StructuralConnectionHandler

represents connections between structural elements. A StructuralConnectionHandler can connect structural walls, floors, foundations, framings, or columns.

Some methods and properties include:

- StructuralConnectionHandler.Create() - Creates a new instance of a StructuralConnectionHandler, which defines the connection between the given elements. The first element given is set as the primary one.
- StructuralConnectionHandler.GetConnectedElementIds()
- StructuralConnectionHandler.IsDetailed() - Checks if the StructuralConnectionHandler has the detailed connection type.
- StructuralConnectionHandler.ApprovalTypeId
- StructuralConnectionHandler.SingleElementEndIndex - The element end index for single element connections. 0 indicates the start and 1 the end.

The new class:

- Autodesk.Revit.DB.Structure.StructuralConnectionHandlerType

defines the type of a StructuralConnectionHandler.

The new class:

- Autodesk.Revit.DB.Structure.StructuralConnectionApprovalType

defines a type element which represents a connection approval type.

The new class:

- Autodesk.Revit.DB.Structure.StructuralConnectionSettings

provides access to project-wide structural connection settings. It contains the following methods and properties:

- static StructuralConnectionSettings.GetStructuralConnectionSettings()
- StructuralConnectionSettings.IncludeWarningControls - If set to true, a yellow triangle will be displayed with StructuralConnectionElements which have associated warnings.

Rebar Couplers

The new class:

- Autodesk.Revit.DB.Structure.RebarCoupler

represents a rebar coupler element.

RebarCoupler has the following methods and properties:

- static RebarCoupler.Create()
- RebarCoupler.CouplerLinkTwoBars() - Determines whether the coupler sits on two rebar or caps a single rebar.
- RebarCoupler.GetCoupledReinforcementData() - If the coupler connects two rebars, this method returns a list of two ReinforcementData. If it only connects one, there will be one ReinforcementData.
- RebarCoupler.GetPointsForPlacement() - Gets the point or points where the coupler is placed.

- RebarCoupler.GetCouplerPositionTransform() - Gets a transform representing the relative position of the coupler at index couplerPositionIndex in the set.
- RebarCoupler.GetCouplerQuantity() - Identifies the number of couplers in a set.
- RebarCoupler.CouplerMark

The new method:

- Rebar.GetCouplerId()

returns the id of the RebarCoupler applied to the rebar at the specified end.

The new enum value:

- Autodesk.Revit.DB.Structure.StructuralNumberingSchemas.RebarCoupler

indicates the built-in schema for numbering rebar coupler elements.

The new enum:

- Autodesk.Revit.DB.Structure.RevitCouplerError

contains various error states for a rebar coupler.

The new class:

- Autodesk.Revit.DB.Structure.RebarReinforcementData

contains information about rebar couplers.

RebarReinforcementData has the following methods:

- RebarReinforcementData.Create()
- RebarReinforcementData.GetId() - The id of the associated Rebar.
- RebarReinforcementData.SetId()
- RebarReinforcementData.GetEnd() - The end of the rebar to which a coupler is attached.

Rebar end treatments

The new class:

- Autodesk.Revit.DB.Structure.EndTreatmentType

represents an end treatment type for rebar.

It has the following methods:

- static EndTreatmentType.Create(Document)
- static EndTreatmentType.Create(Document, String) - Creates a new EndTreatmentType with the given name.
- static EndTreatmentType.CreateDefaultEndTreatmentType() - Creates a new EndTreatmentType with a default name.

- EndTreatmentType.GetEndTreatment() - The value of the END_TREATMENT parameter.
- EndTreatmentType.SetEndTreatment()

Several methods have been added to other classes to support end treatments:

- RebarShape.GetEndTreatmentTypeId() - The id of the end treatment type for the designated shape end.
- RebarShape.SetEndTreatmentTypeId()
- RebarShape.HasEndTreatment()
- Rebar.GetEndTreatmentTypeId()

The new enum value:

- ElementGroupType.EndTreatmentType

indicates an end treatment type.

The new property:

- ReinforcementSettings.RebarShapeDefinesEndTreatments

indicates whether end treatments are defined by the RebarShape of the Rebar element. This value can be changed if the document contains no rebars, area reinforcements, or path reinforcements.

