

SD124625

# **New API to Modify Visual Appearance of Materials in Revit**

Boris Shafiro  
Autodesk, Inc

## **Learning Objectives**

- Learn how to use new API to modify visual appearance of Materials in Revit
- Learn how to navigate coding workflow to edit rendering asset appearance for Revit Materials
- Learn how to use multiple schemas for regular and advanced materials in Revit
- Learn how to write a sample plug-in for basic modification of the visual appearance of Revit materials

## **Description**

The ability to use the Revit API to modify visual appearances of materials was among the top customer requests for years. This new API has been finally implemented. This class will present the new Materials API, coding workflows, and usage of multiple schemas for the visualization properties of materials in Revit software.

## **Speaker**

Boris Shafiro holds Ph.D. in Mechanical Engineering from Tufts University. He was an early member of the Revit startup team and joined Autodesk in 2002 through the acquisition. He worked as Principal Software Engineer and then Dev Manager in several areas of Revit development, including views, annotations, families, geometry, interoperability and API. He taught multiple classes on Revit API for ADN members. Currently he is a Team Manager, Scrum Master and the facilitator of Revit API Guild.

## Table of Contents

The basics .....	3
Terminology .....	3
Connection between Materials UI and API .....	4
Material API building blocks and relationship between classes.....	6
New editing capabilities in Materials API.....	8
Editing the properties of an Appearance Asset.....	8
Editing AssetProperty values.....	8
Connected Assets .....	10
Validation .....	11
Utilities .....	11
Schemas and property names .....	14
Standard Material Schemas .....	15
Advanced Material Schemas.....	15
Schemas for connected Assets .....	15
SDK sample plug-in AppearanceAssetEditing.....	19
Enable tint color property if supported .....	19
Edit tint color property.....	20

## The basics


Some of the API classes, methods and properties presented here are not new. The ability to access Materials and Appearance Assets in a read-only manner existed in Revit for several releases already. The new API functionality that was introduced in Revit 2018.1 is the ability to edit Appearance Asset properties including the textures associated with them.

New API namespace **Autodesk.Revit.DB.Visual** has been added. Many existing classes and all new classes (relevant to the visual appearance of Materials) were moved into this new namespace.

