Discovery Extensibility and Add-ins Developer Guide

Ansys, Inc.
Southpointe
2600 ANSYS Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
(T) 724-746-3304
(F) 724-514-9494

Release 2024 R1
January 2024

Contents

1: Introduction

Purpose

This document is intended to provide an overview of the Discovery API and its fundamental concepts.

Programming Language

Although the Discovery API can be called from any .NET programming language (for example: C#, F#, C++/CLI, Visual Basic .NET, IronPython), all examples in this document are in C#.

Conventions

Words in **bold** indicate types or members in the API.

2: Creating Add-in Applications

The Discovery Application Programming Interface (API) Developer Guide contains documentation and samples that allow you to create add-in applications that extend the functionality of Discovery. An add-in is a managed code DLL that uses the Discovery API. The API version referred to in this document is API.V22.

This document contains information on the following:

- The basic infrastructure for Add-in customization.
- How to add custom elements to the User Interface.
- An overview of the geometry architecture.

2.1. Getting Started

The Discovery extensibility shares the same architecture as the SpaceClaim Add-in based customization. As a result of this design, many of the extensibility related APIs are coming from the SpaceClaim.Api.V22 dll. The physics related Discovery APIs, as well as some UI extensibility APIs that are specific to the Discovery product, are found in the Discovery.Api.V22.dll. There is no forward compatibility between SpaceClaim and Discovery Add-in, but the underlying architecture is the same: an Add-in that uses the SpaceClaim APIs to customize the User Interface will typically need some tweaks and adjustments to account for the difference in the User Interface between the two products. On the other hand, the geometry APIs used in a SpaceClaim add-in are fully functional in Discovery.

A Discovery add-in is a .NET class library assembly that implements at a minimum the IExtensibility interface. This interface provides the entry point for the add-in and allows the add-in to perform any initialization.

Add-ins that customize the Discovery Ribbon will also need to implement the ICommandExtensibility and IRibbonExtensibility interfaces.

Add-in Startup

During startup, Discovery will perform the following steps to load and initialize add-ins:

1. Locates all XML manifest files that describe available add-ins.
2. Uses the assembly and type name specified in the manifest file to create the add-in object.
3. Verifies that the add-in object implements the `IExtensibility` interface, and then calls `IExtensiblity.Connect`. The add-in performs any internal initialization at this time.
4. Checks to see if the add-in implements the optional interfaces `ICommandExtensibility` and `IRibbonExtensibility`.

Add-in Closedown

When Discovery is closed, add-in’s `IExtensiblity.Disconnect` method is called.

2.2. Creating a C# Add-in

The steps below describe how to create a Discovery C# add-in.

Creating the Add-In Visual Studio Project

1. In Microsoft Visual Studio, create a new C# Class Library project.
2. Add a reference to the API assemblies. Use the Add Reference command in Visual Studio, go to the Browse tab, and locate the API assemblies.
   - For APIs shared with SpaceClaim in the SpaceClaim.Api.V22 folder inside the Discovery installation folder.
   - For Discovery specific APIs in the Discovery.Api.V22 folder inside the Discovery installation folder.

Add-in Initialization

1. The class that serves as the entry point for the add-in must inherit from AddIn and implement the IExtensibility interface.
2. Perform any internal add-in initialization in the Connect method.
3. Perform any internal add-in cleanup in the Disconnect method.

Example code

```
using SpaceClaim.Api.V22;

using SpaceClaim.Api.V22.Extensibility;
namespace YourCompany.Example {
    public class SampleAddIn : AddIn, IExtensibility {
        #region IExtensibility Members
        public bool Connect() {
            // perform any initialization for your add-in here
            return true;
        }

        public void Disconnect() {
            // perform any cleanup for your add-in here
        }
        #endregion
    }
}
```

2.3. Creating Manifest Files

Discovery uses XML `manifest files` to describe add-ins. The manifest file provides two types of information:

- Metadata (name, description). This is presented to the user in the Discovery Options dialog, even for disabled add-ins, which don't get loaded.
- Execution data (assembly, type name, hosting option). This is used to load the add-in when Discovery starts up.

Discovery of Add-ins

Discovery will search for manifest files in all of the following areas:

- In folders beneath the `add-ins` folder: `<CommonApplicationData>\Discovery\AddIns`.
  - The default location on the latest version of Windows is: "`C:\ProgramData\Discovery\AddIns`".
- In folders beneath the AddIns folder inside the Discovery installation folder.
- In the `add-ins` registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\Discovery\AddIns`.

- A string value (REG_SZ or REG_EXPAND_SZ) may be added to this registry key to specify the path of a single manifest filename or the path of a folder where manifest files are located.
- Many of these string values can be added to the `AddIns` registry key.
- The name of each string value is not significant, but it must be unique within the **AddIns** registry key. Therefore a name such as "`BeachSoft.SurfingAddIn`" is recommended.

• The manifest file specified by the `/AddInManifestFile` command line switch supplied when Discovery is run.

  - `Discovery.exe /AddInManifestFile="<path of manifest file>"`.

Notes:

1. Putting the manifest file in the `ProgramData\Discovery\Addins` directory is the approach to follow when an Add-in should only be visible for a single user.
2. Putting the manifest file in the Addins folder inside the Discovery installation folder typically requires Administrative privileges and should be done for add-ins that are installed as part of Discovery.
3. Creating a string value in the add-ins registry is the recommended approach when a third party add-in is installed. This way, Discovery and the add-in do not need to know where each other is installed.