Additionally, the following methods and properties have been modified to require that the RebarShape has no end treatments:

- RebarContainer.AppendItemFromRebar()
- RebarContainer.AppendItemFromRebarShape()
- RebarContainer.AppendItemFromCurvesAndShape()
- RebarContainerItem.SetFromRebar()
- RebarContainerItem.SetFromRebarShape()
- RebarContainerItem.SetFromCurvesAndShape()
- RebarContainerItem.RebarShapeId

Other Structure API additions

New properties have been added to ReinforcementSettings:

- ReinforcementSettings.NumberVaryingLengthRebarsIndividually - Modifies the way varying length bars are numbered (individually or as a whole).
- ReinforcementSettings.RebarVaryingLengthNumberSuffix - A unique identifier used for a bar within a variable length rebar set.
- RebarConstraintsManager.isRebarConstrainedPlacementEnabled - Enables/Disables the 'Rebar Constrained Placement' setting in the current Revit Application Session.

One property has been added to Rebar:

- Rebar.DistributionType - Modifies the type of a rebar set. Rebar sets can be Uniform or VaryingLength.

The new Rebar method:

- getParameterValueAtIndex(ElementId paramId, int barPositionIndex)

returns the ParameterValue at the given bar index inside a rebar set.

Fabrication API additions

FabricationPart - product list support

To specify a size, some FabricationPart elements, such as purchased duct and pipe fittings, have a Product Entry field in the Properties palette. In the API these FabricationPart elements are identified as having a "product list". The product list entries represent a catalog of available sizes for the selected part.

The following new members are added to support product list FabricationPart elements:

- FabricationPart.ProductListEntry - The product list entry index of the fabrication part. A value of -1 indicates the fabrication part is not a product list.
- FabricationPart.IsProductList()
- FabricationPart.GetProductListEntryName()
- FabricationPart.GetProductListEntryCount()
- FabricationPart.IsProductListEntryCompatibleSize() - Checks to see if this part can be changed to the specified product list entry without altering any connected dimensions.

Design to fabrication conversion

The new class:

- DesignToFabricationConverter

supports the conversion of design elements to FabricationPart elements. Use the method:

- DesignToFabricationConverter.Convert()

to carry out the conversion, and use the available accessor methods to get the elements created during the conversion, and the elements which failed to convert for various reasons.

FabricationPart - Stretch and fit

The new method:

- FabricationPart.StretchAndFit()

supports the operation to stretch the fabrication part from the specified connector and fit to the target routing end. The routing end is indicated as a FabricationPartRouteEnd object, which can be obtained from:

- FabricationPartRouteEnd.CreateFromConnector()
- FabricationPartRouteEnd.CreateFromCenterline()

Support for fabrication configuration profiles

The new method:

- FabricationConfiguration.SetConfiguration(Autodesk.Revit.DB.FabricationConfigurationInfo info, System.String profile)

sets the fabrication configuration with the specified profile.

The new methods:

- FabricationConfiguration.GetProfile()
- FabricationConfiguration.GetProfiles()

return the profile associated with the loaded configuration or all configurations, respectively.

Support for accessing fabrication configuration data abbreviations added

The new methods:

- FabricationConfiguration.GetMaterialAbbreviation()
- FabricationConfiguration.GetSpecificationAbbreviation()
- FabricationConfiguration.GetInsulationSpecificationAbbreviation()

MEP Fabrication API

The following new members and properties have been added to the MEP fabrication and FabricationPart capabilities:

The new properties:

- FabricationPart.ServiceId
- FabricationPart.ServiceName
- FabricationPart.ServiceAbbreviation
- FabricationPart.TopOfPartElevation
- FabricationPart.BottomOfPartElevation
- FabricationPart.ItemNumber
- FabricationPart.ItemCustomId
- FabricationPart.Size
- FabricationPart.Slope
- FabricationPart.OverallSize
- FabricationPart.Weight
- FabricationPart.HasInsulation
- FabricationPart.InsulationType
- FabricationPart.InsulationThickness
- FabricationPart.InsulationArea
- FabricationPart.HasLining
- FabricationPart.LiningType
- FabricationPart.LiningThickness
- FabricationPart.LiningArea

