# Major changes and renovations to the Revit API

# 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.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, IList<double> weights)  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.