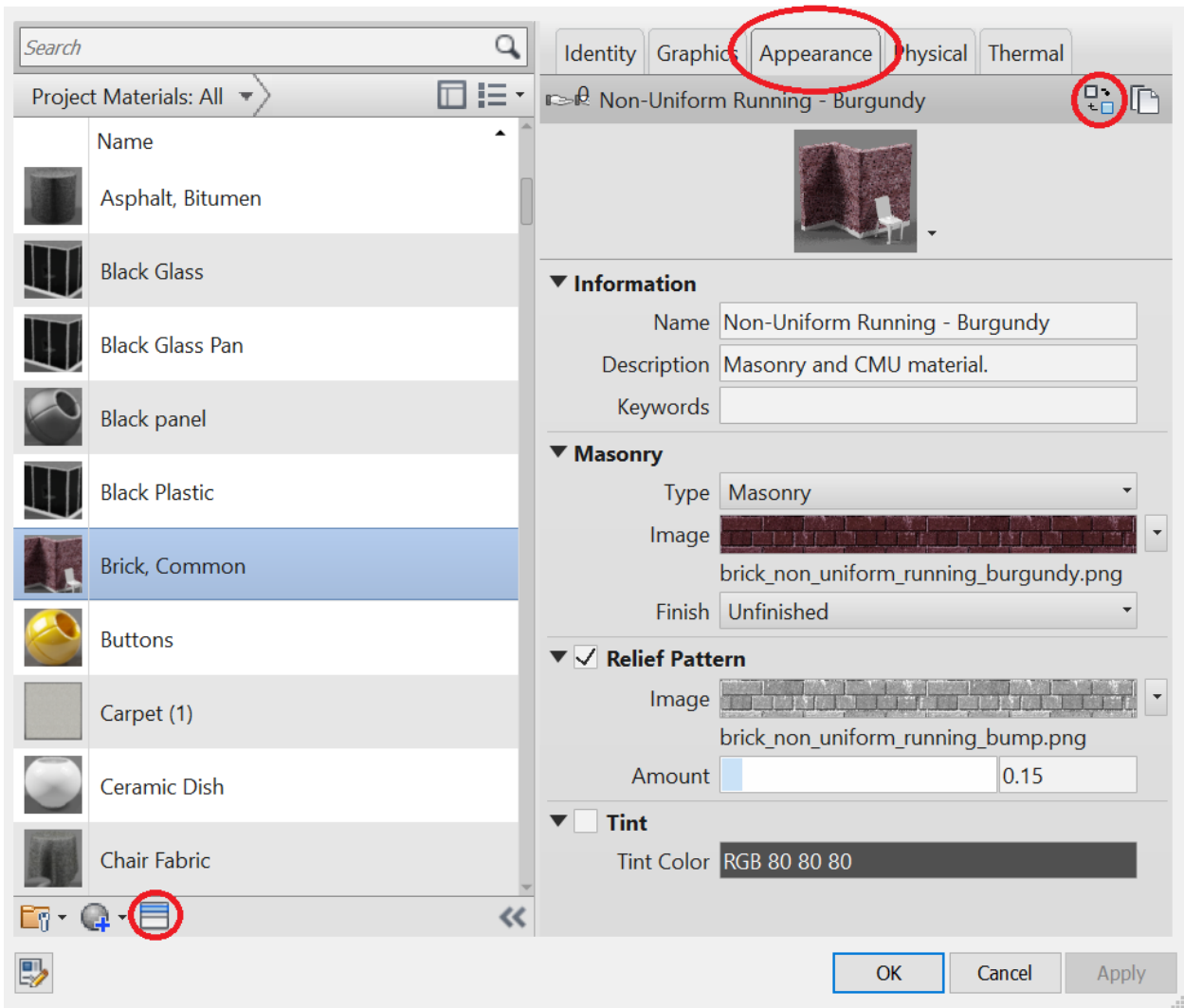
### Terminology

Term	Definition
Revit Material	<p>An element in Revit representing a material, made of a collection of property sets (graphics properties used in shaded/consistent color/hidden line views, rendering appearance information, structural properties and thermal properties). Some materials are only used for visual representation and do not have structural or thermal properties.</p> <p>API class name <b>Material</b> (namespace <b>Autodesk.Revit.DB</b>)</p>
Asset	<p>For Revit/Revit API users: The class representing a package of properties that represent something necessary for visual rendering. Assets in Revit may represent a collection of visual material's properties, an image used in material properties, lighting, background images, environments, etc.</p> <p>API class name <b>Asset</b> (namespace <b>Autodesk.Revit.DB.Visual</b>)</p>
Appearance Asset	<p>The particular instances of Assets representing visual material properties. The appearance asset is obtained from the AppearanceAssetElement via GetRenderingAsset().</p> <p>API class name <b>Asset</b> (its property <b>AssetType</b> equals <b>Appearance</b>)</p>
Appearance Asset Element	<p>An element that stores an appearance asset in the Revit model. In the Revit API, the Revit material provides access to this element via the Material.AppearanceAssetId property.</p> <p>API class name <b>AppearanceAssetElement</b> (namespace <b>Autodesk.Revit.DB</b>)</p>
Asset property	<p>One particular property of an asset. This will have a name used for identification, and a type representing the value. In the Revit API, the AssetProperty class is the base class for this.</p> <p>API class name <b>AssetProperty</b> (namespace <b>Autodesk.Revit.DB.Visual</b>)</p>

## Connection between Materials UI and API

Materials UI is accessed on the Manage Tab of the Ribbon, button Materials . Pressing this button will open the Material Browser dialog (see below). Left side of the dialog contains the list of Materials, and right side of the dialog has up to five tabs (each tab corresponds to an asset). There are diverse types of assets describing material properties. In this presentation we focus on the appearance properties of Materials (Appearance Tab).

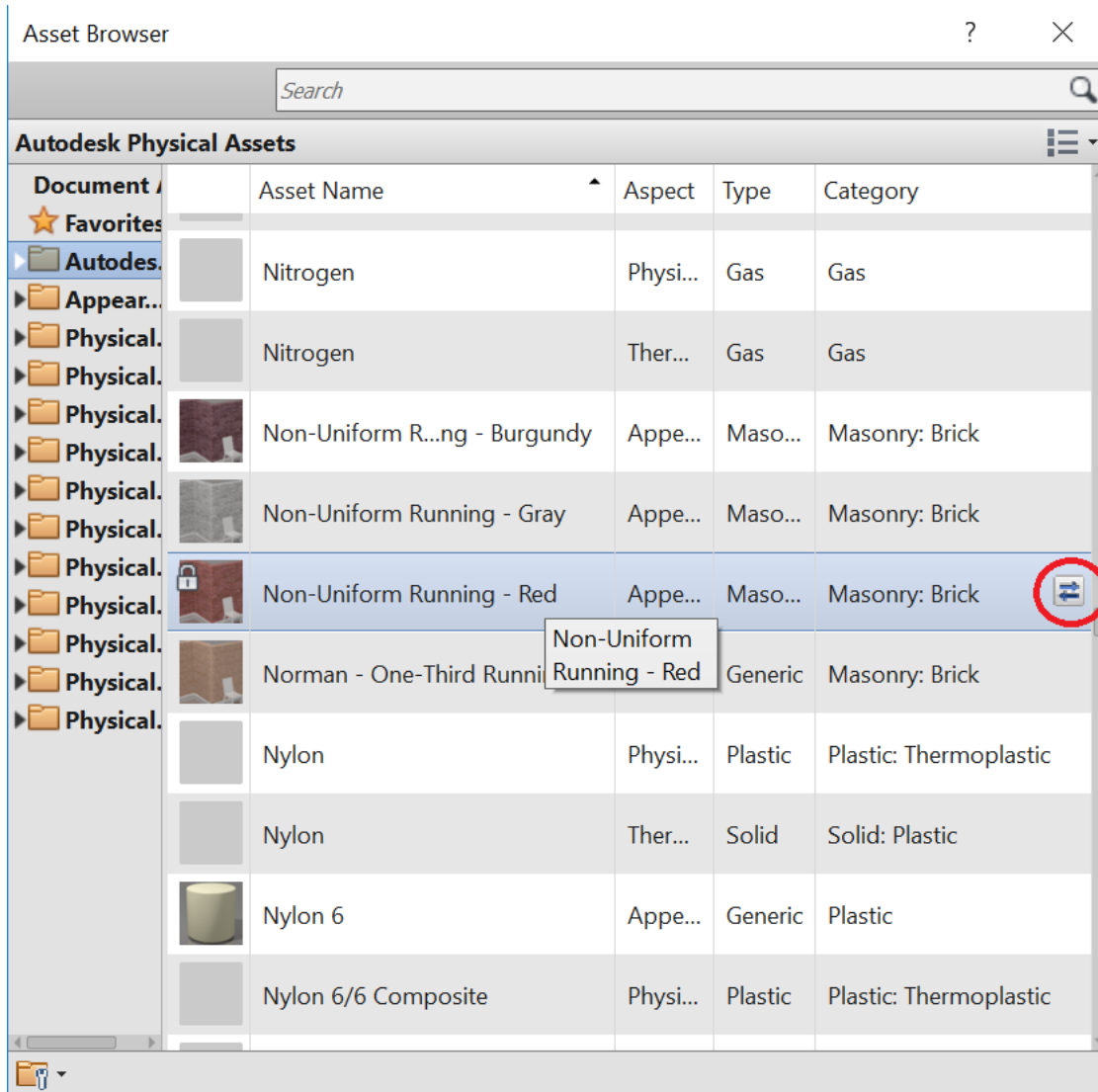
Material Browser - Brick, Common



Material Browser.

Please note \*.png names below the images: these texture files correspond to “connected assets” discussed further in this presentation.

It is possible to associate the selected Material with a different Appearance Asset. Pressing either one of the two circled buttons on the Material Browser dialog (above) will open the Asset Browser dialog (see below). Hovering mouse over any of the listed Assets will display the button (circled below): pressing this button will associate the selected Material with the selected Asset.



Asset Browser.

The new API discussed below covers the functionality of this Materials UI. It also enables editing automation of the Material appearance properties, including the possibility of third-party plugins extending current Revit functionality.

## Material API building blocks and relationship between classes

Material class has a property AppearanceAssetId.


### Material Members

[Material Class Methods Properties See Also Send Feedback](#)

The [Material](#) type exposes the following members.