- FabricationPart.SheetMetalArea
- FabricationPart.MaterialThickness
- FabricationPart.HasDoubleWall
- FabricationPart.DoubleWallMaterial
- FabricationPart.DoubleWallMaterialThickness
- FabricationPart.DoubleWallMaterialArea
- FabricationPart.IsBoughtOut
- FabricationPart.Notes
- FabricationPart.Alias
- FabricationPart.Vendor
- FabricationPart.VendorCode
- FabricationPart.ProductName
- FabricationPart.ProductShortDescription
- FabricationPart.ProductLongDescription
- FabricationPart.ProductFinishDescription
- FabricationPart.ProductSpecificationDescription
- FabricationPart.ProductMaterialDescription
- FabricationPart.ProductSizeDescription
- FabricationPart.ProductDataRange
- FabricationPart.ProductOriginalEquipmentManufacture
- FabricationPart.ProductInstallType

The new methods:

- FabricationPart.RotateConnecteFabricationPart.ServiceIddPartByConnector()
- FabricationPart.RotateConnectedTap()
- FabricationPart.StretchAndFit()
- FabricationServiceButton.ContainsFabricationPartType()
- FabricationServiceButton.GetConditionDescription()
- FabricationPart.CreateHanger() - Creates a free placed hanger.
- FabricationPart.Reposition() - Repositions the fabrication straight part to another end of the run.
- FabricationPart.PlaceFittingAsCutIn() - Places the fitting on the straight part by cut in, use the fitting's focal point as the insertion position.
- FabricationPart.CanAdjustEndLength() - Checks if the end of fabrication part can be adjusted.
- FabricationPart.AdjustEndLength() - Adjusts the length for the specified connector of the fabrication part.
- FabricationHostedInfo.GetBearerCenterline() - Gets the centerline of the bearer. This method is applicable only for bearer hangers.
- FabricationRodInfo.SetRodEndPosition() - Sets the position of the rod end.
- FabricationRodInfo.AttachToHanger() - Attaches the hanger rod to another bearer hanger.
- FabricationRodInfo.IsRodLockedWithHost() - Checks if the rod is locked with the host.
- FabricationRodInfo.SetRodLockedWithHost() - Locks the rod with the host.
- FabricationRodInfo.GetBearerExtension() - Gets the bearer extension.
- FabricationRodInfo.SetBearerExtension() - Sets the bearer extension.

Electrical API additions

The new method:

- Wire.GetMEPSystems()

gets the system(s) to which the wire belongs.

The new property:

- ElectricalSetting.CircuitRating

provides access to the default circuit rating for a newly created circuit.

The new property:

- ElectricalSetting.CircuitLoadCalculationMethod

provides read and write access to method Revit will use to calculate the apparent circuit load.

Cable Tray API

The new class:

- Autodesk.Revit.DB.Electrical.CableTraySettings

contains settings for cable trays.

It has the following methods and properties:

- CableTraySettings.GetCableTraySettings()
- CableTraySettings.ConnectorSeparator - The cable tray connector separator string.
- CableTraySettings.FittingAnnotationSize
- CableTraySettings.RiseDropAnnotationSize
- CableTraySettings.SizeSeparator
- CableTraySettings.SizeSuffix
- CableTraySettings.UseAnnotationScaleForSingleLineFittings

The new class:

- Autodesk.Revit.DB.Electrical.CableTraySizes

contains info about the cable tray sizing options in the project.

Some methods include:

- static CableTraySizes.GetCableTraySizes()
- CableTraySizes.AddSize() - Inserts a new MEPSize into the cable tray sizes. For cable trays, the nominal diameter MEPSize is used.

The new class:

- Autodesk.Revit.DB.Electrical.CableTraySizeIterator

allows iterating over the CableTraySizes.

Conduit API

The new class:

- Autodesk.Revit.DB.Electrical.ConduitSettings

contains settings for conduits.

It has the following methods and properties:

- ConduitSettings.GetConduitSettings()
- ConduitSettings.ConnectorSeparator - The conduit separator string.
- ConduitSettings.FittingAnnotationSize
- ConduitSettings.RiseDropAnnotationSize
- ConduitSettings.SizeSeparator
- ConduitSettings.SizeSuffix
- ConduitSettings.UseAnnotationScaleForSingleLineFittings

The new class:

- Autodesk.Revit.DB.Electrical.ConduitSize

contains basic size information for a conduit.

It contains the following methods and properties:

- ConduitSize.ConduitSize() - Constructs a ConduitSize object.
- ConduitSize.BendRadius - The minimum bend radius.
- ConduitSize.InnerDiameter
- ConduitSize.NominalDiameter
- ConduitSize.OuterDiameter
- ConduitSize.UsedInSizeLists
- ConduitSize.UsedInSizing

The new classes:

- Autodesk.Revit.DB.Electrical.ConduitSizes
- Autodesk.Revit.DB.Electrical.ConduitSizeIterator

allow traversal of the ConduitSize objects in the project.