Contents of Manifest Files

Here is an example manifest file:

```
<?xml version="1.0" encoding="utf-8" ?>
<AddIns>
  <AddIn
    name="Sample Add-in"
    description=”Sample commands illustrating how to use the API.”
    assembly="SampleAddIn.dll"
    typename=”Sample.AddIn”
    host="NewAppDomain" />
</AddIns>
```

The AddIns element encloses a list of add-ins. A single manifest file can describe more than one add-in, although commonly it describes just one add-in.

The **AddIn** element has five attributes.

Attribute	Description
name	The name of the add-in. This is presented to the user in the Options dialog.
description	The one-line description for the add-in is presented to the user in the Options dialog.
assembly	The filename or full path of the DLL that contains the add-in. If the assembly attribute only specifies a filename and not a full path, the assembly will be assumed to be in the same folder as the manifest file.
typename	The type of the add-in class to create when Discovery starts up. This type must be derived from AddIn and must implement the IExtensibility interface.
host	The way in which the add-in should be hosted. The only supported value for Discovery is: SameAppDomain. The SpaceClaim options NewAppDomain or NewProcess are related to .NetRemoting, which is not supported in Discovery.

Only the `host` attribute has a default value if not present. All other attributes must be specified.

While developing an add-in, the add-ins folder can be used in one of two ways:

1. Place the add-in assembly and its manifest file in a folder inside the add-ins folder. You can set the build output path directly to this folder to save any copying and set the properties of the manifest file so that it is copied to the output directory. In the manifest file, specify the add-in filename (not the full path) in the assembly attribute.
2. Place the manifest file in the add-ins folder, and the add-in assembly in a folder

Customizing the Ribbon

1. If the add-in needs to customize the ribbon it will implement the IRibbonExtensibility interface.
2. In the GetCustomUI method, return a string that contains the XML description of the items to be added to the ribbon.

2.4. Creating Custom Commands

Custom Commands

All User Interface (UI) objects in Discovery must be associated with a Command. A Command controls the text, tooltip, image, and handles the Executing and Updating events. If a new UI object is added to the Ribbon by the add-in, the add-in must either create a new Command or associate it with an existing Command.

An add-in implements the `ICommandExtensibility` interface to modify or add commands. Discovery calls the `ICommandExtensibility.Initialize` method at startup which allows the add-in to add or modify commands using the appropriate methods on the Command class.

Example code

```
using SpaceClaim.Api.V22;
using SpaceClaim.Api.V22.Extensibility;

namespace YourCompany.Example {
 public class SampleAddIn : AddIn, IExtensibility, ICommandExtensibility {
 ...

#region ICommandExtensibility Members
public void Initialize() {
 Command command = Command.Create("SampleAddIn.CreateGear");
 command.Text = Properties.Resources.CreateGearCommandText;
 command.Hint = Properties.Resources.CreateGearCommandHint;
 command.Image = Resources.CreateGear;
 command.Updating += CreateGear_Updating;
 command.Executing += CreateGear_Executing;
 }

 #endregion

 ...
 }
}
```

2.5. Customizing the Ribbon

Ribbon Customization

In Discovery, add-ins can be used to customize the user interface for their workflows. A good place to start with workflow customization is the ribbon. The ribbon is a graphical control element that appears along a horizontal strip at the top edge of an application window and can be used to organize related commands so that they are easier to find. It is the location in the user-interface where tabs like "Design" and "Simulation" are found. Typically, a ribbon includes tabbed toolbars that are filled with graphical buttons and other graphical control elements and grouped by functionality.

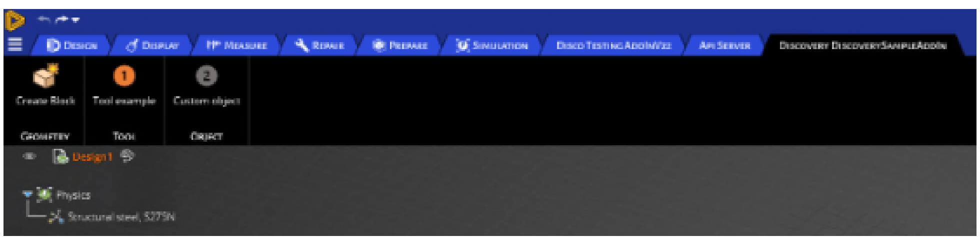
Each add-in can have its own tab or tabs in the ribbon to display and organize the tools created in the add-in.

To customize the ribbon, an add-in implements the IRibbonExtensibility interface. When Discovery is started it calls the add-in’s IRibbonExtensibility.GetCustomUI method, and the add-in returns a string that contains an XML description of the changes to be made to the ribbon. The add-in can add new UI objects to the ribbon or change existing ones.

Each UI object such as buttons, check boxes, etc. must be associated with a previously created Command. The command handles the setting of the UI object's text, image, tooltip, and any events that are raised.

Tabs

Add-ins can add a new tab to the ribbon.



