# Major changes and renovations to the Revit API

# API changes

## API enabled in perspective views

### Element modifications and contextual commands

Modification of many different object types is now allowed in perspective views. Most commands allowed in 3D orthographic views are now also allowed in perspective views. This includes contextual commands. Annotations are an exception and cannot be created in perspective views.

### External commands and applications

External API commands and applications are now enabled by default in perspective views. The following behavior applies:

* Any external command that was automatically disabled by Revit when a perspective view is active will now be active.  No code changes are necessary to make this happen.
* Any external command that was explicitly disabled in perspective views will remain disabled. For example, if your command has an accessibility function limiting it to use in certain view types, the command will not be accessible in perspective views unless that function accepts ViewType.ThreeD as a view type that is allowed.
* Macros, the Macro Manager tools, Dynamo scripts and the Dynamo editor are also newly enabled when a perspective view is active.

## Subelements

Several Revit elements can now contain a subdivision known as a Subelement.     Subelements provide a way for parts of an element to behave as though they were real elements without incurring the overhead of adding more full elements to the model.

Many Revit features – for example parameters, schedules, and tags – were designed to operate on Elements.  As a result, the Revit code needs to represent objects as Elements for them to participate in those features.  This can lead to scalability problems, because every Element adds overhead and adding many Elements may decrease the performance of the model.

An alternative is to use Subelements.  An element can expose a set of "Subelements" that it contains, specifying characteristics like their category and parameters, and certain Revit capabilities will treat those Subelements the same as ordinary Elements.  For example, a Subelement may contribute geometry to the main element and may be able to be selected independently of its parent Element.  It will possibly have its own (settable) type as well as an assigned category which can be different from its parent Element.

In the API, the new Subelement class is used to refer to either an Element or a specific subelement of a given Element.    It is typically directly related to a Reference to either the Element or the specific subelement.

Note that creation of new Subelements for a given element is not done generically.  Instead, the given Element may provide the ability to modify its definition, resulting in the creation of new Subelements.

Examples of Elements which may incorporate Subelements in practice include:

* Rebar
* RebarContainer
* FabricSheet
* Stairs elements which make up MultistoryStairs elements
* Railing
* ContinuousRail

To get access to a particular Subelement, you may use any of the following:

* Subelement.Create()
* Subelement.IsValidSubelementReference()
* Document.GetSubelement(Reference) - Gets the subelement referenced by the input reference.
* Document.GetSubelement(String uniqueId) - Gets the subelement referenced by a unique id string.
* Element.GetSubelements() - Returns the collection of the Element's Subelements.

To access the basic Subelement properties, including its category and geometry, use:

* Subelement.GetBoundingBox()
* Subelement.GetGeometryObject()
* Subelement.IsModifiable()
* Subelement.Document
* Subelement.Element
* Subelement.Category
* Subelement.GetReference()
* Subelement.UniqueId
* ExportUtils.GetExportId(Subelement)

To access the Subelement's type, use:

* Subelement.TypeId
* Subelement.ChangeTypeId()
* Subelement.GetValidTypes()
* Subelement.IsValidType()
* Subelement.CanHaveTypeAssigned()

To access the Subelement's parameters, use:

* Subelement.GetAllParameters()
* Subelement.GetParameterValue()
* Subelement.SetParameterValue()
* Subelement.IsParameterModifiable()
* Subelement.HasParameter()

For Elements which allow deletion of individual Subelements, use:

* Element.DeleteSubelement()
* Element.DeleteSubelements()
* Element.CanDeleteSubelement()

### References and selection of subelements

The new enumerated value:

* ObjectType.Subelement

provides the ability to prompt a user to select subelements interactively using Selection.PickObject() or Selection.PickObjects().

The new enumerated value:

* ElementReferenceType.REFERENCE\_TYPE\_SUBELEMENT

identify a reference as a reference to a specific subelement.

The new methods:

* Reference.EqualTo()
* Reference.Contains()

provide useful checks related to the contents of a given Reference object, applicable to subelement references as well as other types of references.

## Changes to APIs for accessing version

 The property:

* Application.IsSubscriptionUpdate

has been deprecated and replaced by:

* Application.SubVersionNumber

The new property returns a string representing the major-minor version number for the Revit application. For example, "2018.0.0".  This number is updated for major and minor updates.

In RevitAddinUtility, the similar property:

* RevitProduct.IsSubscriptionUpdate

has been deprecated and replaced by:

* RevitProduct.ReleaseSubVersion

This new string property returns a similar major-minor version number for installed Revit versions.

## Asset API Changes

The following API classes moved from the Autodesk.Revit.Utility namespace to a new namespace Autodesk.Revit.DB.Visual:

* AssetProperty
* AssetProperties
* Asset
* AssetSet
* AssetType
* AssetPropertyType
* AssetPropertyDouble
* AssetPropertyDoubleArray2d
* AssetPropertyDoubleArray3d
* AssetPropertyDoubleArray4d
* AssetPropertyDoubleMatrix44
* AssetPropertyFloat
* AssetPropertyFloatArray
* AssetPropertyInt64
* AssetPropertyUInt64
* AssetPropertyBoolean
* AssetPropertyDistance
* AssetPropertyEnum
* AssetPropertyReference
* AssetPropertyString
* AssetPropertyTime
* AssetPropertyList

Two AssetProperty properties were deprecated and replaced:

|  |  |
| --- | --- |
| **Deprecated** | **Replacement (new methods)** |
| AssetPropertyDouble3.Value | IList<double> APropertyDouble3::GetValueAsDoubles()  XYZ APropertyDouble3::GetValueAsXYZ() |
| AssetPropertyDouble4.Value | IList<double> APropertyDouble4::GetValueAsDoubles() |

Values for the enumerated type AssetPropertyType were renamed to better adhere to the API standards. Note that corresponding integer values are the same:

|  |  |  |
| --- | --- | --- |
| **Old values** | **Replacement** | **Integer value** |
| APT\_Unknown | Unknown | 0 |
| APT\_Properties | Properties | 1 |
| APT\_Boolean | Boolean | 2 |
| APT\_Enum | Enumeration | 3 |
| APT\_Integer | Integer | 4 |
| APT\_Float | Float | 5 |
| APT\_Double | Double1 | 6 |
| APT\_DoubleArray2d | Double2 | 7 |
| APT\_DoubleArray3d | Double3 | 8 |
| APT\_DoubleArray4d | Double4 | 9 |
| APT\_Double44 | Double44 | 10 |
| APT\_String | String | 11 |
| APT\_Time | Time | 12 |
| APT\_Distance | Distance | 14 |
| APT\_Asset | Asset | 15 |
| APT\_Reference | Reference | 16 |
| APT\_Int64 | Longlong | 17 |
| APT\_UInt64 | ULonglong | 18 |
| APT\_List | List | 19 |
| APT\_FloatArray | Float3 | 20 |

## Dynamic Updaters on Reload Latest

Dynamic updaters are now triggered on Reload Latest for the elements added or changed in the central file.

## Export to DWG/DXF API change

The new AutoCAD version (R2018) has been added to the ACADVersion enumerated type.  This is now the default version used when exporting to DWG and DXF.

## UIDocument.PromptForFamilyInstancePlacement() behavioral change

The behavior for UIDocument.PromptForFamilyInstancePlacement() was changed to be same as that of PickObject() methods - the placement operation will be cancelled when the "x" button of Revit is clicked during the placement operation instead of closing Revit directly.

## IndependentTag API changes

The new method:

* IndependentTag.Create()

replaces Autodesk.Revit.Creation.Document.NewTag(), which has been marked obsolete.  The new version supports tagging of either elements or subelements.