▣ **Methods**

▣ **Properties**

Name	Description
 <a href="#">AppearanceAssetId</a>	The ElementId of the AppearanceAssetElement.


This id is used to obtain AppearanceAssetElement from Document.

### AppearanceAssetElement Members

[AppearanceAssetElement Class Methods Properties See Also Send Feedback](#)

The [AppearanceAssetElement](#) type exposes the following members.

▣ **Methods**

Name	Description
 <a href="#">GetRenderingAsset</a>	Gets the rendering asset for the appearance asset element.

Using method AppearanceAssetElement.GetRenderingAsset() we obtain Asset of the AssetType Appearance.

### Asset Class

[Members Example See Also Send Feedback](#)

▣ **Inheritance Hierarchy**

- [System.Object](#)
- [Autodesk.Revit.DB.Visual.AssetProperty](#)
- [Autodesk.Revit.DB.Visual.AssetProperties](#)
- Autodesk.Revit.DB.Visual.Asset**

Asset in turn contains an array of properties: these properties are the AssetProperty derived classes (see below). These properties can be accessed using either FindByName() method or operator []. The available names of the properties are described by material schemas described further in this document. Each Asset belongs to a particular schema that defines available properties.

Property names are just strings that can be obtained from schema documentation (see for example “generic\_diffuse”, “unified\_bitmap”, etc. in the code examples below. Alternatively, for API user convenience, static classes for each Schema were introduced with static properties corresponding to all available asset property names. See details below in the section “Schemas and property names”.

## AssetProperty Class

[Members](#) [See Also](#) [Send Feedback](#)

### Inheritance Hierarchy

[System.Object](#)

#### Autodesk.Revit.DB.Visual.AssetProperty

[Autodesk.Revit.DB.Visual.AssetProperty](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyBoolean](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDistance](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDouble](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDoubleArray2d](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDoubleArray3d](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDoubleArray4d](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyDoubleMatrix44](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyEnum](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyFloat](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyFloatArray](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyInt64](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyInteger](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyList](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyReference](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyString](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyTime](#)  
[Autodesk.Revit.DB.Visual.AssetPropertyUInt64](#)

Each AssetProperty contained in an appearance Asset can have one (or potentially more) connected Asset. In all current schemas included with the materials, asset properties will have at most one connected Asset, but this will not necessarily be the case in the future.

## AssetProperty Members

[AssetProperty Class](#) [Methods](#) [Properties](#) [See Also](#) [Send Feedback](#)

### Methods

	Name	Description
	<a href="#">AddConnectedAsset</a>	Adds a new connected asset attached to this asset property, if it allows it.
	<a href="#">AddCopyAsConnectedAsset</a>	Makes a copy of the asset and connects it to this property.
	<a href="#">GetSingleConnectedAsset</a>	Gets the single connected asset attached to this asset property, if it exists.
	<a href="#">RemoveConnectedAsset</a>	Removes the connected asset attached to this asset property if any.

In order to edit properties of an Asset, it must be placed in AppearanceEditScope. Currently only one Asset can be placed in the edit scope. Details on usage of the edit scope will be discussed further in the section “New editing capabilities in Materials API”.

## AppearanceAssetEditScope Members

[AppearanceAssetEditScope Class](#) [Constructors](#) [Methods](#) [Properties](#) [See Also](#) [Send Feedback](#)

The [AppearanceAssetEditScope](#) type exposes the following members.

### Constructors

	Name	Description
	<a href="#">AppearanceAssetEditScope</a>	Constructs a new instance of an AppearanceAssetEditScope.

### Methods

	Name	Description
	<a href="#">Cancel</a>	Cancels the edit scope.
	<a href="#">Commit</a>	Finishes the edit scope.
	<a href="#">Dispose</a>	Releases all resources used by the <a href="#">AppearanceAssetEditScope</a>
	<a href="#">Start</a>	Starts the edit scope.

## New editing capabilities in Materials API

### Editing the properties of an Appearance Asset

New Revit API capabilities have been introduced to edit the properties contained in an appearance asset of a material. These properties appear in the Appearance tab of the Materials dialog and govern the appearance of the material in realistic views and rendering.

Editing the properties in an appearance Asset requires establishment of an edit scope. The new class

- Autodesk.Revit.DB.Visual.AppearanceAssetEditScope

allows an application to create and maintain an editing session for an appearance asset. The scope provides access to an editable Asset object whose properties may be changed. Once all of the desired changes have been made to the asset's properties, the edit scope should be committed, which causes the changes to be sent back into the document. (This is the only part of the process when a transaction must be opened).

The new class has the following methods:

- AppearanceAssetEditScope.Start() – Starts the edit scope for the asset contained in a particular AppearanceAssetElement. The editable Asset is returned from this method.
- AppearanceAssetEditScope.Commit() – Finishes the edit scope: all changes made during the edit scope will be committed. Provides an option to forces the update of all open views.
- AppearanceAssetEditScope.Cancel() – Cancels the edit scope and discards any changes.

### Editing AssetProperty values

The following properties are now writeable from within an AppearanceAssetEditScope, to support modification of an asset property's stored value:

- AssetPropertyString.Value
- AssetPropertyBoolean.Value
- AssetPropertyString.Value
- AssetPropertyInteger.Value
- AssetPropertyDouble.Value
- AssetPropertyFloat.Value
- AssetPropertyEnum.Value
- AssetPropertyDistance.Value



Please note that AssetPropertyDistance is different from all other places in Revit API that specify length: in most of the cases the length is specified in feet. Here AssetPropertyDistance has its property DisplayUnitType that specifies the units used for Value.

Also, the following new methods have been added to support modification of property values:

- AssetPropertyDouble3.SetValueAsXYZ()
- AssetPropertyDouble4.SetValueAsDoubles()
- AssetPropertyDouble4.SetValueAsColor()



AssetPropertyList now has new methods to allow changes to the members of the list:

- AssetPropertyList.AddNewAssetPropertyDouble()
- AssetPropertyList.InsertNewAssetPropertyDouble()
- AssetPropertyList.AddNewAssetAsColor()
- AssetPropertyList.InsertNewAssetAsColor()
- AssetPropertyList.AddNewAssetPropertyInteger()
- AssetPropertyList.InsertNewAssetPropertyInteger()
- AssetPropertyList.RemoveAssetProperty().

Let us consider a code sample that shows how a particular AssetProperty can be modified.