Below is the XML code you can use to create the ribbon tab.

```
<customUI
   xmlns="http://schemas.spaceclaim.com/customui"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://schemas.spaceclaim.com/customui
         http://schemas.spaceclaim.com/customui/SpaceClaimCustomUI.V20.xsd">

 <ribbon>
    <tabs>
      <tab id="Discovery DiscoverySampleAddIn"
command="DiscoverySampleAddIn.RibbonTab" label = “My Tab”>
        <group id ="Geometry" command="DiscoverySampleAddIn.Geometry"
label="Geometry">
   <button id="DiscoverySampleAddIn.CreateBlock" size="large"
command="DiscoverySampleAddIn.CreateBlock"/>
        </group>
        <group id ="Tool" command="DiscoverySampleAddIn.Tool" label="Tool">
   <button id="DiscoverySampleAddIn.button1" size="large"
command="DiscoverySampleAddIn.FooTool"/>
 </group>
 <group id ="Object" command="DiscoverySampleAddIn.Object" label="Object">
   <button id="DiscoverySampleAddIn.button2" size="large"
command="DiscoverySampleAddIn.BarTool"/>
```

```
  </group>
       </tab>
     </tabs>
   </ribbon>
</customUI>
```

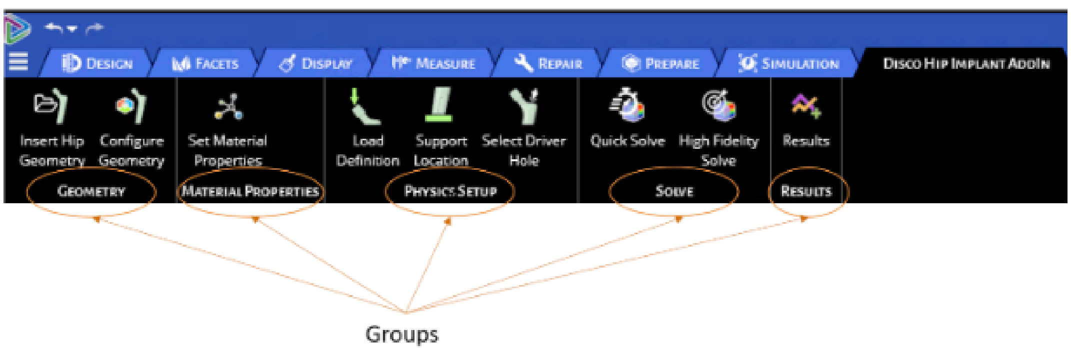
The XML content that defines the ribbon should be placed into the customUI tags. Under the ribbon tag, you can define a new tab using the tab object. It has the following three attributes:

- tab id
- label (optional)
- command (optional)

If a label is defined, it is used as the displayed name on the user-interface. If the label is not defined, the tab id is used as the displayed name.

Groups

A tab is the collection of buttons, but the buttons can be further categorized using "groups." Groups are useful to bring relevant buttons together and give them more context.



In the example above, "geometry," "material properties," etc. are groups that contain relevant buttons and functionalities.

Buttons

Buttons are the controls that can be used to execute a function (like theactivation of a tool). When defining the buttons in the customUI XML, you need to define its identifier (display name), command, and the size of the button. The size of the button should be defined as "large" for standard buttons requiring a row or "small" for mini buttons that require less space. Below is the schema file that contains examples.

```
<?xml version="1.0" encoding="utf-8"?>
<customUI
  xmlns="http://schemas.spaceclaim.com/customui"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.spaceclaim.com/customui
http://schemas.spaceclaim.com/customui/SpaceClaimCustomUI.V20.xsd">
  <ribbon>
   <tabs>
      <tab id="Discovery DiscoverySampleAddIn"
command="DiscoverySampleAddIn.RibbonTab">
        <group id ="Geometry" command="DiscoverySampleAddIn.Geometry"
label="Geometry">
          <button id="DiscoverySampleAddIn.CreateBlock" size="large"
command="DiscoverySampleAddIn.CreateBlock"/>
```

```
        </group>
        <group id ="Tool" command="DiscoverySampleAddIn.Tool" label="Tool">
          <button id="DiscoverySampleAddIn.button1" size="large"
command="DiscoverySampleAddIn.FooTool"/>
        </group>
        <group id ="Object" command="DiscoverySampleAddIn.Object"
label="Object">
          <button id="DiscoverySampleAddIn.button2" size="large"
command="DiscoverySampleAddIn.BarTool"/>
        </group>
      </tab>
  </tabs>
  </ribbon>
</customUI>
```

Customizing the File Menu

Additionally, custom buttons can be added to the file menu (at the top left, under the Discovery icon).

To add custom buttons to the file menu, encapsulate the tool buttons with <menu> <menu/>.

```
<ribbon>
    <menu>
      <button id="DiscoverySampleAddIn.Button1Tool" position="save" command="
 DiscoverySampleAddIn.Button1Tool"/>
      <button id=" DiscoverySampleAddIn.CustomTestTool3" position="close"
command=" DiscoverySampleAddIn.CustomTestTool3"/>
    </menu>
</ribbon
```