The new method:

* IndependentTag.GetTaggedReference()

returns a reference to the item which has been tagged.  This reference may be to a Subelement, which can also be identified by:

* IndependentTag.IsTaggedOnSubelement()

The following properties and their methods now throw more informative exceptions:

* IndependentTag.LeaderElbow:
  + IndependentTag.GetLeaderElbow() - "The tag does not have a leader or its leader is straight."
  + IndependentTag.SetLeaderElbow() - "The tag does not have a leader."
* IndependentTag.LeaderEnd:
  + IndependentTag.GetLeaderEnd() - "There is no leader end because the tag does not use a free end leader."
  + IndependentTag.SetLeaderEnd() - "There is no leader end because the tag does not use a free end leader."

The new property:

* IndependentTag.HasElbow

indicates if the leader of the tag has an elbow point or not.

## Shared Coordinates API changes

The following properties have been deprecated and replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| ProjectLocation.ProjectPosition | ProjectLocation.GetProjectPosition()  ProjectLocation.SetProjectPosition() |
| ProjectLocation.SiteLocation | ProjectLocation.GetSiteLocation() |

## New DirectShape behaviors

DirectShape elements now support new behaviors.  Other than the limitations listed below, no code changes are required to enable these new behaviors for DirectShape elements.

* Tagging - if the element's assigned category supports tagging and a tag type exists, the DirectShape can be tagged with Revit tagging tools.
* Edge dimensions - if the DirectShape is referenceable, it will now support dimensioning to edge references as well as face references.
* Connector elements - in families, if the DirectShape is referenceable, DirectShape planar faces can be used to host connector elements.
* Rebar hosting - if the DirectShape is of any of the following categories it can now act as a host for rebar:
  + OST\_StructuralFraming
  + OST\_StructuralColumns
  + OST\_StructuralFoundation
  + OST\_StructuralConnections
  + OST\_Walls
  + OST\_Floors
  + OST\_EdgeSlab
  + OST\_Parts
  + OST\_Stairs
  + OST\_GenericModels

## Rebar API Changes

Rebar now supports two different layout options: shape-driven and free-form.  Previously, all Rebar elements were shape-driven.  The new methods:

* Rebar.GetShapeDrivenAccessor() - Returns the Shape Driven interface which exposes specific Shape Driven logic.
* Rebar.GetFreeFormAccessor() - Returns the Free Form interface which exposes specific Free Form logic.
* Rebar.IsRebarFreeForm - Returns true if the rebar is free form and false if shape driven.
* Rebar.IsRebarShapeDriven - Returns true if the rebar is shape driven and false if free form.

Rebar members which are applicable only for shape-driven rebar have been deprecated and replaced with equivalents in the class RebarShapeDrivenAccessor.   Specifically, the following members are deprecated:

|  |  |
| --- | --- |
| **Deprecated member** | **Replacement method** |
| Rebar.GetDistributionPath() | RebarShapeDrivenAccessor.GetDistributionPath() |
| Rebar.ComputeDrivingCurves() | RebarShapeDrivenAccessor.ComputeDrivingCurves() |
| Rebar.GetBarPositionTransform() | RebarShapeDrivenAccessor.GetBarPositionTransform() |
| Rebar.ScaleToBox() | RebarShapeDrivenAccessor.ScaleToBox() |
| Rebar.ScaleToBoxFor3D() | RebarShapeDrivenAccessor.ScaleToBoxFor3D() |
| Rebar.SetLayoutAsSingle() | RebarShapeDrivenAccessor.SetLayoutAsSingle() |
| Rebar.SetLayoutAsFixedNumber() | RebarShapeDrivenAccessor.SetLayoutAsFixedNumber() |
| Rebar.SetLayoutAsMaximumSpacing() | RebarShapeDrivenAccessor.SetLayoutAsMaximumSpacing() |
| Rebar.SetLayoutAsNumberWithSpacing() | RebarShapeDrivenAccessor.SetLayoutAsNumberWithSpacing () |
| Rebar.SetLayoutAsMinimumClearSpacing() | RebarShapeDrivenAccessor.SetLayoutAsMinimumClearSpacing() |
| Rebar.Normal | RebarShapeDrivenAccessor.Normal |
| Rebar.BarsOnNormalSide | RebarShapeDrivenAccessor.BarsOnNormalSide |
| Rebar.Height | RebarShapeDrivenAccessor.Height |
| Rebar.ArrayLength | RebarShapeDrivenAccessor.ArrayLength |
| Rebar.BaseFinishingTurns | RebarShapeDrivenAccessor.BaseFinishingTurns |
| Rebar.MultiplanarDepth | RebarShapeDrivenAccessor.MultiplanarDepth |
| Rebar.TopFinishingTurns | RebarShapeDrivenAccessor.TopFinishingTurns |
| Rebar.Pitch | RebarShapeDrivenAccessor.Pitch |
| Rebar.RebarShapeId | getter - Rebar.GetShapeId() setter - RebarShapeDrivenAccessor.SetRebarShapeId() |

## FabricSheet API Changes

The following methods have been deprecated and replaced:

|  |  |  |
| --- | --- | --- |
| **Deprecated method** | **Replacement method** | **Notes** |
| FabricSheetType.SetLayoutAsCustomPattern(double, double , double , double , IList<FabricWireItem> , IList<FabricWireItem>) | FabricSheetType.SetLayoutAsCustomPattern(double, double , IList<FabricWireItem> , IList<FabricWireItem>) | Both end overhangs will now be read only and computed internally. |
| FabricWireItem.Create(double distance, double wireLength, ElementId wireType) | FabricWireItem.Create(double distance, double wireLength, ElementId wireType, double wireOffset) | Older calls can use the new method with wireOffset set to 0.0. |

## Structural Section API Changes

Several section properties were moved from subclasses to the base class StructuralSection:

* StructuralSection.ElasticModulusStrongAxis
* StructuralSection.ElasticModulusWeakAxis
* StructuralSection.MomentOfInertiaStrongAxis
* StructuralSection.MomentOfInertiaWeakAxis
* StructuralSection.NominalWeight
* StructuralSection.Perimeter
* StructuralSection.PlasticModulusStrongAxis
* StructuralSection.PlasticModulusWeakAxis
* StructuralSection.PrincipalAxesAngle
* StructuralSection.SectionArea
* StructuralSection.ShearAreaStrongAxis
* StructuralSection.ShearAreaWeakAxis
* StructuralSection.TorsionalModulus
* StructuralSection.TorsionalMomentOfInertia
* StructuralSection.WarpingConstant

Several specific structural sections offer new constructors with additional input parameters.  Their original constructors have been deprecated.

* StructuralSectionCSlopedFlange
* StructuralSectionISlopedFlange
* StructuralSectionISplitSlopedFlange
* StructuralSectionLAngle
* StructuralSectionStructuralTees

Several new specific structural section classes have been introduced:

* StructuralSectionGeneralC - Defines parameters for Channel Cold Formed shape.
* StructuralSectionGeneralCEx - Defines parameters for Channel with Fold Cold Formed shape.
* StructuralSectionGeneralF - Defines parameters for Flat Bar.
* StructuralSectionGeneralH - Defines parameters for Rectangular Pipe structural section.
* StructuralSectionGeneralI - Defines parameters for general Double T shape.
* StructuralSectionGeneralLA - Defines parameters for Angle Cold Formed structural section.
* StructuralSectionGeneralLZ - Defines parameters for Z Cold Formed shape.
* StructuralSectionGeneralR - Defines parameters for pipes.
* StructuralSectionGeneralS - Defines parameters for Round Bar structural section.
* StructuralSectionGeneralT - Defines parameters for Tees shape.
* StructuralSectionGeneralU - Defines parameters for general Channel shape.
* StructuralSectionGeneralW - Defines parameters for Angle structural section.

