**Major changes and renovations to the Revit API**

**.NET 4.0 for compilation**

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

**New macro environment**

VSTA has been replaced by SharpDevelop.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

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

**Revit.ini registration disabled**

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

**Document.Element properties replaced**

The indexed properties:

* Document.Element[ElementId]
* Document.Element[String]

have been replaced by methods:

* Document.GetElement(ElementId)
* Document.GetElement(String)

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

**Replacements for use of old-style collections**

**ElementArray and ElementSet are now obsolete**

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

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

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

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

**Definitions and DefinitionGroups**

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

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

**GeometryElement.Objects**

The property

* GeometryElement.Objects

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

**Replacements for wall creation methods**

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

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

These methods replace the obsolete methods on Autodesk.Revit.Creation.Document (NewWall() (all overloads)). The NewWalls() overloads are obsoleted without dedicated replacements, as described here: [ElementArray and ElementSet are now obsolete](#RevitAPIChanges2013-ObsoleteAPICleanup)

**Idling event and External Events**

**Idling event frequency**

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

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

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

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

* IdlingEventArgs.SetRaiseWithoutDelay()

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

**Idling event with no active document**

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

**External Events framework**

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

1. Implement an external event by deriving from the interface:
   * IExternalEvent
2. Create an ExternalEvent with
   * ExternalEvent.Create()
3. When an event happens in your modeless UI where a Revit action needs to be taken, call
   * ExternalEvent.Raise()
4. Revit will wait for an available Idling timecycle and call the implementation of your event:
   * IExternalEvent.Execute()

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

**Calling OpenAndActivateDocument during events**

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

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

**Document & worksharing changes**

**Worksharing properties**

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

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

replace the properties

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

The new members:

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

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

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

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

The property:

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

The property:

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

**BasicFileInfo**

The new class

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

**Point cloud engine API changes**

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

**Free()**

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

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

**GetOffset()**

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

**CreatePointSetIterator()**

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

**IFC export APIs**

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

**Construction Modeling changes**

The following members are now obsolete:

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

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

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

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

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

The methods:

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

**Structure changes**

**Rebar.GetCenterlineCurves()**

The overload taking one argument is now obsolete.

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

* adjust for self-intersection
* include hooks
* include bend radius curves (fillets)

**Rebar member changes**

These members are now obsolete:

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

**AreaReinforcement and PathReinforcement**

**BarDescription class**

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

* AreaReinforcement.NumBarDescriptions()
* AreaReinforcement.GetBarDescription()
* PathReinforcement.NumBarDescriptions()
* PathReinforcement.GetBarDescription()

are obsolete in 2013.

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

New methods:

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

**MEP changes**

**ConnectorProfileType, PartType**

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

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

**ConnectorElement**

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

* IsPrimary - moved to ConnectorElement
* AssignAsPrimary - moved to ConnectorElement
* LinkedConnector - moved to ConnectorElement as GetLinkedConnectorElement and SetLinkedConnectorElement
* Please query the parameters RBS\_PIPE\_CONNECTOR\_SYSTEM\_CLASSIFICATION\_PARAM, RBS\_DUCT\_CONNECTOR\_SYSTEM\_CLASSIFICATION\_PARAM, or RBS\_ELEC\_CIRCUIT\_TYPE on ConnectorElement to obtain the PipeSystemType, DuctSystemType, and ElectricalSystemType values on the now obsolete PipeConnector, DuctConnector, and ElectricalConnector elements.

Connector element types can now be created using new methods:

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

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

**Obsolete API Cleanup**

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

**Classes removed**

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

**Methods and properties removed**

* AnalysisDisplayLegend.Width - replaced by legend settings of AnalysisDisplayStyle.
* AnalysisDisplayLegendSettings.SetTextTypeId(ElementId , Document)
* AnalysisDisplayMarkersAndTextSettings.SetTextTypeId(ElementId , Document)
* AnalyticalModel methods
  + IsValidAnalyticalProjectionType(AnalyticalDirection , AnalyticalProjectionType) replaced by IsValidProjectionType
  + GetAnalyticalProjectionDatumPlane(AnalyticalDirection ) replaced by GetAnalyticalProjectionDatumPlane(AnalyticalElementSelector , AnalyticalDirection)
  + IsValidDatumPlaneForProjection(AnalyticalDirection, ElementId ) replaced by IsValidProjectionDatumPlane
  + SetAnalyticalProjectionDatumPlane(AnalyticalDirection , ElementId ) replaced by setAnalyticalProjectionDatumPlane(AnalyticalElementSelector, AnalyticalDirection, ElementId)
  + SetAnalyticalOffset(XYZ) replaced by SetOffset
  + GetAnalyticalOffset() replaced by GetOffset
  + GetAnalyticalProjectionType(AnalyticalDirection) replaced by GetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection)
  + SetAnalyticalProjectionType(AnalyticalDirection , AnalyticalProjectionType ) replaced by SetAnalyticalProjectionType(AnalyticalElementSelector, AnalyticalDirection, AnalyticalProjectionType)
* IFCExportOptions.CurrentViewOnly
* PointCloudFilter.GetFilterOutline() - replaced by PointCloudFilterUtils.GetFilteredOutline()
* RebarHostData.HasCoverTypeForReference(Reference ) - replaced by IsFaceExposed()
* SpatialFieldManager members
  + CurrentUnits replaced by AnalysisResultSchema.CurrentUnits and SpatialFieldManager.SetResultSchema()
  + UpdateSpatialFieldPrimitive(int , Autodesk.Revit.DB.Analysis.FieldDomainPoints, Autodesk.Revit.DB.Analysis.FieldValues)
  + SetUnits(System.Collections.Generic.IList<System.String> , System.Collections.Generic.IList<double>) replaced by AnalysisResultSchema.SetUnits() and SpatialFieldManager.SetResultSchema()
  + GetDescription() replaced by SpatialFieldManager.SetResultSchema() and AnalysisResultSchema.Description
  + SetDescription(System.String) replaced by AnalysisResultSchema.Description and SpatialFieldManager.SetResultSchema()
* AnnotationSymbol.AsFamilyInstance
* Area.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
* Element members
  + ObjectType replaced by GetTypeId() and ChangeTypeId()
  + SimilarObjectTypes replaced by GetValidTypes()
* Materials members
  + AddOther(System.String) replaced by Material.Create
  + AddWood(System.String) replaced by Material.Create
  + AddConcrete(System.String) replaced by Material.Create
  + AddSteel(System.String) replaced by Material.Create
  + AddGeneric(System.String) replaced by Material.Create
* PropertySetElement.Create(Document, MaterialAspect) replaced by PropertySetElement.Create(Document, StructuralAsset) and PropertySetElement.Create(Document, ThermalAsset)
* Mullion.AsFamilyInstance
* Panel.AsFamilyInstance
* Rebar.IsShownInFrontOfGeometryInView[View] - replaced by IsUnobscuredInView() and SetUnobscuredInView()
* PointOnEdge members
  + PointOnEdge(Autodesk.Revit.DB.Reference, double) replaced by PointLocationOnCurve
  + CurveParameter replaced by LocationOnCurve
* Room.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
* Face.MaterialElement - replaced by MaterialElementId
* Reference members
  + Element replaced by Document.GetElement(Reference)
  + GeometryObject replaced by Element.GetGeometryObjectFromReference(Reference)
  + Transform replaced by ReferenceByContext.GetInstanceTransform() after obtaining ReferenceWithContext from Document.FindReferencesWithContextByDirection()
  + ProximityParameter replaced by ReferenceByContext.ProximityParameter after obtaining ReferenceWithContext from Document.FindReferencesWithContextByDirection()
* PipingSystem members
  + FluidType, FluidType is now a property of the system type. Get the system type, and set the parameter RBS\_PIPE\_FLUID\_TYPE\_PARAM directly
  + FluidTemperature, FluidTemperature is now a property of the system type. Get the system type, and set the parameter RBS\_PIPE\_FLUID\_TEMPERATURE\_PARAM directly
  + FluidDensity, FluidDensity is now a property of the system type. Get the system type, and set the parameter RBS\_PIPE\_FLUID\_DENSITY\_PARAM directly
  + FluidViscosity, FluidViscosity is now a property of the system type. Get the system type, and set the parameter RBS\_PIPE\_FLUID\_VISCOSITY\_PARAM directly
* Space.Boundary - replaced by GetBoundarySegments(SpatialElementBoundaryOptions)
* Application.LibraryPaths - replaced by GetLibraryPaths() and SetLibraryPaths()
* ControlledApplication.LibraryPaths - replaced by GetLibraryPaths() and SetLibraryPaths()
* Application.NewPointOnEdge(Autodesk.Revit.DB.Reference , double edgeParam) - replaced by other constructor that uses PointLocationOnCurve
* Document members
  + Move() - all overloads - replaced by methods of ElementTransformUtils
  + Rotate() - all overloads - replaced by methods of ElementTransformUtils
  + Mirror() - all overloads - replaced by methods of ElementTransformUtils
  + Array() - all overloads - replaced by methods of LinearArray and RadialArray
  + ArrayWithoutAssociate() - all overloads - replaced by methods of LinearArray and RadialArray
  + SaveAs(System.String, bool) - replaced by SaveAs(String, SaveAsOptions)
  + FindReferencesByDirection() - replaced by FindReferencesWithContextByDirection() and ReferenceIntersector methods
  + ExternalFileReference & ImportFileData - all methods and enums related to material paths
  + AbbreviationItem - empty class removed completely
  + WorksharingCentralFilename - replaced by GetWorksharingCentralModelPath()
* ModelPathUtils.GetRevitServerPrefix() - RSN:// is the prefix string for Revit Server path.