The new class:

- Autodesk.Revit.DB.Electrical.ConduitSizeSettings

contains settings information about the ConduitSize objects in the project.

Some methods include:

- static ConduitSizeSettings.GetConduitSizeSettings()

- ConduitSizeSettings.AddSize() - Inserts a new ConduitSize into the conduit size settings.
- ConduitSizeSettings.CreateConduitStandardTypeFromExistingStandardType() - Creates one conduit standard type with the given name and assigns the conduit sizes from the existing standard type.

Other MEP API additions

DuctSettings Class

New properties:

- FlatOnTop - Gets/Sets the abbreviation of the Flat On Top (FOT) string
- FlatOnBottom - Gets/Sets the abbreviation of the Flat On Bottom (FOB) string
- SetUp - Gets/Sets the abbreviation of the Set Up (SP) string
- SetDown - Gets/Sets the abbreviation of the Set Down (SD) string
- Centerline - Gets/Sets the abbreviation of the Centerline (=) string

PipeSettings Class

New properties:

- FlatOnTop - Gets/Sets the abbreviation of the Flat On Top (FOT) string
- FlatOnBottom - Gets/Sets the abbreviation of the Flat On Bottom (FOB) string
- SetUp - Gets/Sets the abbreviation of the Set Up (SP) string
- SetDown - Gets/Sets the abbreviation of the Set Down (SD) string
- Centerline - Gets/Sets the abbreviation of the Centerline (=) string

MechanicalUtils

The new method:

- MechanicalUtils.BreakCurve()

breaks the duct or duct placeholder into two parts at the given position.

PlumbingUtils

The new method:

- PlumbingUtils.BreakCurve()

breaks the pipe or pipe placeholder into two parts at the given position.

Validation for fitting and accessory pressure drop calculations

The new calculation type:

- FittingAndAccessoryCalculationType.ValidateCurrentSettings

can be passed by Revit to servers as a member of the bitmask in PipeFittingAndAccessoryPressureDropData.CalculationType and DuctFittingAndAccessoryPressureDropData.CalculationType.

ValidateCurrentSettings indicates that the server should validate the settings stored in the current entity. A server should implement this calculation type if its settings can become invalid after changes such as flow. The server should return the status of the validation in the new properties:

- PipeFittingAndAccessoryPressureDropData.IsCurrentEntityValid
- DuctFittingAndAccessoryPressureDropData.IsCurrentEntityValid

If the setting validation returns false, Revit will replace the setting with the default one, and update the fitting pressure drop accordingly.

Family Connector Info

The new class:

- MEPFamilyConnectorInfo

adds the following methods:

- MEPFamilyConnectorInfo.GetAssociateFamilyParameterId() - Gets the associate family parameter id of the specified connector parameter id.
- MEPFamilyConnectorInfo.GetConnectorParameterValue() - Gets the parameter value of the specified connector parameter id.

Revit Link API additions

Link instance locations

The new method:

- RevitLinkInstance.MoveBasePointToHostBasePoint()

will move a RevitLinkInstance so that the link's base point and host project's base point are in the same location.

The new method:

- RevitLinkInstance.MoveOriginToHostOrigin()

moves this link instance so that the internal origin of the linked document is aligned to the internal origin of the host document.

Both methods cause a one-time movement and do not set up any shared coordinates relationship.

Local unload of Revit Links

The new method:

- RevitLinkType.UnloadLocally()

allows unloading a Revit link in a workshared file for the current user only. When another user opens their local model, the link will still be loaded for them. This method accepts an instance of a new interface class:

- ISaveSharedCoordinatesCallbackForUnloadLocally

The response to the method in this interface is used to control Revit when trying to unload locally a Revit link with changes in shared coordinates.

The new method:

- RevitLinkType.RevertLocalUnloadStatus()

turns off a user's local link override. If the link is loaded for other users, this function will reload the link. If the link is unloaded for other users, then the link will remain unloaded, but the local unload override will be cleared.

Category API additions

Line Patterns

The new functions:

- Category.GetLinePatternId()
- Category.SetLinePatternId()

can be used to get or set the line pattern id associated with that category for the given graphics style type.