In addition, the API for structural sections offers a few other new capabilities:

* StructuralSection.GetBoundarySize() - returns the size of the section boundary.
* StructuralSection.AnalysisParams - accesses a common set of parameters for structural analysis which can be associated to a section.
* StructuralSection.StructuralSectionGeneralShape - returns an enumerated value identifying the general shape for the structural section representing geometry only.
* StructuralSectionHotRolled.FlangeThicknessLocation - this new property has been introduced for this class and the specific sections that derive from it.
* StructuralSectionHotRolled.WebThicknessLocation - this new property has been introduced for this class and the specific sections that derive from it.
* StructuralSectionUtils.GetStructuralElementDefinitionData() - returns data defining the section and the position of the structural element.

## ElectricalSystem API changes

The following functions have been deprecated and replaced:

|  |  |
| --- | --- |
| **Deprecated member** | **New/replacement member** |
| Autodesk.Revit.Creation.Document.NewElectricalSystem(Connector, ElectricalSystemType) | ElectricalSystem.Create(Connector, ElectricalSystemType) |
| Autodesk.Revit.Creation.Document.NewElectricalSystem(ICollection < ElementId > , ElectricalSystemType) | ElectricalSystem.Create(Document, IList<ElementId>, ElectricalSystemType) |

## Pipe Pressure Loss Calculation change

The methods:

* Autodesk.Revit.DB.Plumbing.PipeSettings.GetPressLossCalculationServerInfo()
* Autodesk.Revit.DB.Plumbing.PipeSettings.SetPressLossCalculationServerInfo()

have been deprecated in Revit 2018 and will be removed in the next version of Revit. Custom pipe pressure loss calculations will no longer be supported. Similar functionality can be accessed by setting Autodesk.Revit.DB.Plumbing.PipeSettings.AnalysisForClosedLoopHydronicPipingNetworks.

## Corrected names of AutoRouteFailures values

The following BuiltInFailures.AutoRouteFailures values were renamed due to spelling errors:

* AttemptToConnectNonSlopingElementToSlopedPipeWarning (renamed from AttemptToComnnectNonSlopingElementToSlopedPipeWarning)
* AttemptToConnectNonSlopingElementToSlopedPipeError (renamed from AttemptToComnnectNonSlopingElementToSlopedPipeError)

## 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:

#### Classes

* Autodesk.Revit.DB.Plumbing.PipeType
* Autodesk.Revit.DB.Structure.FabricSheet.BentFabricWiresOrientation
* Autodesk.Revit.DB.Structure.BentFabricWiresOrientation
* Autodesk.Revit.DB.FormattedTextRun
* Autodesk.Revit.DB.FormatUtils
* Autodesk.Revit.DB.PropertySetLibrary

#### Methods

* Autodesk.Revit.DB.NurbSpline.Create(IList<XYZ>, IList<Double>) : NurbSpline
* Autodesk.Revit.DB.NurbSpline.Create(IList<XYZ>, IList<Double>, IList<Double>, Int32, Boolean, Boolean) : NurbSpline
* Autodesk.Revit.Creation.Application.NewPlane(CurveArray) : Plane
* Autodesk.Revit.Creation.Application.NewPlane(XYZ, XYZ) : Plane
* Autodesk.Revit.Creation.Application.NewPlane(XYZ, XYZ, XYZ) : Plane
* Autodesk.Revit.CurveAPIUtils.CreateNurbSpline(IList<XYZ>, IList<Double>) : NurbSpline
* Autodesk.Revit.CurveAPIUtils.CreateNurbSpline(IList<XYZ>, IList<Double>, IList<Double>, Int32, Boolean, Boolean) : NurbSpline
* Autodesk.Revit.Creation.Document.NewDuct(Connector, Connector, DuctType) : Duct
* Autodesk.Revit.Creation.Document.NewDuct(XYZ, Connector, DuctType) : Duct
* Autodesk.Revit.Creation.Document.NewDuct(XYZ, XYZ, DuctType) : Duct
* Autodesk.Revit.DB.Structure.LoadCase.Create(Document, String, ElementId, LoadNatureCategory) : LoadCase
* Autodesk.Revit.DB.WallFoundation.GetFoundationType() : WallFoundationType
* Autodesk.Revit.DB.WallFoundation.SetFoundationType(WallFoundationType) : Void
* Autodesk.Revit.DB.ViewSchedule.GroupHeaders(Int32, Int32, Int32, Int32) : Void
* Autodesk.Revit.DB.PointClouds.PointCloudOverrides.GetPointCloudOverrideSettings(ElementId, String, Document) : PointCloudOverrideSettings
* Autodesk.Revit.DB.PointClouds.PointCloudOverrides.GetPointCloudOverrideSettings(ElementId) : PointCloudOverrideSettings
* Autodesk.Revit.DB.PointClouds.PointCloudOverrides.SetPointCloudOverrideSettings(ElementId, PointCloudOverrideSettings, String, Document) : Void
* Autodesk.Revit.DB.PointClouds.PointCloudOverrides.SetPointCloudOverrideSettings(ElementId, PointCloudOverrideSettings) : Void
* Autodesk.Revit.DB.DirectShape.CreateElement(Document, ElementId, String, String) : DirectShape
* Autodesk.Revit.DB.DirectShape.CreateElementInstance(Document, ElementId, ElementId, String, Transform, String, String) : DirectShape
* Autodesk.Revit.DB.DirectShape.SetGUIDs(String, String) : Void
* Autodesk.Revit.DB.TextNode.GetFormattedTextRuns() : IList<FormattedTextRun>
* Autodesk.Revit.DB.View.GetVisibility(Category) : Boolean
* Autodesk.Revit.DB.View.SetVisibility(Category, Boolean) : Void
* Autodesk.Revit.Creation.Document.NewFoundationWall(WallFoundationType, Wall) : WallFoundation
* Autodesk.Revit.DB.Ellipse.Create(XYZ, double, double, XYZ, XYZ, double, double) : Ellipse

#### Properties

* Autodesk.Revit.DB.Structure.LoadCase.NatureCategory : LoadNatureCategory
* Autodesk.Revit.DB.Mechanical.MechanicalSystem.Flow : Double
* Autodesk.Revit.DB.Mechanical.MechanicalSystem.StaticPressure : Double
* Autodesk.Revit.DB.Plumbing.PipingSystem.FixtureUnits : Double
* Autodesk.Revit.DB.Plumbing.PipingSystem.Flow : Double
* Autodesk.Revit.DB.Plumbing.PipingSystem.StaticPressure : Double
* Autodesk.Revit.DB.ScheduleField.HasTotals : Boolean
* Autodesk.Revit.DB.Events.DocumentPrintingEventArgs.Settings : IPrintSetting
* Autodesk.Revit.DB.Events.ViewPrintingEventArgs.Settings : IPrintSetting
* Autodesk.Revit.DB.Mechanical.DuctFittingAndAccessoryConnectorData.Coordination : Transform
* Autodesk.Revit.DB.Plumbing.PipeFittingAndAccessoryConnectorData.Coordination : Transform
* Autodesk.Revit.UI.Events.DialogBoxShowingEventArgs.HelpId : Int32

#### Enumerated types

* Autodesk.Revit.DB.Structure.LoadNatureCategory
* Autodesk.Revit.Attributes.TransactionMode.Automatic

# API additions

## API to get list of reviewable warnings from a Document

The new method:

* Document.GetWarnings()

returns a list of failure messages generated from persistent (reviewable) warnings accumulated in the document.

## API access to FamilyInstance references

The following new methods have been added to enable easy access to FamilyInstance references that correspond to reference planes and reference lines in the family.  Some use the options in the new enumeration FamilyInstanceReferenceType as input to identify "Strong" or "Weak" references or specific positional references in each of the 3 coordinate directions (as determined by the possible values of parameter "Is Reference" of reference planes and parameter "Reference" of reference lines in families).

* FamilyInstance.GetReferences()
* FamilyInstance.GetReferenceByName()
* FamilyInstance.GetReferenceType()
* FamilyInstance.GetReferenceName()

## Multistory Stairs API

The new class:

* MultistoryStairs

allows users to create stairs that span multiple levels.  A multistory stairs element may contain multiple stairs whose extents are governed by base and top levels.