### Change paint color

```
public void SetDiffuseColor(Material material, Color color)
{
    ElementId appearanceAssetId = material.AppearanceAssetId;
    AppearanceAssetElement assetElem
        = material.Document.GetElement(appearanceAssetId) as AppearanceAssetElement;

    // A transaction is necessary.
    // Multiple changes to the same asset can be made in one transaction if required.
    using(Transaction t = new Transaction(assetElem.Document, "Change material color"))
    {
        t.Start();
        using(AppearanceAssetEditScope editScope
            = new AppearanceAssetEditScope(assetElem.Document))
        {
            Asset editableAsset = editScope.Start(assetElem.Id);
            // returns an editable copy of the appearance asset
            // unlike AppearanceAssetElement.GetRenderingAsset(), the edit scope
            // sets up the asset for editing using the classes and utilities from
            // MaterialsManager. If the Asset was not obtained from an edit scope,
            // the setter operations for asset properties will throw.

            // Try to change values
            AssetPropertyDoubleArray4d genericDiffuseProperty
                = editableAsset["generic_diffuse"] as AssetPropertyDoubleArray4d;
            genericDiffuseProperty.SetValueAsColor(color);

            // Commit the entire edit scope.
            // If the appearance asset element is used in one or more materials,
            // they will be updated to match any changes made.
            editScope.Commit(true);
        }
        t.Commit();
    }
}
```

 Please note that diffuse color property is special: it is ignored if it has a connected asset specifying texture to be used. Only if the diffuse color has no connected Asset it will govern the Material color.

## Connected Assets

Connected assets are associated to properties in appearance assets, and represent subordinate objects encapsulating a collection of related properties. One example of a connected asset is the "Unified Bitmap" representing an image and its mapping parameters and values. AssetProperty offers new methods to provide the ability to modify, add or delete the asset connected to a property:

- AssetProperty.GetSingleConnectedAsset() – Gets the single connected asset of this property.
- AssetProperty.RemoveConnectedAsset() – Removes the single connected asset of this property.
- AssetProperty.AddConnectedAsset (String schemaId) – Create a new default asset of schema type and connects it to this property.
- AssetProperty.AddCopyAsConnectedAsset() – Connects the property to a copy of the asset.



Please note that there is a special property AssetPropertyReference. It is used only to have a connected Asset.

Let us consider a code sample that shows how a connected Asset can be edited.

### Edit connected asset properties

```
public void SetBumpmapBitmap(Material material, String bumpmapImagePath)
{
    ElementId appearanceAssetId = material.AppearanceAssetId;
    AppearanceAssetElement assetElem
        = material.Document.GetElement(appearanceAssetId) as AppearanceAssetElement;

    using(Transaction t
        = new Transaction(material.Document, "Change material bumpmap bitmap"))
    {
        t.Start();

        using(AppearanceAssetEditScope editScope
            = new AppearanceAssetEditScope(assetElem.Document))
        {
            // returns an editable copy of the appearance asset
            Asset editableAsset = editScope.Start(assetElem.Id);

            AssetProperty bumpMapProperty = editableAsset["generic_bump_map"];

            // Find the connected asset (with a shortcut to get the only one)
            Asset connectedAsset = bumpMapProperty.GetSingleConnectedAsset();
            if (connectedAsset == null)
            {
                // Add a new default connected asset
                bumpMapProperty.AddConnectedAsset("UnifiedBitmap");
                connectedAsset = bumpMapProperty.GetSingleConnectedAsset();
            }
            if (connectedAsset != null)
            {
                // Find the target asset property
                AssetPropertyString bumpmapBitmapProperty
                    = connectedAsset["unifiedbitmap_Bitmap"] as AssetPropertyString;
```

```
        if (bumpmapBitmapProperty.IsValidValue(bumpmapImageFilepath))
            bumpmapBitmapProperty.Value = bumpmapImageFilepath;
    }
    editScope.Commit(true);
}
t.Commit();
}
```

## Validation

Inputs to change the value of asset properties are validated against the requirements of the associated schema. The new methods:

- AssetPropertyString.IsValidValue(String)
- AssetPropertyInteger.IsValidValue (int)
- AssetPropertyEnum.IsValidValue (int)
- AssetPropertyDouble.IsValidValue (double)
- AssetPropertyFloat.IsValidValue (float)
- AssetPropertyDistance.IsValidValue (double)
- AssetPropertyDouble3.IsValidValue (XYZ)
- AssetPropertyDouble4.IsValidValue (IList<double>)
- AssetPropertyDouble4.IsValidValue (Color)

identify if the input value is a valid value that can be set to the given asset property.

The new method:

- AssetProperty.IsEditable()

identifies the AssetProperty can currently be edited.

## Utilities

The new method:

- Application.GetAssets(AssetType)

returns a list of assets available to the session.

The new method:

- AppearanceAssetElement.Duplicate()

creates a copy of an appearance asset element and the asset contained by it.

The new operator:

- Asset.operator[ ]

accesses a particular AssetProperty associated to the given asset.

A code sample below shows how a Material can be duplicated and then modified:

### Duplicate Material

```
public void DuplicateAndModifyMaterial(Material material)
{
    ElementId appearanceAssetId = material.AppearanceAssetId;
    AppearanceAssetElement assetElem
        = material.Document.GetElement(appearanceAssetId) as AppearanceAssetElement;
    ElementId duplicateAssetElementId = ElementId.InvalidElementId;
    using (Transaction t
        = new Transaction(material.Document, "Duplicate Red Carpet Material"))
    {
        t.Start();
        // Duplicate the material
        Material duplicateMaterial = material.Duplicate("Blue Carpet");
        // Duplicate the appearance asset element and the rendering asset in it
        AppearanceAssetElement duplicateAssetElement
            = assetElem.Duplicate("Blue Carpet");
        // Assign the asset element to the material
        duplicateMaterial.AppearanceAssetId = duplicateAssetElement.Id;
        duplicateAssetElementId = duplicateAssetElement.Id;
        t.Commit();
    }

    // Make changes to the duplicate asset
    using (Transaction t2
        = new Transaction(material.Document, "Change blue carpet material"))
    {
        t2.Start();
        using (AppearanceAssetEditScope editScope
            = new AppearanceAssetEditScope(assetElem.Document))
        {
            // returns an editable copy of the appearance asset
            Asset editableAsset = editScope.Start(duplicateAssetElementId);
            // Description
            AssetPropertyString descriptionProperty =
                editableAsset["description"] as AssetPropertyString;
            descriptionProperty.Value = "blue carpet";
            // Diffuse image
            AssetPropertyDoubleArray4d genericDiffuseProperty
                = editableAsset["generic_diffuse"] as AssetPropertyDoubleArray4d;
            genericDiffuseProperty.SetValueAsColor(new Color(0x00, 0x00, 0xFF));
            Asset connectedAsset = genericDiffuseProperty.GetSingleConnectedAsset();
            AssetPropertyString bitmapProperty
                = connectedAsset["unifiedbitmap_Bitmap"] as AssetPropertyString;
            bitmapProperty.Value = "Finishes.Flooring.Carpet.4.png";
            editScope.Commit(true);
        }
        t2.Commit();
    }
}
```