2.6. Customizing Tools

Custom Tool

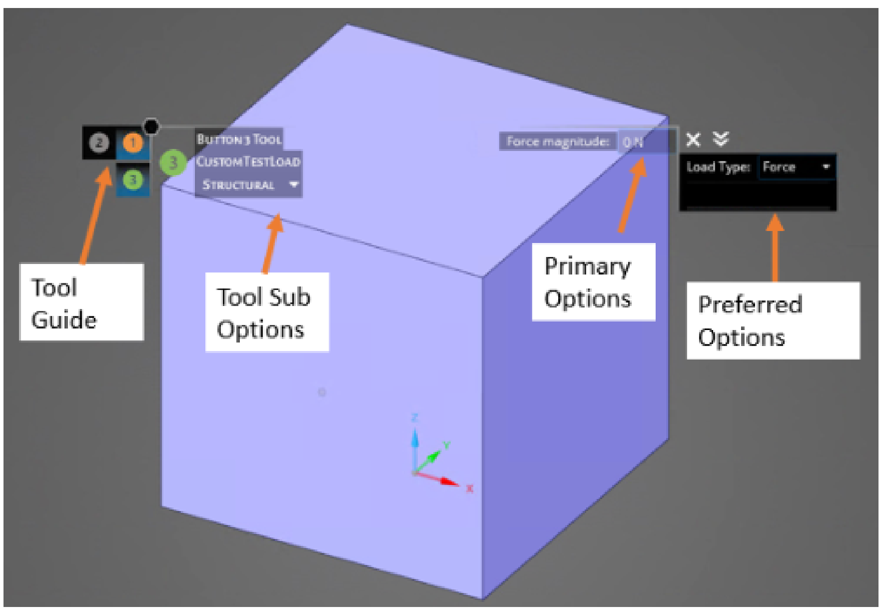
In Discovery, a tool is defined as a piece of functionality related to geometry manipulation (for example, the Pull tool for geometry extrusion) or a physical condition definition (for example, the creation of a Force). The definition of a tool requires user inputs, called tool options.

In the C# code, a tool is defined by a class derived from SpaceClaim.Api.V22.Tool. The OptionsXml property defines the layout of the panel that contains the tool options, like the ribbon XML.

```
class CustomTool : Tool
{
 public CustomTool()
   : base(InteractionMode.Solid) {}
      /*
       * Defines an options panel for the custom tool.
       */
 public override string OptionsXml
     {
  get { return Resources.CustomToolOptions; }
```

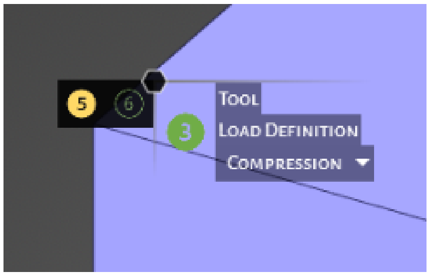
```
        }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<customUI xmlns="http://schemas.spaceclaim.com/customui">
  <panel command="DiscoverySampleAddIn.BarTool" label="Tool"
modeLabel="TextBox">
    <group id="DiscoverySampleAddIn.BarToolGuide" label="Tool Guides"
optionsType="toolGuide">
      <button id="DiscoverySampleAddIn.TickMark"
command="DiscoverySampleAddIn.TickMark"/>
    </group>
    <group id="DiscoverySampleAddIn.BarTool" label="Textbox Options"
optionsType="primary">
      <container id="DiscoverySampleAddIn.BarToolContainer"
layoutOrientation="vertical">
        <textBox id="DiscoverySampleAddIn.CustomObjetMagnitude"
command="DiscoverySampleAddIn.CustomObjectMagnitude" width="80" label="Double
 only:" />
      </container>
    </group>
  </panel>
</customUI>
```



Tool Guide

Another configurable tool item by the add-ins is the tool guide. It is a short cut for reaching tools from various tabs quickly.




```
<group id=" DiscoverySampleAddIn.CustomTestTool3ToolGuideGroup1" label="Tool
Guides 1" optionsType="toolGuide">
 <button id=" DiscoverySampleAddIn.CustomTestTool3ToolGuide" command="
DiscoverySampleAddIn.TextBoxTool"/>
 <button id=" DiscoverySampleAddIn.CustomTestTool3ToolGuide" command="
DiscoverySampleAddIn.CheckBoxTool"/>
</group>
```

Tool Sub-Options

If a tool has options to choose from, it should be expressed as below:

```
<group id="DiscoverySampleAddIn.FooToolSubOption" label="Tool Sub Options"
optionsType="toolSubOption">
   <toolSubOption id="DiscoverySampleAddIn.FooToolSubOption"
command="DiscoverySampleAddIn.FooToolSubOption" width="80">
        <item label="Sub Option1"/>
        <item label="Sub Option2"/>
   </toolSubOption>
</group>
```

Tool sub option is a collection of tool sub option items (unbounded) that will be shown ina dropdown. In the example above, there are two items with labels, "Sub Option1" and "Sub Option 2," which are also the display names. Using this code, items tools options can be grouped and filtered. The command for the tool sub option can be defined as shown below in C# and linked to the XML by the command name.

```
static class FooToolSubOptionComboBox
    {
        const string commandName = "DiscoverySampleAddIn.FooToolSubOption";

        public static readonly string[] toolSubOptionStrings =
        {
            "Sub Option1",
            "Sub Option2"
        };

        public static void Initialize()
        {
            Command command = Command.Create(commandName);

            string[] items = Array.ConvertAll(toolSubOptionStrings,
toolSubOption => toolSubOption.ToString());
            command.ControlState = ComboBoxState.CreateFixed(items, 0);
        }

        public static Command Command => Command.GetCommand(commandName);

        public static string SelectedLoadType
```

```
        {
            get
            {
                var state = (ComboBoxState)Command.ControlState;
                return toolSubOptionStrings[state.SelectedIndex];
            }
        }

    }
```

Primary Options

The primary options are mostly used to add another level of selection. They should be created as another group in the tool XML file. They are most likely the key input item for the tool.