This element will contain one or more Stairs elements. Stairs elements are either a reference instance which is copied to each level covered by groups of identical stairs instances which share the same level height, or individual Stairs instances which are not connected to a group with the same level height. By default, when adding new levels to the multistory stair, new stairs will be added to the group.

For groups of duplicate stairs at different levels, the instances can be found as Subelements of the Stairs element.

Stairs in a connected group can be edited together by modifying the associated Stairs instance. For specific floors that need special designs, stairs can be separated from a group by unpinning the element, changes made to this Stairs will not affect other any other instance in the element, or add the stairs back into the group if needed. However, any changes made to the stair will be lost since the stair's properties will be overridden by the group specifications.

The class has the following methods:

* MultistoryStairs.AddStairsByLevelIds() - Adds stairs to the given levels.
* MultistoryStairs.RemoveStairsByLevelIds() - Removes stairs from the given levels. This will regenerate the multistory stairs from the remaining levels.
* MultistoryStairs.CanAddStair() - Checks if the input level id can be used to add stairs into multistory stairs.
* MultistoryStairs.CanRemoveStair() - Checks if the input level id can be used to remove stairs from the multistory stairs.
* MultistoryStairs.GetAllConnectedLevels() - Gets the ids of all levels connected to the multistory stairs.
* MultistoryStairs.GetAllStairsIds() - Gets the ids of all of the stairs in the multistory stairs.
* MultistoryStairs.Create() - Creates a multistory stairs object.
* MultistoryStairs.GetStairsConnectedBaseLevelIds() - Gets the base level ids for the stairs contained in this multi-story stairs element.
* MultistoryStairs.IsPinned() - Checks if a stair is pinned as a propagation group.
* MultistoryStairs.Unpin() - Unpins a story of stairs by giving its base level id.
* MultistoryStairs.Pin() - Pins a unpinned stairs back into a story of a stairs.

The new property:

   Stairs.MultistoryStairsId

indicates the id of the MultistoryStairs element to which the Stairs belong to.

Related to StairsPath functionality for multistory stairs, the new functions:

* StairsPath.CanCreateOnMultistoryStairs()
* StairsPath.CreateOnMultistoryStairs()

support creation of new stairs paths in a plan view for stairs instances in a multistory stairs element.

### Railings API additions related to MultistoryStairs

The new methods:

* Railing.GetMultistoryStairsPlacementLevels()
* Railing.SetMultistoryStairsPlacementLevels()
* Railing.GetSubelementOnLevel()

provide access to Railing elements which are hosted on members of Multistory stairs.

The method:

* Railing.Create(Document document, ElementId multistoryStairsId, ElementId levelId,  ElementId railingTypeId,  RailingPlacementPosition placePosition)

Creates new railings on a given level of given multistory stairs - the created railing will be hosted on stairs included in MultistoryStairs element on given level.

The existing method:

* Railing.Create(Document document, ElementId stairsId,  ElementId railingTypeId,  RailingPlacementPosition placePosition)

now supports MulitstoryStairs elements as input, where the resulting railing will be added on every level of given stairs in multistory stairs.

The existing property:

* Railing.HostId

now supports stairs or stairs components coming from MultistoryStairs elements as well.

## Dimension API additions

### DimensionEqualityLabelFormating API

The new class:

* DimensionEqualityLabelFormatting

allows users to set dimension equality formulas for continuous linear or angular dimensions.

New methods and properties include:

* DimensionEqualityLabelFormatting.LeadingSpaces
* DimensionEqualityLabelFormatting.LabelType
* DimensionEqualityLabelFormatting.Prefix
* DimensionEqualityLabelFormatting.Suffix
* DimensionEqualityLabelFormatting.GetFormatOptions()
* DimensionEqualityLabelFormatting.SetFormatOptions()

The following new methods in DimensionType allow access to equality formulas:

* DimensionType.GetEqualityFormula()
* DimensionType.SetEqualityFormula()

### UnitsFormatOptions in DimensionType

The new functions:

* DimensionType.GetUnitsFormatOptions()
* DimensionType.SetUnitsFormatOptions()

allow users to get or set the FormatOptions for a DimensionType.

### OrdinateDimensionSetting

The new class:

* OrdinateDimensionSetting

allows users to customize ordinate dimensions.

The new enum:

* OrdinateDimensionLineStyle

allows users to choose continuous or segmented line styles for their dimensions.

The new enums:

* OrdinateTextOrientation
* OrdinateTextPosition

allows users to orient text in relation to the dimension lines or witness lines.

The new enum:

* OrdinateOriginVisibility

allows users to control visibility of their dimensions.

New properties in OrdinateDimensionSetting include:

* OrdinateDimensionSetting.DimLineLength
* OrdinateDimensionSetting.DimLineStyle
* OrdinateDimensionSetting.TextOrientation
* OrdinateDimensionSetting.TextPosition
* OrdinateDimensionSetting.OriginVisibility
* OrdinateDimensionSetting.OriginTickMarkId

The following new methods in DimensionType allow access to the OrdinateDimensionSetting:

* DimensionType.GetOrdinateDimensionSetting()
* DimensionType.SetOrdinateDimensionSetting()

## SpatialElementTag API additions

SpatialElementTag is the base element for Room, Area and Space tag classes.

The following new properties have been added:

* SpatialElementTag.HasElbow - Identifies if the tag's leader has an elbow point or not.
* SpatialElementTag.TagText - The text displayed by the tag.

## Geometry API additions

### Surface and Face API

The new method:

* Face.GetSurface()

returns a copy of the Face's surface.

In order to correctly leverage this information, the following properties:

* Face.OrientationMatchesSurfaceOrientation
* Surace.OrientationMatchesParametricOrientation

identify whether the face's orientation matches the surface orientation, and whether the surface orientation is the same as or opposite to its parametric orientation.

### RevolvedSurface API

The new method:

* RevolvedSurface.GetProfileCurveInWorldCoordinates()

returns a copy of the profile curve expressed in the world coordinate system.

### RuledSurface API

The newly added methods:

* RuledSurface.HasFirstProfilePoint()
* RuledSurface.HasSecondProfilePoint()

check if a point was used to define one of the surface profiles.

## Level API addition

The new method:

* Level.FindAssociatedPlanViewId()

finds the id of the first available associated floor or structural plan view associated with this level. If there are multiple associated views, Revit will return the first one it finds.

## Dockable Frame API Additions

Custom Dockable Panes now support the ability for display of dynamic UI elements, such as web browser controls.  This capability should be used in cases where the UI for the pane (layout, buttons etc.) changes dynamically during the lifetime of the Revit session. To use this, implement the new interface:

* IFrameworkElementCreator

with a method:

* IFrameworkElementCreator.CreateFrameworkElement()

that constructs and returns the WPF Framework element to embedded in the Revit dockable pane.

The new members:

* DockablePaneProviderData.GetFrameworkElement()
* DockablePaneProviderData.FrameworkElementCreator

provide the ability for the application to deliver a dynamic framework element to the dockable pane.

The property:

* DockablePaneProviderData.FrameworkElement

is now allowed to be null, in situations where the FrameworkElement will be dynamically created.

## DirectContext3D for display of externally managed 3D graphics in Revit

DirectContext3D is an API for displaying external graphics in the context of a Revit model. The API provides a more connected experience to users who can benefit from the ability to display graphics based on geometry that is either difficult or costly to fully import into Revit.

An external plugin can use DirectContext3D API to render geometry by encoding it inside pairs of vertex and index buffers. The communication between Revit and the plugin is accomplished with the use of the External Service Framework (ESF). Revit’s rendering pipeline asks registered servers of the DirectContext3D service to provide the geometry for rendering. Revit informs the plugin about certain rendering state, such as the display style and whether the current rendering pass is for transparent objects. The plugin also communicates certain information to Revit, such as the bounding box of the geometry to be rendered.