Here is a code sample that duplicates an Asset and connects newly created copy to a property in other Material:

### Duplicate connected asset

```
public void CopyAndAddConnectedAsset(Material sourceMaterial, Material targetMaterial)
{
    Document doc = targetMaterial.Document;

    // Get the appearance asset of the source material
    ElementId otherAppearanceAssetId = sourceMaterial.AppearanceAssetId;
    AppearanceAssetElement otherAssetElem
        = doc.GetElement(otherAppearanceAssetId) as AppearanceAssetElement;
    Asset otherAsset = otherAssetElem.GetRenderingAsset();

    // Get the connected asset of the source material appearance asset
    // (that contains the bitmap)
    AssetProperty otherGenericDiffuseProperty = otherAsset[Generic.GenericDiffuse];
    Asset otherConnectedAsset = otherGenericDiffuseProperty.GetSingleConnectedAsset();

    using (Transaction t = new Transaction(doc, "Change a connected asset by a copy"))
    {
        t.Start();

        using (AppearanceAssetEditScope editScope = new AppearanceAssetEditScope(doc))
        {
            ElementId appearanceAssetId = targetMaterial.AppearanceAssetId;

            // Returns an editable copy of the appearance asset
            Asset editableAsset = editScope.Start(appearanceAssetId);
            AssetProperty genericDiffuseProperty = editableAsset[Generic.GenericDiffuse];

            // Find the connected asset (with a shortcut to get the only one)
            Asset genericDiffuseConnectedAsset
                = genericDiffuseProperty.GetSingleConnectedAsset();
            if (genericDiffuseConnectedAsset == null)
            {
                // Target material has only a color for "GenericDiffuse"
                // This will assign a bitmap to it instead
                // by making a copy of the asset and use it as connected asset
                genericDiffuseProperty.AddCopyAsConnectedAsset(otherConnectedAsset);
            }
            editScope.Commit(true);
        }
        t.Commit();
    }
}
```

## Schemas and property names

Appearance asset properties are aligned with specific schemas. Each schema contains necessary properties which define how the appearance of the material will be generated.

The new method:

- AssetProperty.IsValidSchemaIdentifier(String schemaName)

identifies if the input name is a valid name for a supported schema.

To assist in creating code accessing and manipulating the properties of a given schema, predefined properties have been introduced to allow a compile-time reference to a property name without requiring you to transcribe it as a string in your code. These predefined property names are available in static classes named similar to the schema names, above, e.g. Autodesk.Revit.DB.Visual.Ceramic.









There is one SchemaCommon that describes set of properties present in all standard and advanced Materials:

### SchemaCommon Members

[SchemaCommon Class Properties](#) [See Also](#) [Send Feedback](#)

The [SchemaCommon](#) type exposes the following members.

#### Properties

	Name
 <b>S</b>	<a href="#">BaseSchema</a>
 <b>S</b>	<a href="#">Category</a>
 <b>S</b>	<a href="#">Description</a>
 <b>S</b>	<a href="#">Hidden</a>
 <b>S</b>	<a href="#">Keyword</a>
 <b>S</b>	<a href="#">Thumbnail</a>
 <b>S</b>	<a href="#">UIName</a>
 <b>S</b>	<a href="#">VersionGUID</a>

## **Standard Material Schemas**

There are 14 standard material schemas:

- Ceramic
- Concrete
- Generic
- Glazing
- Hardwood
- MasonryCMU
- Metal
- MetallicPaint
- Mirror
- PlasticVinyl
- SolidGlass
- Stone
- WallPaint
- Water

## **Advanced Material Schemas**

In addition, there are 5 schemas representing "advanced" materials - these may be encountered as a result of import from other Autodesk products:














- AdvancedLayered
- AdvancedMetal
- AdvancedOpaque
- AdvancedTransparent
- AdvancedWood.


## **Schemas for connected Assets**

Finally, there are 10 schemas used for the aspects of the connected assets:

- BumpMap
- Checker
- Gradient
- Marble
- Noise
- Speckle
- Tile
- UnifiedBitmap
- Wave
- Wood.

The most commonly used schema for connected Assets is UnifiedBitmap. Let's look at its properties:

UnifiedBitmap Members		
<a href="#">UnifiedBitmap Class Properties</a> <a href="#">See Also</a> <a href="#">Send Feedback</a>		
The <a href="#">UnifiedBitmap</a> type exposes the following members.		
<b>Properties</b>		
	Name	Description
	<a href="#">TextureLinkTextureTransforms</a>	The property labeled "Link texture Transforms" from the "UnifiedBitmap" schema.
	<a href="#">TextureOffsetLock</a>	The property labeled "Offset Lock" from the "UnifiedBitmap" schema.
	<a href="#">TextureRealWorldOffsetX</a>	The property labeled "Offset X" from the "UnifiedBitmap" schema.
	<a href="#">TextureRealWorldOffsetY</a>	The property labeled "Offset Y" from the "UnifiedBitmap" schema.
	<a href="#">TextureRealWorldScaleX</a>	The property labeled "Size X" from the "UnifiedBitmap" schema.
	<a href="#">TextureRealWorldScaleY</a>	The property labeled "Size Y" from the "UnifiedBitmap" schema.
	<a href="#">TextureScaleLock</a>	The property labeled "Scale Lock" from the "UnifiedBitmap" schema.
	<a href="#">TextureURepeat</a>	The property labeled "U Repeat" from the "UnifiedBitmap" schema.
	<a href="#">TextureVRepeat</a>	The property labeled "V Repeat" from the "UnifiedBitmap" schema.
	<a href="#">TextureWAngle</a>	The property labeled "Angle" from the "UnifiedBitmap" schema.
	<a href="#">UnifiedbitmapBitmap</a>	The property labeled "Source" from the "UnifiedBitmap" schema.
	<a href="#">UnifiedbitmapInvert</a>	The property labeled "Invert Image" from the "UnifiedBitmap" schema.
	<a href="#">UnifiedbitmapRGBAmount</a>	The property labeled "Brightness" from the "UnifiedBitmap" schema.