**Major enhancements to the Revit API**

**Stairs and railings API**

**Stairs and stairs components**

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

* Stairs
* StairsType

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

The classes:

* StairsLanding
* StairsLandingType
* StairsRun
* StairsRunType

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

**Railings and railing components**

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

* Railing
* RailingType

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

**Stairs annotations**

The new class

* CutMarkType

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

The new classes

* StairsPath
* StairsPathType

provide access to the stairs path annotation.

**View API**

**View Creation**

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

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

IEnumerable<ViewFamilyType> viewFamilyTypes = from elem in new FilteredElementCollector(doc)

.OfClass(typeof(ViewFamilyType))

let type = elem as ViewFamilyType

where type.ViewFamily == ViewFamily.CeilingPlan

select type;

ViewPlan ceilingPlan = ViewPlan.Create(doc, viewFamilyTypes.First().Id, level.Id);

ceilingPlan.Name = "New Ceiling Plan for " + level.Name;

ceilingPlan.DetailLevel = ViewDetailLevel.Fine;

//3D views can be created with View3D.CreateIsometric and View3D.CreatePerspective.

//The new ViewOrientation3D object is used to get or set the orientation of 3D views.

View3D view = View3D.CreateIsometric(doc, viewFamily3d);

XYZ eyePosition = new XYZ(10, 10, 10);

XYZ upDirection = new XYZ(-1, 0, 1);

XYZ forwardDirection = new XYZ(1, 0, 1);

view.SetOrientation(new ViewOrientation3D(eyePosition, upDirection, forwardDirection));

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

**DisplayStyle**

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

**ViewDetailLevel**

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

**ViewRange**

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

**3D View Locking**

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

**View.Duplicate**

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

**UIView class and UIDocument.GetOpenUIViews()**

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

**Pan and Zoom**

The method

* UIView.ZoomAndCenterRectangle()

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

The method

* UIView.GetZoomCorners()

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

**PlanViewDirection**

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

**ViewFamilyType class and ViewFamily enum**

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

**Temporary view modes**

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

**Schedules**

**Schedule API**

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

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

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

**Running API commands in Schedule Views**

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

**Schedule export**

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

**Discipline controls**

The properties:

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

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

**Rendering Options**

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

The new methods:

* View3D.GetRenderingSettings
* View3D.SetRenderingSettings

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

The class

* RenderingSettings

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

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

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

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

Example: checking rendering options for a 3D view

|  |
| --- |
| RenderingSettings renderingSettings = view3D.GetRenderingSettings(); |
| Transaction newTran = new Transaction(doc, "Change Rendering Settings"); | |

|  |
| --- |
| newTran.Start(); |
| renderingSettings.BackgroundStyle = BackgroundStyle.Image; | |

|  |  |
| --- | --- |
| view3D.SetRenderingSettings(renderingSettings); | |
| newTran.Commit(); |

**Construction Modeling**

**Linked element parents for parts**

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

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

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

**Merged parts**

The methods:

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

provide support for merging of parts.