Below is an example add-in showing how you can create primary options.

```
<group id="DiscoverySampleAddIn.FooToolPrimary" label="Textbox Options"
optionsType="primary">
      <container id="DiscoverySampleAddIn.FooToolContainer"
layoutOrientation="vertical">
        <checkBox id="DiscoverySampleAddIn.CheckBox" label="Unitless"
command="DiscoverySampleAddIn.CheckBox"/>
        <textBox id="DiscoverySampleAddIn.TextBoxDouble"
command="DiscoverySampleAddIn.TextBoxDouble" width="80" label="Double only:"
/>
        <textBox id="DiscoverySampleAddIn.TextBoxDouble2"
command="DiscoverySampleAddIn.TextBoxDouble2" width="80" label="Double only:"
 />
        <textBox id="DiscoverySampleAddIn.TextBoxQuantity"
command="DiscoverySampleAddIn.TextBoxQuantity" width="80" label="Quantity :"
quantityType="temperature" />
      </container>
</group>
```

In primary options, multiple buttons or textbox fields can be stacked together. As shown in the example above, they are added to a container with a layout orientation of "vertical."

The example above also defines some check boxes and text boxes. (See controls for the completelist of controls applicable in the tools.)

Textbox can be set to work with a specific type by setting the quantityType attribute.

Preferred Options

Preferred options can be found on the right-side of the tool and they appear with a black background. Preferred options are built in a similar way to the primary options. You can customizepreferred options with a variety of buttons and set their visibility.

To create a preferred options block, the add-in needs to define a group with optionsType="preferred." Then, using containers, the items layout can be specified.

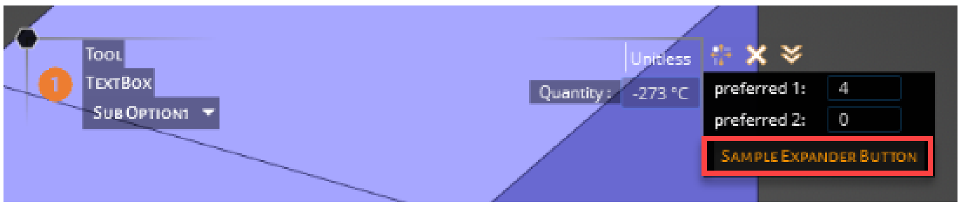
In the example below, the preferred optionsare stacked vertically. As you can see, a button item can be used as an expander to group other items. Also, shown below, the spin box is only visible when a button is clicked. This button functionality can be useful when there are multiple fields that need to be activated under certain conditions.

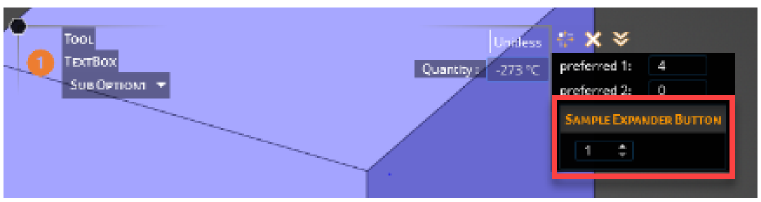
```

<group id="DiscoverySampleAddIn.FooToolPreferred" label="Textbox Options"
optionsType="preferred">
```

```
     <container id="DiscoverySampleAddIn.FooToolContainer"
layoutOrientation="vertical">
       <textBox id="DiscoverySampleAddIn.PreferredTextBox1"
command="DiscoverySampleAddIn.PreferredTextBox1" width="80" label="preferred
1:" />
       <textBox id="DiscoverySampleAddIn.PreferredTextBox2"
command="DiscoverySampleAddIn.PreferredTextBox2" width="80" label="preferred
2:" />
       <button id="DiscoverySampleAddIn.ExpanderButton"
command="DiscoverySampleAddIn.ExpanderButton"  label="Sample Expander Button">

          <spinBox id="DiscoverySampleAddIn.SpinBox"
command="DiscoverySampleAddIn.SpinBox" minimumValue="1" maximumValue="8"
width="35"/>
       </button>
     </container>
</group>
```





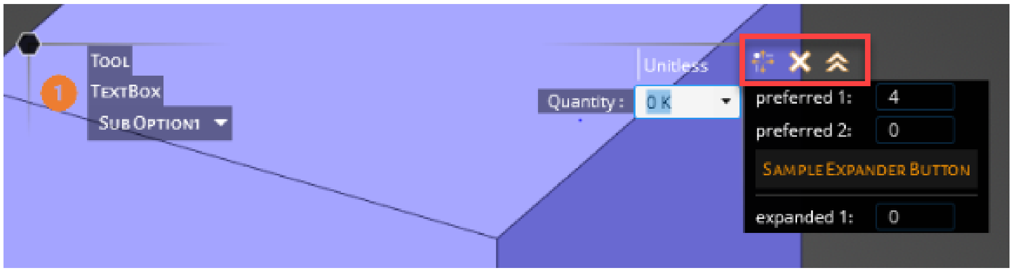
Expanded Options

Expanded options allow for further grouping on the tools and they work in a similar way to buttons, but visually they are represented as arrows on the preferred options. To create expanded options, define a group in XML using optionsType="expanded."

```
<group id="DiscoverySampleAddIn.FooToolExpanded" label="Textbox Options"
optionsType="expanded">
        <container id="DiscoverySampleAddIn.FooToolContainer"
layoutOrientation="vertical">
          <textBox id="DiscoverySampleAddIn.ExpandedDouble1"
command="DiscoverySampleAddIn.ExpandedDouble1" width="80" label="expanded 1:"
 />
        </container>
...
```

The items inside of this group only become visible after clicking the double arrow above the preferred options.