 Please note that UnifiedBitmap.UnifiedbitmapBitmap property contains the path to the image file location. This path is relative if the image file is located in default Material Library location or in one of the locations specified in Options/Rendering/Additional Render Appearance Paths. In all other cases, the path stored in UnifiedBitmap.UnifiedbitmapBitmap is absolute.

In order to better understand the contents of Appearance Asset, we include here a printout from an API macro that lists all properties present in the Appearance Asset of Material "Brick, Common". This Material is selected in UI of Material Browser at the beginning of this document. Note that the main Appearance Asset (of MasonryCMUSchema) has one Asset (of UnifiedBitmapSchema) connected to AssetProperty "masonrycmu\_color" and one more Asset (also of MasonryCMUSchema) connected to AssetProperty "masonrycmu\_pattern\_map". The first connected Asset points to brick\_non\_uniform\_running\_burgundy.png and the second connected Asset points to brick\_non\_uniform\_running\_bump.png:

```
name: AdvancedUIDefinition | type: AssetPropertyString | value: Mats/MasonryCMU/MasonryCMUAdvancedUI.xml
name: AssetLibID | type: AssetPropertyString | value: AD121259-C03E-4A1D-92D8-59A22B4807AD
name: assettype | type: AssetPropertyString | value: materialappearance
name: BaseSchema | type: AssetPropertyString | value: MasonryCMUSchema
name: category | type: AssetPropertyString | value: :Masonry/Brick:Default:Brick
name: common_Shared_Asset | type: AssetPropertyInteger | value: 1
name: common_Tint_color | type: AssetPropertyDoubleArray4d | value: R:80 G:80 B:80
name: common_Tint_toggle | type: AssetPropertyBoolean | value: False
name: description | type: AssetPropertyString | value: Masonry and CMU material.
name: ExchangeGUID | type: AssetPropertyString | value:
name: Hidden | type: AssetPropertyBoolean | value: False
name: ImplementationGeneric | type: AssetPropertyString | value: Mats/MasonryCMU/Generic.xml
name: ImplementationMentalRay | type: AssetPropertyString | value: Mats/MasonryCMU/MentalImage.xml
name: ImplementationOGS | type: AssetPropertyString | value: Mats/MasonryCMU/OGS.xml
name: ImplementationPreview | type: AssetPropertyString | value: Mats/MasonryCMU/PreviewColor.xml
name: keyword | type: AssetPropertyString | value:
name: localname | type: AssetPropertyString | value: Masonry
name: localtype | type: AssetPropertyString | value: Appearance
```



```

name: masonrycmu_ao_details | type: AssetPropertyBoolean | value: False
name: masonrycmu_ao_distance | type: AssetPropertyDouble | value: 4
name: masonrycmu_ao_on | type: AssetPropertyBoolean | value: False
name: masonrycmu_ao_samples | type: AssetPropertyInteger | value: 16
name: masonrycmu_application | type: AssetPropertyInteger | value: 2
name: masonrycmu_color | type: String | value: R:120 G:120 B:120
- connected (1): name: AdvancedUIDefinition | type: AssetPropertyString | value: Maps/UnifiedBitmap/UnifiedBitmapAdvancedUI.xml
- connected (1): name: AssetLibID | type: AssetPropertyString | value: AD121259-C03E-4A1D-92D8-59A22B4807AD
- connected (1): name: BaseSchema | type: AssetPropertyString | value: UnifiedBitmapSchema
- connected (1): name: ExchangeGUID | type: AssetPropertyString | value:
- connected (1): name: Hidden | type: AssetPropertyBoolean | value: False
- connected (1): name: ImplementationGeneric | type: AssetPropertyString | value:
- connected (1): name: ImplementationMentalRay | type: AssetPropertyString | value: Maps/UnifiedBitmap/MentalImage.xml
- connected (1): name: ImplementationOGS | type: AssetPropertyString | value: Maps/UnifiedBitmap/OGS.xml
- connected (1): name: ImplementationPreview | type: AssetPropertyString | value:
- connected (1): name: SchemaVersion | type: AssetPropertyInteger | value: 5
- connected (1): name: UIDefinition | type: AssetPropertyString | value: Maps/UnifiedBitmap/UnifiedBitmapUI.xml
- connected (1): name: UIName | type: AssetPropertyString | value: Masonry-002_masonrycmu_color
- connected (1): name: VersionGUID | type: AssetPropertyString | value: Masonry-002_masonrycmu_color
- connected (1): name: assettype | type: AssetPropertyString | value: texture
- connected (1): name: category | type: AssetPropertyString | value:
- connected (1): name: common_Shared_Asset | type: AssetPropertyInteger | value: 0
- connected (1): name: common_Tint_color | type: AssetPropertyDoubleArray4d | value: R:80 G:80 B:80
- connected (1): name: common_Tint_toggle | type: AssetPropertyBoolean | value: False
- connected (1): name: description | type: AssetPropertyString | value: Unified Bitmap.
- connected (1): name: keyword | type: AssetPropertyString | value: :masonry:brick:burgundy:non-uniform:materials
- connected (1): name: localname | type: AssetPropertyString | value: Bitmap Texture
- connected (1): name: localtype | type: AssetPropertyString | value: Texture
- connected (1): name: revision | type: AssetPropertyInteger | value: 1
- connected (1): name: swatch | type: AssetPropertyString | value:
- connected (1): name: texture_LinkTextureTransforms | type: AssetPropertyBoolean | value: False
- connected (1): name: texture_MapChannel | type: AssetPropertyInteger | value: 1
- connected (1): name: texture_MapChannel_ID_Advanced | type: AssetPropertyInteger | value: 1
- connected (1): name: texture_MapChannel_UVWSource_Advanced | type: AssetPropertyInteger | value: 0
- connected (1): name: texture_OffsetLock | type: AssetPropertyBoolean | value: False
- connected (1): name: texture_RealWorldOffsetX | type: AssetPropertyDistance | value: 0' - 0"
- connected (1): name: texture_RealWorldOffsetY | type: AssetPropertyDistance | value: 0' - 0"
- connected (1): name: texture_RealWorldScaleX | type: AssetPropertyDistance | value: 3' - 4"
- connected (1): name: texture_RealWorldScaleY | type: AssetPropertyDistance | value: 3' - 10 3/4"
- connected (1): name: texture_ScaleLock | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_UOffset | type: AssetPropertyDouble | value: 0
- connected (1): name: texture_URepeat | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_UScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_UVScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_VOffset | type: AssetPropertyDouble | value: 0
- connected (1): name: texture_VRepeat | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_VScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_WAngle | type: AssetPropertyDouble | value: 0
- connected (1): name: thumbnail | type: AssetPropertyString | value: Maps\UnifiedBitmap\UnifiedBitmap.png
- connected (1): name: unifiedbitmap_Bitmap | type: AssetPropertyString | value: 1\mats\brick_non_uniform_running_burgundy.png
- connected (1): name: unifiedbitmap_Bitmap_urn | type: AssetPropertyString | value:
- connected (1): name: unifiedbitmap_Blur | type: AssetPropertyDouble | value: 0.01
- connected (1): name: unifiedbitmap_Blur_Offset | type: AssetPropertyDouble | value: 0
- connected (1): name: unifiedbitmap_Filtering | type: AssetPropertyInteger | value: 0
- connected (1): name: unifiedbitmap_Invert | type: AssetPropertyBoolean | value: False
- connected (1): name: unifiedbitmap_RGBAmount | type: AssetPropertyDouble | value: 0.865
- connected (1): name: version | type: AssetPropertyInteger | value: 2
name: masonrycmu_color_by_object | type: AssetPropertyBoolean | value: False
name: masonrycmu_pattern | type: AssetPropertyInteger | value: 1
name: masonrycmu_pattern_height | type: AssetPropertyDouble | value: 0.15
name: masonrycmu_pattern_map | type: String | value: Reference
- connected (1): name: AdvancedUIDefinition | type: AssetPropertyString | value: Maps/UnifiedBitmap/UnifiedBitmapAdvancedUI.xml
- connected (1): name: AssetLibID | type: AssetPropertyString | value: AD121259-C03E-4A1D-92D8-59A22B4807AD
- connected (1): name: BaseSchema | type: AssetPropertyString | value: UnifiedBitmapSchema
- connected (1): name: ExchangeGUID | type: AssetPropertyString | value:

```

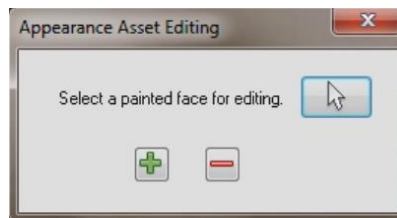
```

- connected (1): name: Hidden | type: AssetPropertyBoolean | value: False
- connected (1): name: ImplementationGeneric | type: AssetPropertyString | value:
- connected (1): name: ImplementationMentalRay | type: AssetPropertyString | value: Maps/UnifiedBitmap/MentalImage.xml
- connected (1): name: ImplementationOGS | type: AssetPropertyString | value: Maps/UnifiedBitmap/OGS.xml
- connected (1): name: ImplementationPreview | type: AssetPropertyString | value:
- connected (1): name: SchemaVersion | type: AssetPropertyInteger | value: 5
- connected (1): name: UIDefinition | type: AssetPropertyString | value: Maps/UnifiedBitmap/UnifiedBitmapUI.xml
- connected (1): name: UIName | type: AssetPropertyString | value: Masonry-002_masonrycmu_pattern_map
- connected (1): name: VersionGUID | type: AssetPropertyString | value: Masonry-002_masonrycmu_pattern_map
- connected (1): name: assettype | type: AssetPropertyString | value: texture
- connected (1): name: category | type: AssetPropertyString | value:
- connected (1): name: common_Shared_Asset | type: AssetPropertyInteger | value: 0
- connected (1): name: common_Tint_color | type: AssetPropertyDoubleArray4d | value: R:80 G:80 B:80
- connected (1): name: common_Tint_toggle | type: AssetPropertyBoolean | value: False
- connected (1): name: description | type: AssetPropertyString | value: Unified Bitmap.
- connected (1): name: keyword | type: AssetPropertyString | value: :maps:misc
- connected (1): name: localname | type: AssetPropertyString | value: Bitmap Texture
- connected (1): name: localtype | type: AssetPropertyString | value: Texture
- connected (1): name: revision | type: AssetPropertyInteger | value: 1
- connected (1): name: swatch | type: AssetPropertyString | value:
- connected (1): name: texture_LinkTextureTransforms | type: AssetPropertyBoolean | value: False
- connected (1): name: texture_MapChannel | type: AssetPropertyInteger | value: 1
- connected (1): name: texture_MapChannel_ID_Advanced | type: AssetPropertyInteger | value: 1
- connected (1): name: texture_MapChannel_UVWSource_Advanced | type: AssetPropertyInteger | value: 0
- connected (1): name: texture_OffsetLock | type: AssetPropertyBoolean | value: False
- connected (1): name: texture_RealWorldOffsetX | type: AssetPropertyDistance | value: 0' - 0"
- connected (1): name: texture_RealWorldOffsetY | type: AssetPropertyDistance | value: 0' - 0"
- connected (1): name: texture_RealWorldScaleX | type: AssetPropertyDistance | value: 3' - 4"
- connected (1): name: texture_RealWorldScaleY | type: AssetPropertyDistance | value: 3' - 10 3/4"
- connected (1): name: texture_ScaleLock | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_UOffset | type: AssetPropertyDouble | value: 0
- connected (1): name: texture_URRepeat | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_UScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_UVScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_VOffset | type: AssetPropertyDouble | value: 0
- connected (1): name: texture_VRepeat | type: AssetPropertyBoolean | value: True
- connected (1): name: texture_VScale | type: AssetPropertyDouble | value: 1
- connected (1): name: texture_WAngle | type: AssetPropertyDouble | value: 0
- connected (1): name: thumbnail | type: AssetPropertyString | value: Maps\UnifiedBitmap\UnifiedBitmap.png
- connected (1): name: unifiedbitmap_Bitmap | type: AssetPropertyString | value: 1\mats\brick_non_uniform_running_bump.png
- connected (1): name: unifiedbitmap_Bitmap_urn | type: AssetPropertyString | value:
- connected (1): name: unifiedbitmap_Blur | type: AssetPropertyDouble | value: 0.01
- connected (1): name: unifiedbitmap_Blur_Offset | type: AssetPropertyDouble | value: 0
- connected (1): name: unifiedbitmap_Filtering | type: AssetPropertyInteger | value: 0
- connected (1): name: unifiedbitmap_Invert | type: AssetPropertyBoolean | value: False
- connected (1): name: unifiedbitmap_RGBAmount | type: AssetPropertyDouble | value: 0.865
- connected (1): name: version | type: AssetPropertyInteger | value: 2
name: masonrycmu_refl_depth | type: AssetPropertyInteger | value: 0
name: masonrycmu_roundcorners_allow_different_materials | type: AssetPropertyBoolean | value: False
name: masonrycmu_roundcorners_on | type: AssetPropertyBoolean | value: False
name: masonrycmu_roundcorners_radius | type: AssetPropertyDouble | value: 0.25
name: masonrycmu_type | type: AssetPropertyInteger | value: 1
name: mode | type: AssetPropertyInteger | value: 4
name: PatternOffset | type: AssetPropertyDoubleArray2d | value: 0,0,
name: reflection_glossy_samples | type: AssetPropertyInteger | value: 32
name: revision | type: AssetPropertyInteger | value: 1
name: SchemaVersion | type: AssetPropertyInteger | value: 4
name: swatch | type: AssetPropertyString | value: Swatch-Wall
name: thumbnail | type: AssetPropertyString | value: C:/Temp/Material/Thumbnails_PID_2ec0/56055580.png
name: UIDefinition | type: AssetPropertyString | value: Mats/MasonryCMU/MasonryCMUUI.xml
name: UIName | type: AssetPropertyString | value: Non-Uniform Running - Burgundy
name: version | type: AssetPropertyInteger | value: 2
name: VersionGUID | type: AssetPropertyString | value: Masonry-002