**Excluded parts**

The new property

* Part.Excluded

allows identification and assignment of parts to be excluded.

**Part category**

The new members:

* Part.OriginalCategoryId
* Part.GetSourceElementOriginalCategoryIds()

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

**Part division**

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

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

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

**Assembly views**

The method

* AssemblyViewUtils.AcquireAssemblyViews()

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

**Assembly instance transform and location**

The methods

* AssemblyInstance.GetTransform()
* AssemblyInstance.SetTransform()

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

The inherited property

* AssemblyInstance.Location

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

**Structure improvements**

**New Rebar members**

* RebarShapeMatchesCurvesAndHooks() checks if rebarShape matches curves and hooks
* CreateFromCurvesAndShape() creates a new instance of a Rebar element with default shape parameters from the RebarShape
* GetBarPositionTransform()
* GetBarExistsAtPosition()
* GetHookTypeId() and SetHookTypeId()
* ScheduleMark property
* TotalLength property
* Volume property

**ReinforcementSettings**

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

**AreaReinforcement/PathReinforcement**

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

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

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

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

**Analytical model**

The new element type

* AnalyticalLink

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

**MEP improvements**

**Routing Preferences**

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

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

**MEP Sections**

The new class

* MEPSection

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

**FluidType and FluidTemperature**

The new class

* FluidTemperature

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

The class

* FluidType

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

**Panel schedule - spare values**

The new functions

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

provide access to the parameter values of spare circuits.

**LabelUtils**

New overloads of

* LabelUtils.GetLabelFor()

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

**Physical Properties**

**Thermal properties**

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

* class ThermalProperties
  + Absorptance
  + HeatTransferCoefficient
  + Roughness
  + ThermalMass
  + ThermalResistance

This class is a property on the following types:

* ThermalProperties WallType.ThermalProperties
* ThermalProperties FloorType.ThermalProperties
* ThermalProperties CeilingType.ThermalProperties
* ThermalProperties RoofType.ThermalProperties
* ThermalProperties BuildingPadType.ThermalProperties
* class FamilyThermalProperties
  + AnalyticConstructionTypeId - This value corresponds to the 'id' property of a constructionType node in Constructions.xml, a supplied data library of thermal properties of common construction types.
* HeatTransferCoefficient
* SolarHeatGainCoefficient
* ThermalResistance
* VisualLightTransmittance
* Find() - returns a FamilyThermalProperties object from an "id" string in Constructions.xml
* FamilySymbol.GetThermalProperties() – Gets the thermal properties of a family symbol of a door, window, or curtain panel.
* FamilySymbol.SetThermalProperties()-- Sets the thermal properties of a family symbol of a door, window, or curtain panel.

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

* FamilyThermalProperties FamilySymbol.ThermalProperties

**Structural properties**

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

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

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

**Material assets**

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

**GBXML Export**

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

**User interface API improvements**

**Contextual help support**

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

* RibbonItem.SetContextualHelp()

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

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

* ContextualHelp.Launch()

**Support for Keyboard Shortcuts and Quick Access Toolbar**

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

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

**Replace implementation of commands**

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

* RevitCommandId

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

* AddInCommandBinding

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

* UIApplication.CreateAddInCommandBinding()
* UIControlledApplication.CreateAddInCommandBinding()

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

**Preview control**

The new class

* PreviewControl

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

**Options dialog customization**

Subscribe to the new event

* UIApplication.DisplayingOptionsDialog

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

* DisplayingOptionsDialogEventArgs.AddTab()

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

* TabbedDialogExtension

**Drag & Drop support**

The new methods

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

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

**Translation API enhancements**

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

**DGN import**

The new method

* Document.Import(string, DGNImportOptions, View)

supports import of DGN files into Revit views.

**DXF import**

The method

* Document.Import(string, DWGImportOptions)

now also supports DXF files for import.