The following list contains the major added classes and their descriptions:

* DirectContext3D.IDirectContext3DServer - The interface to be implemented by a server of the DirectContext3D external service.
* DirectContext3D.DrawContext - A class that provides drawing functionality for use by DirectContext3D servers.
* DirectContext3D.Vertex - The base class for DirectContext3D vertices.
* DirectContext3D.VertexStream - The base class for DirectContext3D vertex streams, which are used to write vertex data into buffers.
* DirectContext3D.VertexBuffer - A buffer that stores vertex data for rendering.
* DirectContext3D.VertexFormat - A specification of the format of vertex data contained in a piece of geometry.
* DirectContext3D.VertexFormatBits - Vertex format (i.e., the type of data associated with a vertex) represented as a number.
* DirectContext3D.IndexPrimitive - The base class for index buffer primitives.
* DirectContext3D.IndexStream - The base class for DirectContext3D index streams, which are used to write vertex indices into buffers.
* DirectContext3D.IndexBuffer - A buffer that stores vertex indices for rendering.
* DirectContext3D.EffectInstance - An effect instance that controls the appearance of geometry.
* DirectContext3D.PrimitiveType - Type of geometry primitive represented as a number.
* DirectContext3D.ClipPlane - A set of parameters representing a clip plane in DirectContext3D.
* DirectContext3D.ProjectionMethod - The projection method used by a DirectContext3D camera.
* DirectContext3D.Camera - A collection of camera settings for DirectContext3D.

### View update for DirectContext3D

The new method:

* UIDocument.UpdateAllOpenViews()

updates all open views in this document after elements have been changed, deleted, selected or de-selected. Graphics in the views are fully redrawn regardless of which elements have changed.  This function should only rarely be needed, but might be required when working with graphics drawn from outside of Revit's transactions and elements, for example, when using DirectContext3D.

This function is potentially expensive as many views may be updated at once, including regeneration of view's geometry and redisplay of graphics.  Thus for most situations it is recommended that API applications rely on the Revit application framework to update views more deliberately.

## Coordination Model elements

Coordination Model elements current can link the graphical contents of Navisworks files and display them in context in the Revit session.  These elements leverage the DirectContext3D framework to handle the display of the external graphics, and are the first example of an element which is designated to contain a link to externally managed DirectContext3D graphics (a "DirectContext3D handle" element).

There is no current way to create new Coordination Model or DirectContext3D handle elements via the API.   However, you can use the capabilities of the related classes to identify and manipulate these elements. These elements can be accessed from the following new API classes:

* DirectContext3DHandleUtils - provides utilities related to the identification of types and instances which are storing externalized graphics via DirectContext3D
* DirectContext3DHandleSettings - provides access to override settings applied to DirectContext3D handles through the Visibility/Graphics dialog.
* DirectContext3DHandleOverrides - provides access to DirectContext3DHandleSettings that are stored by a given view.

## Shared Coordinates API additions

### Acquire and Publish coordinates API additions

Two new methods allow users to acquire and publish shared coordinates:

* Document.AcquireCoordinates() - Acquires project coordinates from the specified link instance. This method accepts both Revit links (RevitLinkInstance) and DWG links (ImportInstance).
* Document.PublishCoordinates() - Publishes shared coordinates to the specified ProjectLocation. This method works only on Revit links.

### SiteLocation API additions

Two new read-only properties have been added to provide information on the geographic coordinate system of a SiteLocation. The geographic coordinate system is imported from a DWG file from AutoCAD or Civil 3D. If the SiteLocation has geographic coordinate system information, the latitude and longitude of the SiteLocation will be updated automatically   
when the model's Survey Point is moved.

* SiteLocation.GeoCoordinateSystemId - Gets a string corresponding to geographic coordinate system ID, such as "AMG-50" or "Beijing1954/a.GK3d-40" for the SiteLocation. The value will be the empty string if there is no coordinate system specified for the SiteLocation. This property is read-only.
* SiteLocation.GeoCoordinateSystemDefinition - Gets an XML string describing the geographic coordinate system. The value will be the empty string if there is no coordinate system specified for the SiteLocation. This property is read-only.

The new method:

* SiteLocation.IsCompatibleWith()- Checks whether the geographic coordinate system of this site is compatible with the given site.

### ProjectLocation API additions

The new method:

* ProjectLocation.Create()

creates a new ProjectLocation in the document from the given SiteLocation and with the given name.

### Revit Link API additions

The new method:

* static RevitLinkInstance.Create(ImportPlacement placement)

creates a new instance of a linked Revit project (RevitLinkType). Instances will be placed origin-to-origin or by shared coordinates according to the input placement type.

## Link API additions

### External Resource framework additions

IExternalResourceServer can now provide CAD format links, DWF markups, and IFC links. The following new values have been added to ExternalResourceType.BuiltInExternalResourceTypes:

* CADLink
* DWFMarkup
* IFCLink

### CADLinkType additions

Several new methods have been added to CADLinkType as part of external resource framework enhancements:

* CADLinkType.Reload() - Reloads a CADLinkType from its current location.
* CADLinkType.Reload(CADLinkOptions options) - Reloads a CADLinkType from its current location, including options to control graphic overrides.
* CADLinkType.LoadFrom(String fileName) - Reloads a CADLinkType from a file on disk.
* CADLinkType.LoadFrom(ExternalResourceReference reference) - Reloads a CADLinkType from an external resource server.

### ImportInstance additions

Several new methods have been added to create ImportInstance elements, either from an existing link type or a new link type:

* ImportInstance.Create(DWGImportOptions options, String fileName) - Creates a new DWG or DXF type and instance from a file on disk.
* ImportInstance.Create(DWGImportOptions options, ExternalResourceReference reference) - Creates a new DWG or DXF type and instance from an external resource location.
* ImportInstance.Create(ElementId typeId) - Creates a new DWG or DXF instance from an existing CADLinkType.

Methods have also been added to create DGN, SAT, and SKP links from external resource locations.

### IFC Link API additions

The method:

* RevitLinkType.UpdateFromIFC()

now has a new version that allows for an ExternalResourceReference to specify the IFC file name, instead of a string.

## Workshared operation progress changed events

Subscribe to the Autodesk.Revit.ApplicationServices.Application.WorksharedOperationProgressChanged event to be notified when progress has changed during Collaboration for Revit's synchronizing.

This event consists of several phases. Different event arguments are used during each phase.

### WorksharedOperationProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.WorksharedOperationProgressChangedEventArgs

provides a base class for event arguments for worksharing-based progress events.

### DocumentSaveToLocalProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.DocumentSaveToLocalProgressChangedEventArgs provides information during the save to local phase of the DocumentSynchronizingWithCentralProgressChanged event.

It has the following properties:

* DocumentSaveToLocalProgressChangedEventArgs.Location - Full path of the central model which is to be synchronized.
* DocumentSaveToLocalProgressChangedEventArgs.Status - Gets the API event status, which reflects the current operation execution status.
* DocumentSaveToLocalProgressChangedEventArgs.BeforeSaveToCentral - True if the "save to local" operation is occurring before "save to central"; false if after.
* DocumentSaveToLocalProgressChangedEventArgs.FinishedStreams - The number of streams finished since the last time this event was raised.
* DocumentSaveToLocalProgressChangedEventArgs.TotalStreams - The total expected number of streams to save to local.
* DocumentSaveToLocalProgressChangedEventArgs.SaveToLocalFinished - Indicates if the current "save to local" operation has finished.

### DataTransferProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.DataTransferProgressChangedEventArgs provides information during the data transferring phase of the WorksharedOperationProgressChanged event. Generally, it is not used directly because It is the base class of Autodesk.Revit.DB.Events.DocumentReloadLatestProgressChangedEventArgs, Autodesk.Revit.DB.Events.DocumentSaveToCentralProgressChangedEventArgs and Autodesk.Revit.DB.Events.CreateRelatedFileProgressChangedEventArgs

It has the following properties:

* DataTransferProgressChangedEventArgs .Location - Full path of the central model which is to be synchronized.
* DataTransferProgressChangedEventArgs .Status - Gets the API event status, which reflects the current operation execution status.
* DataTransferProgressChangedEventArgs .TransferMode - Gets the transfer mode of this data transfer progress. It is Undefined or Download or Upload;
* DataTransferProgressChangedEventArgs .Speed - Gets data transfer speed(bytes/second).
* DataTransferProgressChangedEventArgs .FinishedSize - Gets downloaded or uploaded data size, in bytes, since the last time this event was raised.
* DataTransferProgressChangedEventArgs .TotalSize - Total expected data size to download, in bytes.

### DocumentReloadLatestProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.DocumentReloadLatestProgressChangedEventArgs provides information during the reload latest phase of the DocumentSynchronizingWithCentralProgressChanged event.

It has the following properties:

* DocumentReloadLatestProgressChangedEventArgs.Location - The full path of the central model which is to be synchronized.
* DocumentReloadLatestProgressChangedEventArgs.Status - Gets API event status, reflecting the current operation execution status.
* DocumentReloadLatestProgressChangedEventArgs.TransferMode - Returns DataTransferMode.Download.
* DocumentReloadLatestProgressChangedEventArgs.RetryTimes - The number of times Revit has tried to Reload Latest. Its value is '0' at the first time.
* DocumentReloadLatestProgressChangedEventArgs.Speed - Download speed(bytes/second) in this event.
* DocumentReloadLatestProgressChangedEventArgs.FinishedSize - The downloaded data size, in bytes, since the last time this event was raised.
* DocumentReloadLatestProgressChangedEventArgs.TotalSize - Total expected data size to download, in bytes.
* DocumentReloadLatestProgressChangedEventArgs.IsMerging - Indicates if Revit is merging downloaded data.
* DocumentReloadLatestProgressChangedEventArgs.ReloadLatestFinished - Indicates if current 'reload latest' progress is finished or not.

### DocumentSaveToCentralProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.DocumentSaveToCentralProgressChangedEventArgs

provides information during the save to central phase of the DocumentSynchronizingWithCentralProgressChanged event.

It has the following properties:

* DocumentSaveToCentralProgressChangedEventArgs.Location - Full path of the central model which is to be synchronized.
* DocumentSaveToCentralProgressChangedEventArgs.Status - Gets API event status, reflecting the current operation execution status.
* DocumentSaveToCentralProgressChangedEventArgs.TransferMode - Returns DataTransferMode.Upload.
* DocumentSaveToCentralProgressChangedEventArgs.RetryTimes - The number of times Revit has tried to Save to Central." Its value is '0' at the first time.
* DocumentSaveToCentralProgressChangedEventArgs.Speed - Upload speed(bytes/second) of saving to central.
* DocumentSaveToCentralProgressChangedEventArgs.FinishedSize - The uploaded data size, in bytes, since the last time this event was raised.
* DocumentSaveToCentralProgressChangedEventArgs.TotalSize - Total expected uploaded data size to save to central, in bytes.
* DocumentSaveToCentralProgressChangedEventArgs.SaveToCentralFinished - Indicates if current 'save to central' is finished or not.
* DocumentSaveToCentralProgressChangedEventArgs.FailureDueToConflicts - Indicates the 'save to central' failure is caused by submission, that is because another user finished saving to central first.

### CreateRelatedFileProgressChangedEventArgs

The new class:

* Autodesk.Revit.DB.Events.CreateRelatedFileProgressChangedEventArgs

provides information about the creation of related files when Collaboration for Revit models are being opened.

It has the following properties:

* CreateRelatedFileProgressChangedEventArgs.Location - Full path of the model which is to be created related file.
* CreateRelatedFileProgressChangedEventArgs.Status - Gets API event status, reflecting the current operation execution status.
* CreateRelatedFileProgressChangedEventArgs.TransferMode - Returns DataTransferMode.Download.
* CreateRelatedFileProgressChangedEventArgs.CreatingCloudSharedLocal - Indicates if it is creating cloud shared local model.
* CreateRelatedFileProgressChangedEventArgs.DownloadFinished - Indicates if all data downloads are finished or not.
* CreateRelatedFileProgressChangedEventArgs.Speed - Download speed(bytes/second) in this event.
* CreateRelatedFileProgressChangedEventArgs.FinishedSize - The downloaded data size, in bytes, since the last time this event was raised.
* CreateRelatedFileProgressChangedEventArgs.TotalSize - Total expected data size to download, in bytes.
* CreateRelatedFileProgressChangedEventArgs.FullDownload - Indicates if download the full data of the document, which will take longer than subsequent downloads.

## Events related to linked resources

Subscribe to the events:

* Autodesk.Revit.ApplicationServices.Application.LinkedResourceOpening
* Autodesk.Revit.ApplicationServices.ControlledApplication.LinkedResourceOpening
* Autodesk.Revit.ApplicationServices.Application.LinkedResourceOpened
* Autodesk.Revit.ApplicationServices.ControlledApplication.LinkedResourceOpened

to be notified when Revit is just about to open, or has just opened, a linked resource.  This event supports linked resources of the following types : Revit; IFC; CAD (DWG, DXF, DGN, SAT).

This event will not be raised if there are no updates for linked CAD or IFC resources.  The linked resource cannot be modified during these events.

### LinkedResourceOpeningEventArgs

The new class:

* Autodesk.Revit.DB.Events.LinkedResourceOpeningEventArgs

provides information when Revit is just about to open a linked resource.

It has the following properties:

* LinkedResourceOpeningEventArgs.ResourceType - Linked resource type.
* LinkedResourceOpeningEventArgs.LinkedResourcePathName - The linked resource name to open.

### LinkedResourceOpenedEventArgs

The new class:

* Autodesk.Revit.DB.Events.LinkedResourceOpenedEventArgs

provides information when Revit finishes opening a linked resource.

It has the following properties:

* LinkedResourceOpenedEventArgs.ResourceType - Linked resource type.
* LinkedResourceOpenedEventArgs.LinkedResourcePathName - Opened linked resource name.

## Events related to parallel View Export

When exporting multiple views to DWF or DWFx format, Revit uses background processes to compute the exported output of several views in parallel. Two events are now available on the Application class to receive progress updates during a parallel export operation:

* Application.ViewExporting - This event is raised when Revit is just about to export a view of the document.
* Application.ViewExported - This event is raised immediately after Revit has finished exporting a view of the document.

These two events are raised only during accelerated export jobs, in which views are exported in parallel using a background process. Accelerated export only occurs when exporting to DWF formats and not combining views into a single file.

## DWG export API additions

The new enumerated value:

* ExportColorMode.TrueColorPerView

specified an export where all colors from the Revit project will be exported as 24-bit RGB values as specified in view (where ExportColorMode.TrueColor indicates that all colors from the Revit project will be exported as 24-bit RGB values as specified in object styles)

The new properties:

* ACADExportOptions.UseHatchBackgroundColor
* ACADExportOptions.HatchBackgroundColor

allow assignment of a color that will be set as hatch background color on the exported hatch.

### ExportDWGSettings API additions

Two new methods have been added to ExportDWGSettings:

* ExportDWGSettings.FindByName() - Returns the pre-defined DWG export settings in the given document with the specified name.
* ExportDWGSettings.GetActivePredefinedSettings() - Returns the active DWG export settings in the given document.

Note that these functions cannot return in-session settings. To access the in-session settings, save them with a name.

## Part API additions

The new methods:

* Part.ResetFaceOffset()
* Part.GetFaceOffset()

provide further capabilities to access and manipulate the offset applied to a given face of a Part element.