```

## SDK sample plug-in AppearanceAssetEditing

This sample demonstrates basic usage of the AppearanceAssetEditScope and AssetProperty classes to change the value of an asset property in a given material. The sample brings up a modeless dialog with two buttons: “+” and “-“. The dialog instructs users to select a painted face in the model. Once a painted face is selected and the material used to paint the face supports Tint Color property, the buttons are enabled and allow users to change the value of the Tint Color property. We do not discuss here the code related to the dialog and selection. It is interesting to look at the parts of the code relevant to the editing of the Appearance Asset.



### Enable tint color property if supported

Two methods below determine if the Tint Color property is present in the Appearance Asset of the Material used to paint the selected face. If it is present but not enabled, it is programmatically enabled and set to default value:

```
private bool SupportTintColor()
{
    if (this.m_currentAppearanceAssetElementId == ElementId.InvalidElementId)
        return false;

    AppearanceAssetElement assetElem
        = m_document.GetElement(this.m_currentAppearanceAssetElementId)
            as AppearanceAssetElement;
    if (assetElem == null)
        return false;

    Asset asset = assetElem.GetRenderingAsset();
    AssetPropertyDoubleArray4d tintColorProp
        = asset["common_Tint_color"] as AssetPropertyDoubleArray4d;
    if (tintColorProp == null)
        return false;

    // If the material supports tint but it is not enabled,
    // it will be enabled first with a value (255 255 255)
    AssetPropertyBoolean tintToggleProp
        = asset["common_Tint_toggle"] as AssetPropertyBoolean;
    if ((tintToggleProp != null) && !(tintToggleProp.Value))
    {
        EnableTintColor();
    }

    return true;
}
```

```
private void EnableTintColor()
{
    using (Transaction transaction = new Transaction(m_document, "Enable tint color"))
    {
        transaction.Start();

        using (AppearanceAssetEditScope editScope
            = new AppearanceAssetEditScope(m_document))
        {
            Asset editableAsset = editScope.Start(m_currentAppearanceAssetElementId);

            // If the material supports tint but it is not enabled,
            // it will be enabled first with a value (255 255 255)
            AssetPropertyBoolean tintToggleProp
                = editableAsset["common_Tint_toggle"] as AssetPropertyBoolean;
            AssetPropertyDoubleArray4d tintColorProp
                = editableAsset["common_Tint_color"] as AssetPropertyDoubleArray4d;

            tintToggleProp.Value = true;
            tintColorProp.SetValueAsColor(new Color(255, 255, 255));

            editScope.Commit(true);
        }
        transaction.Commit();
    }
}
```

## Edit tint color property

Two methods below do the actual editing and make Tint Color of the Material lighter or darker based on the button pressed. Here lighter or darker choice is just a bool input argument.

```
internal void ModifySelectedMaterial(String text, bool lighter)
{
    // Since we'll modify the document, we need a transaction
    // It's best if a transaction is scoped by a 'using' block
    using (Transaction trans = new Transaction(m_document))
    {
        // The name of the transaction was given as an argument
        if (trans.Start(text) == TransactionStatus.Started)
        {
            // apply the requested operation to every door
            EditMaterialTintColorProperty(lighter);

            trans.Commit();
        }
    }
}
```

```
internal void EditMaterialTintColorProperty(bool lighter)
{
    using (AppearanceAssetEditScope editScope
          = new AppearanceAssetEditScope(m_document))
    {
        Asset editableAsset = editScope.Start(m_currentAppearanceAssetElementId);
        AssetPropertyDoubleArray4d metalColorProp
            = editableAsset["common_Tint_color"] as AssetPropertyDoubleArray4d;
        Color color = metalColorProp.GetValueAsColor();
        byte red = color.Red;
        byte green = color.Green;
        byte blue = color.Blue;
        // Increment factor (value related to 255)
        int factor = 25;
        if (lighter)
        {
            red = (byte)LimitValue(red + factor);
            green = (byte)LimitValue(green + factor);
            blue = (byte)LimitValue(blue + factor);
        }
        else
        {
            red = (byte)LimitValue(red - factor);
            green = (byte)LimitValue(green - factor);
            blue = (byte)LimitValue(blue - factor);
        }
        if (metalColorProp.IsValidValue(color))
            metalColorProp.SetValueAsColor(new Color(red, green, blue));

        editScope.Commit(true);
    }
}
```