Published: 2024-01-05T21:40:16.301-05:00



2.7. Custom Objects and Custom Tree

Discovery allows you to create custom objects using custom wrappers and it allows you to display them in a custom tree. This section is a brief introduction to the APIs for custom wrappers and the definition of the custom tree.

2.7.1. Custom Object Wrappers

Within Discovery add-ins, custom objects can be created by implementing one of the wrapper objects. This section covers the following topics:

- CustomWrapper
- DiscoveryCustomWrapper
- PhysicsCustomWrapper

2.7.1.1. CustomWrapper

This is the standard wrapper class from SpaceClaim. For more information on *CustomWrapper*, refer to SpaceClaim documentation.

2.7.1.2. DiscoveryCustomWrapper

A class derived from *CustomWrapper* that provides additional properties for display in the custom tree. Although it is possible to display a *DiscoveryCustomWrapper* in the design tree, the typical usage involves the display in the custom tree. The position of the objects in the custom tree can be defined in two ways:

1. By specifying a *TreeId* and *Group* that link the object to the custom tree definition (see **Custom Tree** section for more information).
2. By specifying a *TreeParent* that makes the custom object child of another custom object. The parent child/relationship is for custom tree display only.

   If specified, the *TreeParent* property overrides the *Group* property and places the object underneath its parent objects.

After the initial specification, the *TreeParent* and *Group* properties can be changed dynamically if desired. The drag and drop functionality (see **Custom Tree** section for more information) takes advantage of the dynamic specification and allows the User Interface to reparent an object either in another group or as a child of another object.

The *DiscoveryCustomWrapper* objects can also display a field in the custom tree. The field is defined by a text (*TreeFieldText*) and a value (*TreeFieldValue*). The *TreeFieldValue* can be specified as a double (no units).

A complete list of the properties of *DiscoveryCustomWrapper* objects is given below:

- **Name**: gets and sets the name of the wrapper object.
- **Group**: gets and sets the custom tree group of the wrapper object.

- **TreeId**: gets and sets the tree id of the wrapper object. This is how a tree can automatically find all items that are defined to it when it is refreshed.
- **ImageKey**: gets and sets the image key for the wrapper objects.
- **TreeFieldText**: gets and sets the tree field text.
- **TreeFieldValue**: gets and sets the tree field value (double is supported) .
- **TreeParent**: gets and sets the tree parent object which can be another custom object.
- **ShowInDesignTree**: specifies whether the custom object should show in the design tree.

2.7.1.3. PhysicsCustomWrapper

A class derived from *DiscoveryCustomWrapper* with two extra properties:

1. Simulation
2. TreeFieldQuantity

This wrapper can be used for objects connected to simulations. Each wrapper object is linked to one simulation. The Discovery code deletes/updates when a simulation is deleted or duplicated. It also shows in the custom tree only those objects which are associated with the current simulation. When you create a new simulation or change to a different one, the tree updates and the objects not associated with the simulation are not displayed.

Additionally, while *DiscoveryCustomWrapper* objects has only a value field on the custom tree, *PhysicsCustomWrapper* objects can display *TreeFieldQuantity*, which is a value with units. The *TreeFieldQuantity* field can be used to display a physical quantity associated with the object.

Sync-ing TreeField and HUD properties

In the Discovery UI paradigm, the *TreeField* is used as a short cut for an entry from the HUD that displays the tool options associated with the object in the tree. The *TreeField* is visible even though the tool is not selected, so custom objects can be modified regardless of whether their tool is active. When an add-in uses the *TreeField,* it is its responsibility to keep its value in sync with the HUD properties.

```
protected override void OnEnable(bool enable)
        {
            if (enable)
            {
                CustomObjectMagnitude.Command.TextChanged +=
barObjectMagnitudeCommand_TextChanged;
                Document.DocumentChanged += TreeField_Updated;
            }
```

The add-in can register a listener to the *DocumentChanged* event in order to be notified of changes in the quick field. The *DocumentChanged* event can be filtered for changes in the custom object and the appropriate action can be taken to sync the event payload to the corresponding property.

```
private void TreeField_Updated(object sender, DocumentChangedEventArgs e)
        {
            foreach (var changedObj in e.ChangedObjects)
            {
                if (changedObj is CustomObject customobject)
                {
                    var changedLoad = BarObject.GetWrapper(changedObj as
CustomObject);

                    if ((CustomObjectMagnitude.Command.Text !=
changedLoad.TreeFieldQuantity.ToString()))
                    {
```

```
                        CustomObjectMagnitude.Command.Text = 
changedLoad.TreeFieldQuantity.ToString();
                        WriteBlock.ExecuteTask("update quantity", () => 
UpdateTreeQuantity(changedLoad));
                    }
                }
            }
        }
```

When the HUD property associated with *TreeField* is changed, it is the add-in responsibility to set the *TreeField* property to the updated value.

```
customBarObject.TreeFieldQuantity = customBarObject.RatioQuantity;
```

The Discovery code automatically updates the *User Interface* in response to the *TreeField* change.

2.7.2. Custom Tree

Discovery uses trees to represent objects for geometry and physics. Add-ins can use the custom tree to display their own custom objects.

Within an add-in a custom tree is defined by implementing the *ITreeHierarchy*interface. The class that implements *ITreeHierarchy*interface must be marked with the attribute

```
[Export(typeof(ITreeHierarchy))]
```

Using Managed Extensibility FrameWork (MEF), Discovery collects the classes that implement *ITreeHierarchy* and adds them to the application at the startup.

The interface has the properties below that a custom tree can define and modify.