**DWG and DXF export settings changes**

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

* BaseExportOptions

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

* BaseExportOptions.HideScopeBox
* BaseExportOptions.HideReferencePlane
* BaseExportOptions.HideUnreferenceViewTag

**DGN export**

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

* DGNExportOptions

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

The property

* DGNExportOptions.TemplateFile

has been replaced with:

* DGNExportOptions.SeedName

**FBX Export**

Three new options have been added:

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

**DividedPath**

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

The class

* DividedPath

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

**Analysis Visualization Framework**

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

* AnalysisDisplayStyleDeformedShapeTextLabelType class
* AnalysisDisplayDeformedShapeSettings class

A new override to CreateAnalysisDisplayStyle accepts an input argument of AnalysisDisplayDeformedShapeSettings

**Revit Link creation**

The API now contains two creation methods for Revit links.

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

**FilledRegion**

The class

* FilledRegion

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

The class

* FilledRegionType

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

**Light API**

The classes

* LightType
* LightFamily

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

Some examples of lighting parameters exposed are;

* initial color
* initial intensity
* loss factor
* color filter
* dimming color

**Light Group API**

The classes

* LightGroupManager
* LightGroup

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

The LightGroupManager class gives the user the ability to

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

The LightGroup class gives the user the ability to

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

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

**ReferenceIntersector**

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

Key members of ReferenceIntersector

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

**Small enhancements & API interface changes**

**Elements & filtering**

**Element.PhaseCreated and Element.PhaseDemolished**

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

* Element.CreatedPhaseId
* Element.DemolishedPhaseId

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

**Phase filter**

The new class

* PhaseFilter

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

**SelectionFilterElement**

The new class

* SelectionFilterElement

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

**Ceiling, Floor and CeilingAndFloor**

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

**Geometry & sketching**

**Split volumes**

The new method:

* SolidUtils.SplitVolumes()

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

**Solid tessellation**

The new method:

* SolidUtils.TessellateSolidOrShell()

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

* TriangulatedSolidOrShell

which allows access to the stored triangulated components and triangles.

Use the method:

* FacetingUtils.ConvertTrianglesToQuads()

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

**Face.Triangulate()**

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

**Ellipse - axes**

The new properties:

* Ellipse.XDirection
* Ellipse.YDirection

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

**GeometryElement.GetBoundingBox()**

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

**ItemFactoryBase.NewAlignment()**

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

**CylindricalHelix curve type**

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

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

**CurveLoop as IEnumerable<Curve>**

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

**CurveElement - center point reference**

The new property:

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

**ModelCurve.ChangeToReferenceLine**

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

**NewSketchPlane(Document, Reference)**

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

**Detailing & Annotations**

**MultiSegmentGrid class**

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

**Grid geometry**

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

**Diameter dimensions**

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

* FamilyItemFactory.NewDiameterDimension()

**DimensionSegment overrides**

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

* DimensionSegment.Prefix
* DimensionSegment.Suffix
* DimensionSegment.Above
* DimensionSegment.Below
* DimensionSegment.ValueOverride

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

**Detail element draw order**

The class

* DetailElementOrderUtils

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

**Families & content**

**SpatialElementCalculationPoint**

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

**RPC content assets**

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

**NewFamilyInstance validation**

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

**Family.PlacementType**

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

**DataStorage element**

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

**ApplicationInitialized event**

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

**ProgressChanged application event and ProgressChangedEventArgs**

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

* ProgressChangedEventArgs
  + Caption – The name of the current transaction or subtransaction in progress
  + Stage – The type of progress event (Started, RangeChanged, PositionChanged, CaptionChanged, UserCancelled, Finished)
  + Position – The numerical index between zero and UpperRange showing the number of completed steps of the current transaction or subtransaction
  + UpperRange – The total number of steps in the current transaction or subtransaction
  + Cancel - Cancels the current transaction in progress, if possible.
  + Cancellable - indicates if the current transaction can be cancelled.

**SiteLocation - PlaceName**

The new property:

* SiteLocation.PlaceName

provides the place name of the site location.

**Application.DefaultProjectTemplate**

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

**UnitType, DisplayUnit and FamilyParameter updates**

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

One UnitType was renamed: UT\_TemperalExp was renamed to UT\_ThermalExpansion.  
The corresponding ParameterType value (TemperalExp) was also renamed to ThermalExpansion.

**BasePoint and BASEPOINT\_ANGLETON\_PARAM "True North" Parameter**

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

**Journaling of Add-in user interface**

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

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