## Freeform Rebar API additions

The new methods:

* Rebar.CreateFreeForm(Document, RebarBarType, Element, IList<IList<Curve>>, out RebarFreeFormValidationResult)
* Rebar.CreateFreeForm(Document, RebarBarType, Element, IList<CurveLoop>, out RebarFreeFormValidationResult)

Create a freeform rebar instance where bars are created along the path of the input curves.  Constraints cannot later be added to this rebar.

Freeform rebar instances have properties accessible from the RebarFreeFormAccessor class, which can be obtained from:

* Rebar.GetFreeFormAccessor()

The new members of this accessor class allow you to change the bar curves only for unconstrained rebar:

* RebarFreeFormAccessor.SetCurves()
* RebarFreeFormAccessor.IsUnconstrained()

They also provide access to layout, distribution path and hook properties of the rebar, if applicable:

* RebarFreeFormAccessor.SetLayoutAsSingle()
* RebarFreeFormAccessor.SetLayoutAsNumberWithSpacing()
* RebarFreeFormAccessor.SetLayoutAsFixedNumber()
* RebarFreeFormAccessor.SetLayoutAsMaximumSpacing()
* RebarFreeFormAccessor.SetLayoutAsMinimumClearSpacing()
* RebarFreeFormAccessor.GetCustomDistributionPath()
* RebarFreeFormAccessor.SetHookPlaneNormalForBarIdx()
* RebarFreeFormAccessor.GetHookPlaneNormalForBarIdx()
* RebarFreeFormAccessor.SetHookOrientationAngle()
* RebarFreeFormAccessor.GetHookOrientationAngle()

## Custom Service for Freeform Rebar Definition

The new interface class:

* IRebarUpdateServer

 represents an interface that can be overridden to drive the generation and update of freeform rebar geometry.  The interface features several methods:

* IRebarUpdateServer.GetCustomHandles() - This method should define all handles that the Rebar instance will have. This method is called when the Rebar is created.
* IRebarUpdateServer.GetCustomHandleName() - This function should return the user-visible name of the handle.  This function is called during an editing operation for the Rebar constraints when the mouse is over a handle
* IRebarUpdateServer.GenerateCurves() - This function should calculate the bar geometry.
* IRebarUpdateServer.TrimExtendCurves() - This function should calculate the trim and/or extension of the curves that were obtained from calling GenerateCurves(), as well as to assign new constraints for the start and end handles.
* IRebarUpdateServer.GetHandlesPosition() - This function is supposed to provide the positions of handles defined in GetCustomHandles(). These positions will be shown on screen when the bar constraints are edited.  
  If a position for a handle isn't provided, that handle will not be represented on screen while edit constraints.

The interfaces methods each provide an input argument which carries data needed for the calculation, as well as the results.  These new argument classes are:

* RebarHandlesData
* RebarHandleNameData
* RebarUpdateCurvesData
* RebarCurvesData
* RebarTrimExtendData
* RebarHandlePositionData

You can programmatically create a freeform Rebar whose geometry and constraints are governed by the external service using:

* Rebar.Create(Document, GUID, RebarBarType, Element)

You can convert a FreeForm rebar governed by an external service to an unconstrained element using:

* RebarFreeFormAccessor.DisconnectFromServer()

In order to access and modify constraints associated to any custom handles associated to the service-driven freeform Rebar, you may use the members of the RebarConstrainedHandle class:

* RebarConstrainedHandle.IsCustomHandle()
* RebarConstrainedHandle.HandleName
* RebarConstrainedHandle.GetCustomHandleTag()

and use the new members of RebarConstraint and RebarConstraintsManager:

* RebarConstraint.Create(RebarConstrainedHandle, IList<Reference>, bool, double)
* RebarConstraint.ReplaceReferenceTargets()
* RebarConstraint.NumberOfTargets
* RebarConstraint.GetTargetElement()
* RebarConstraint.GetRebarConstraintTargetHostFaceType()
* RebarConstraint.GetTargetHostFaceAndTransform()
* RebarConstraint.GetTargetCoverType()
* RebarConstraint.GetTargetHostFaceReference()
* RebarConstraint.GetCustomHandleTag()
* RebarConstraintsManager.GetAllHandles()
* RebarConstraintsManager.AllowConstraintTargets()

## HVACSpaceType and Space air ventilation API

The following new properties have been added to Autodesk.Revit.DB.Space:

* Space.SpaceTypeId - Gets or sets the Space type element of the Space.
* Space.OutdoorAirPerPerson - Gets the specified outdoor air per person, in cubic feet per second.
* Space.OutdoorAirPerArea - Gets the specified outdoor air per area, in feet per second.
* Space.AirChangesPerHour - Gets the specified air changes per hour of the space.
* Space.OutdoorAirflow - Gets the outdoor airflow of the Space, in cubic feet per second.
* Space.OutdoorAirFlowStandard - Gets the outdoor air flow standard of the space.

## Energy Analysis API additions

Energy Analysis API additions

Several new classes have been added to allow better control over spaces when doing energy analysis.

The new class:

* Autodesk.Revit.DB.Analysis.HVACLoadType

is the base class for HVACLoadSpaceType and HVACLoadBuildingType.

It has the following properties:

* Autodesk.Revit.DB.Analysis.HVACLoadType.AirChangesPerHour - The air changes per hour.
* Autodesk.Revit.DB.Analysis.HVACLoadType.AreaPerPerson - The area per person.
* Autodesk.Revit.DB.Analysis.HVACLoadType.LatentHeatGainPerPerson -  The latent heat gain per person.
* Autodesk.Revit.DB.Analysis.HVACLoadType.LightingLoadDensity - The lighting load density.
* Autodesk.Revit.DB.Analysis.HVACLoadType.OutdoorAirPerArea -  The outdoor air per area.
* Autodesk.Revit.DB.Analysis.HVACLoadType.OutdoorAirPerPerson - The outdoor air per person.
* Autodesk.Revit.DB.Analysis.HVACLoadType.OutdoorAirFlowStandard-The outdoor airflow standard.
* Autodesk.Revit.DB.Analysis.HVACLoadType.PlenumLighting - The percentage of plenum lighting contribution.
* Autodesk.Revit.DB.Analysis.HVACLoadType.PowerLoadDensity - The power load density.
* Autodesk.Revit.DB.Analysis.HVACLoadType.SensibleHeatGainPerPerson - The sensible heat gain per person.

The new class:

* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType

allows users to set the usage type associated with the space, such as “Dining Area” or “Lobby”.

It has the following methods and properties:

* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType.Create()
* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType.IsNameUnique(String)
* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType.IsNameUnique(Document, String)
* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType.IsPlenum
* Autodesk.Revit.DB.Analysis.HVACLoadSpaceType.SpaceTypeName

The new class:

* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType

allows users to set the energy analysis usage type associated with a building, such as “Museum” or “Office”.

It has the following methods and properties:

* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.Create()
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.IsNameUnique(String)
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.IsNameUnique(Document, String)
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.BuildingTypeName
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.OpeningTime
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.ClosingTime
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.IsValidTime() - Checks if the string can be parsed to a valid time for opening time and closing time. A valid string can be "16:30" or "4:30 PM".
* Autodesk.Revit.DB.Analysis.HVACLoadBuildingType.UnoccupiedCoolingSetPoint - The unoccupied cooling set point of the building type with unit kelvin (K).

The new properties:

* Autodesk.Revit.DB.Analysis.MassZone.SpaceTypeId
* Autodesk.Revit.DB.Mechanical.Space.SpaceTypeId

allow users to set the energy analysis type associated with a space or mass zone.

Note that these classes are not ElementTypes. They represent properties of a space or building when doing energy analysis, and do not have instances.