- **ID**: identifier for custom tree.
- **TreeName**: user visible name for the custom tree.
- **DefaultGroup**: default group for custom tree items.
- **RootGroups**: list of the items that displayed as root nodes.
- **SubGroupMapping**: dictionary to define the hierarchy in tree items. Key is the parent node, value is the list of child nodes.
- **GroupsToDisplayName**: dictionary to match the tree groups and their display names on the tree.
- **GroupsToImages**: dictionary to match the tree groups and their images.
- **DefaultGroupThreshold**: minimum number of items required to form a group value of 0 or 1 groups immediately of -1 or lower never groups
- **UseParentGroupingThresholds**: whether subgroups will use the same threshold and other rules as the parent group
- **CountIndirectChildren**: if true, counts direct non-group children and counts each direct subgroup child as its indirect children, else counts each direct subgroup as 1
- **IgnoreThresholdAfterFirstSubGroup**: when true, groups will ignore its threshold and will form immediately if it contains a subgroup and anything else
- **UseDeepGrouping**: when true, grouping will be applied to the children of every node in the tree instead of just the root node
- **GroupsToImageKeys**: when true, enables you to change the grouping of objects using drag and drop
- **RootNodeImageKey**: image key for the root node.
- **StageIdentifiers**: identifiers of stages where this tree will be displayed.
- **IsDragEnabled**: when true, enables you to change the grouping of objects using drag and drop
- **CanDrop**: allows add-in control on enabling/disabling the target of the drop

The *Custom tree* definition is flexible in terms of its hierarchy and how objects are represented with respect to each other. The drag and drop feature also enables changing the hiearchy during the session. As noted in the previous section, the specification of a *TreeParent* for a custom objects overrides the group definition.

Figure 1. Custom objects are organized in linear form.

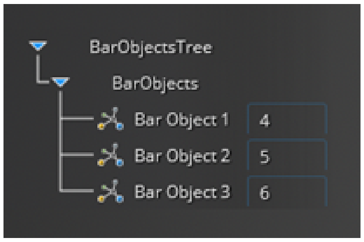
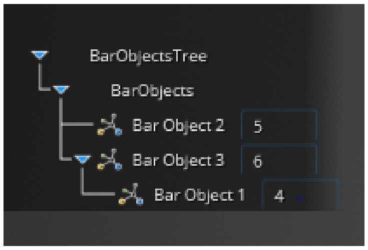


Figure 2. Object 1 is nested into object 3.



By using the *StageIdentifer* property add-ins can control the visibility of the custom tree based upon the current stage. The *Custom Tree* will only show in the stages specified in the *StageIdentifers* list and won't be displayed in the other stages. In the figures below, the tree is shown in the *CustomStage* that the add-in defined, but it is not shown in the *Refine* stage.

2.8. Sample Projects

The Sample Add-in project (C#) contains programming examples of how to use various areas of functionality in the API.

Sample Project	Version	Language	Link
Discovery Basic Template #1: Command Example • Demonstrates how to create an add-in and use a command to create a sample block.	API V241	C#	Visit the Ansys Discovery Space
Discovery Basic Template #2: Tool Example • Demonstrates how to define a button that activates a tool on the HUD. The tool has a textbox for double values and a second toolbox for an example physical quantity temperature. A check box allows you to switch between the two textboxes, chaining their visibility and keeping values in sync between the two fields.	API V241	C#	Visit the Ansys Discovery Space

Discovery Basic Template Installer #3: Custom Object/Custom Tree Example • Includes the functionality from the Discovery Basic Template #2: Tool Example and adds functionality to create a custom object that is displayed in the custom tree. The example shows how to link the custom object with the tool properties displayed in the HUD.	API V241	C#	Visit the Ansys Discovery Space
Sample Add-in • Copy Component - Show how to create a new translated component instance. • Create Assembly - Shows how to create a simple assembly with two positioned instances of one part, and one instance of another part. • Create Block - Shows how to create a block by extruding a rectangular profile. • Create Gear - Shows how to create a simple gear by extruding a complex profile involving lines and arcs. • Create Hole - Shows how to create a hole by subtracting a cylinder. • Create Notes - Shows how to create an annotation plane and then create notes at various angles, with different fonts and colors. • Create Torus - Shows how to create a torus by sweeping a circular profile around a larger circle. • Export - Shows how to export a document in various file formats. • Find Matching Faces - Shows how to traverse bodies and faces in an assembly and compare their geometry. • Scene Graph - Creates a scene graph. • Select Loop - Shows how to access the topology of a body and traverse edges, loops and fins. • Sketch Tool - Shows how to create an interaction tool for sketching lines. • Spin Component - Demonstrates animation by rotating selected component. • Profile Tool - Shows how to create and modify a custom object using an interaction tool.	API V241	C#	Visit the Ansys Discovery Space

Building the Sample Add-in Project

1. Download the SampleAddin zip file and unzip it.

> **Note:** Note: Use the Extract All command on the content menu in Windows Explorer.

2. In the extracted folder, locate the reference to the NuGet package (NuGet.config) and delete it.
3. Launch Visual Studio and open the project file (SampleAddin.csproj). Alternatively, you may double-click the project file and open it with Visual Studio.
4. In the Solution Explorer in Visual Studio, right-click References and select Add Reference.
5. In the Reference Manager, select the Browse tab. Locate the API assembly in the subfolder inside the Discovery Installation folder. Right-click this reference and set Copy Local to False.
6. Build the project is Visual Studio.

   > **Note:** Build the project using x64.