## Electrical API additions

### Electrical Circuit Path API

The new API members:

* ElectricalSystem.PathOffset
* ElectricalSystem.HasPathOffset
* ElectricalSystem.HasCustomCircuitPath
* ElectricalSystem.CircuitPathMode
* ElectricalSystem.GetCircuitPath()
* ElectricalSystem.SetCircuitPath()
* ElectricalSystem.IsCircuitPathValid()

support different options and operations related to electrical circuit paths.

### PanelScheduleView API

The following new methods handle space or spare circuit operations:

* PanelScheduleView.AddSpace() - Add a space at specific cell.
* PanelScheduleView.AddSpare() - Add a spare at specific cell.
* PanelScheduleView.RemoveSpace() - Remove a space at specific cell.
* PanelScheduleView.RemoveSpare() - Remove a spare at specific cell.

## MEPAnalyticalConnection API additions

Two new classes have been added to allow API users to create analytical connections between elements. The behavior is the same as in Revit's UI.

The new class:

* MEPAnalyticalConnection

represents an analytical element that connected mechanical equipment to a piping network. The analytical connection can be used to create a network even if no real pipes are placed yet. It enables the fast establishment of a piping network to analyze the flow and pressure, especially at the early stage of the design process.

It has the following methods:

* MEPAnalyticalConnection.Create() - Creates a new analytical connection between two connectors.
* MEPAnalyticalConnection.CreateMultipleConnections() - Creates new analytical connections between the equipment connector and the nearest point on the piping network.
* MEPAnalyticalConnection.GetFlow() - Returns the up-to-date flow value. If the network is asynchronously calculated, this method would wait until the calculation is completed.

The new class:

* MEPAnalyticalConnectionType

is the type element of an MEPAnalyticalConnection. Its pressure loss value is included in the network critical path calculation.

It has the following methods and properties:

* MEPAnalyticalConnectionType.Create() - Creates a new analytical connection type with the specified name.
* MEPAnalyticalConnectionType.IsNameUnused() - Checks if the name is already used.
* MEPAnalyticalConnectionType.GetAllTypes() - Returns all types in the document.
* MEPAnalyticalConnectionType.PressureLoss - The pressure loss associated with this type. This property is writable.

The new property:

* Autodesk.Revit.DB.Plumbing.PipeSettings.AnalysisForClosedLoopHydronicPipingNetworks

indicates whether to enable analysis for closed loop hydronic piping networks in the background.

## Fabrication API additions

### Hanger Rod additions

The following new methods and properties allow better control of hanger rod lengths:

* FabricationRodInfo.GetRodStructureExtension()
* FabricationRodInfo.SetRodStructureExtension() - Sets the extension of the rod into the structure. The rod will remain hosted by structure and have the extension applied on top of any calculated length. This method is only valid for rods which are already hosted by structure.
* FabricationRodInfo.CanRodsBeHosted - Setting this to false disassociates the hanger from any structure and allows the user to manually set lengths. Setting it to true re-enables rods automatically hosting to structure again.
* FabricationRodInfo.GetRodLength() - Gets the current rod length for the specified rod index (including any extension), whether it is hosted or not.
* FabricationRodInfo.SetRodLength() - Sets the current rod length for the specified rod index - but only if hosting is disabled by setting CanRodsBeHosted = false first.

The new property:

* FabricationPart.HangerRodKit

allows users to get or set the support rod kit override. This gives the user better control over hanger rod thicknesses.

### Placing and splitting fabrication parts

The following methods allow placing and splitting fabrication parts:

* FabricationPart.AlignPartByConnectorToConnector() - Aligns a part by its connector to another connector.
* FabricationPart.AlignPartByInsertionPoint() - Aligns the part by its insertion point to a point and rotation in free space.
* FabricationPart.AlignPartByConnector() - Aligns the part by its connector to a point and rotation in free space.
* FabricationPart.AlignPartByInsertionPointAndCutInToStraight() - Aligns the part by its insertion point to a point and rotation in on a straight.
* FabricationPart.HasNoConnections() - Checks to see if all the part's connectors are open.
* FabricationPart.CanSplitStraight() - Checks to see if a straight can be split into two at a given position.
* FabricationPart.SplitStraight() - Splits a straight at a given position.

### Fabrication Part export additions

The new method:

* Autodesk.Revit.DB.Fabrication.FabricationUtils.ExportToPCF()

exports Fabrication parts to PCF format.

The new property:

* FabricationPart.SpoolName

allows control over Spool names in PCF output.

### Fabrication Part status additions

Two properties were added to FabricationPart to allow the user to query and set the part fabrication status field:

* FabricationPart.PartStatus - Queries or changes the part fabrication status field current value. Setting the value will track when it was changed.
* FabricationPart.PartGuid - Queries the part's fabrication globally unique identifier, as used in applications such as Tracker.

### Fabrication part comparison

The new method:

* FabricationPart.IsSameAs()

compares basic dimensional information of two fabrication parts but also allows the user to specify additional fabrication data to ignore. This allows users to compare parts including fabrication data not exposed in Revit.

### Fabrication Part ancillary usage additions

The new method:

* FabricationPart.GetPartAncillaryUsage()

gets a list of FabricationAncillaryUsage structures that lists the ancillaries that the fabrication part uses. This method can be used to populate cut lists and procure ancillaries needed to fabricate and install the fabrication part.

The new class:

* FabricationAncillaryUsage

contains the ancillary usage data. It has the following read-only properties:

* FabricationAncillaryUsage.Type - A new enumerated type FabricationAncillaryType which describes what kind of ancillary it is (eg. a fixing, airturn vane or support material).
* FabricationAncillaryUsage.UsageType - a new enumerated type FabricationAncillaryUsageType describing the usage of the ancillary (eg. in a support, connector or airturn).
* FabricationAncillaryUsage.AncillaryId - the ancillary database identifier as listed by FabricationConfiguration.GetAncillaries().
* FabricationAncillaryUsage.Length
* FabricationAncillaryUsage.AncillaryWidthOrDiameter - The width / diameter of the ancillary (eg. rod thickness).
* FabricationAncillaryUsage.AncillaryDepth
* FabricationAncillaryUsage.Quantity
* FabricationAncillaryUsage.ProductCode

### Fabrication Part custom data additions

The following methods allow access to Fabrication Part custom data defined per part. This can be used for querying and changing values:

* FabricationPart.HasCustomData() - Queries if a part has a certain optional custom data.
* FabricationPart.Get/SetPartCustomDataText/Integer/Real() - Gets or sets the fabrication part's custom data.
* FabricationPart.Add/RemovePartCustomData() - Adds or removes optional custom data on fabrication parts.

### FabricationConfiguration additions

Mew methods have been added to access fabrication data defined by the currently loaded fabrication configuration. These can be used to query what fabrication data is available for fabrication parts to use:

* FabricationConfiguration.GetAllDampers()
* FabricationConfiguration.GetDamperName()
* FabricationConfiguration.GetAncillaries() - list all ancillary identifiers (by enumerated type)
* FabricationConfiguration.GetAncillaryName()
* FabricationConfiguration.GetAllPartStatuses()
* FabricationConfiguration.GetPartStatusDescription()
* FabricationConfiguration.GetAllPartCustomData()
* FabricationConfiguration.GetPartCustomDataName()
* FabricationConfiguration.GetPartCustomDataType()

### Routing exclusions additions

The following methods allow users to override routing exclusions:

* FabricationService.IsGroupExcluded() - Gets whether a service group is excluded from routing.
* FabricationService.SetServiceGroupExclusions() - Sets the service group exclusions.
* FabricationService.ResetServiceExclusionOverrides() - Resets the overridden service group and button exclusions back to default.
* FabricationService.OverrideServiceButtonExclusion() - Overrides the default service button exclusions.
* FabricationServiceButton.IsExcluded() - Gets if the service button is excluded from routing.

### FabricationServiceButton addition

The new property:

* FabricationServiceButton.IsStraight

returns true if all conditions of the fabrication service button are straight pieces. This indicates the button is suitable for use with multi-point routing, among other uses.