7. Launch Discovery.

   The Sample Add-in will appear in the Discovery Ribbon.

> **Note:** Samples that require a beta version of the API must reference that beta version.

> **Note:** To build the samples or any add-in, .NET 4.8 is required. If .NET 4.8 is not available to build with in your version of Visual Studio, the Microsoft .NET Framework 4.8 Developer Pack can be downloaded and installed for Visual Studio 2017 and Visual Studio 2019.

> **Note:** Sample projects have their build output path set to a folder beneath "%ALLUSERSPROFILE%\SpaceClaim\AddIns", which means Discovery will automatically find the add-in at start-up (see Manifest Files).

Add-ins Frequently Asked Questions

1. **What can I do when my Add-in project can't build because it's missing references or packages? Add-ins require several Discovery DLL libraries:**

   Discovery.API.2X

   SpaceClaim.API.2X

   SpaceClaim.API.2X.Internal

2. **What can I do when my Add-in project is built, but I can't see it in Discovery?**

   Either the Add-in is not completely built, or it is just not visible on Discovery.

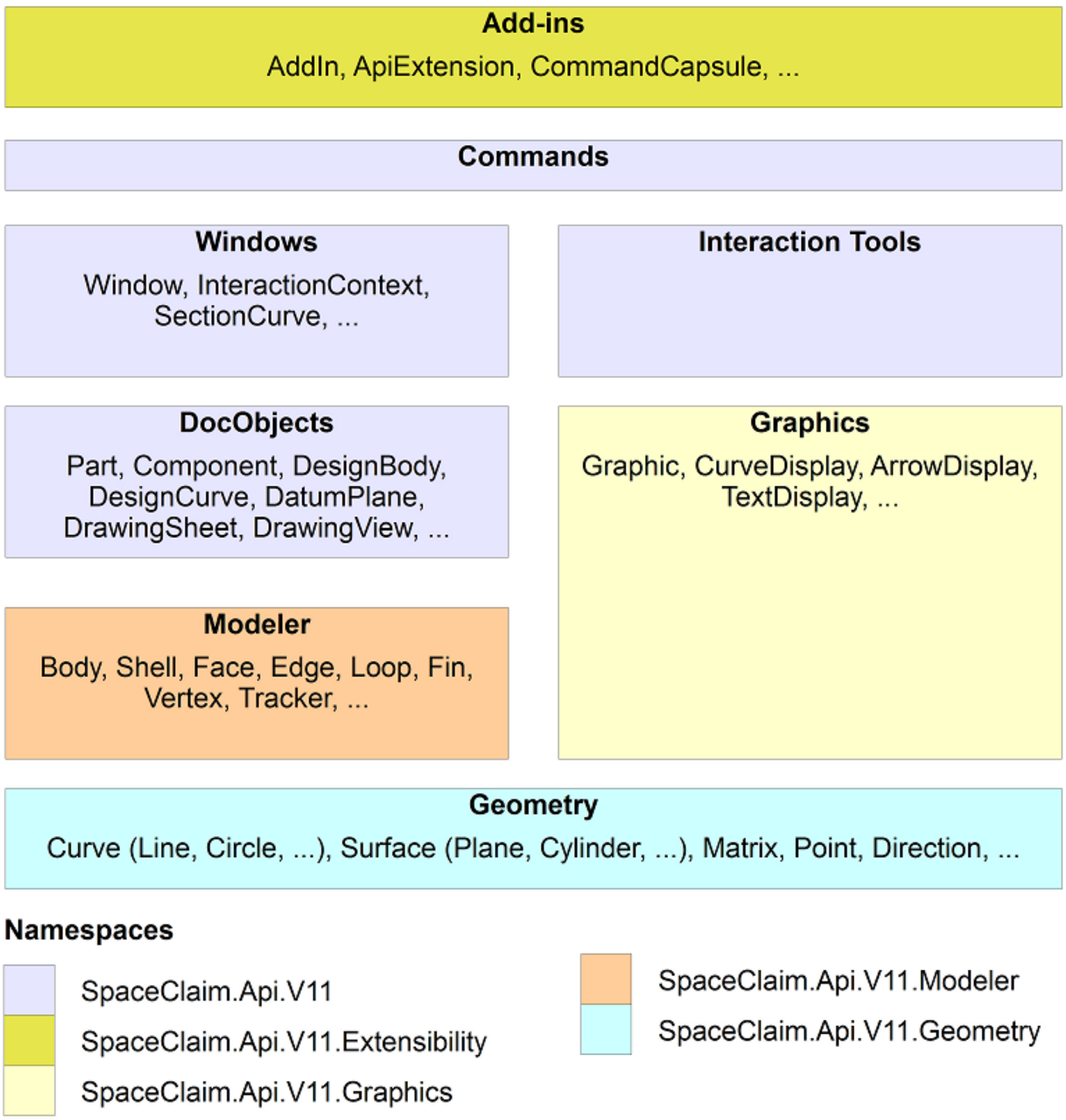
   Check Discovery Settings > Add-in and determine if the add-in is present. If it's found, go to the Ribbon section in the advanced part.

If the add-in is not found, verify that the add-in output files are configured for the Program Data folder. The manifest file will also appear there.

3: Geometry API Overview

The diagram below shows a conceptual view of the geometry architecture as presented through the API.

For the purposes of this diagram, each box represents a module, where each module makes use of other modules below it in the diagram. The box sizes have no significance.



These modules are merely conceptual groupings, since the only separation apparent in the API is the separation into namespaces, which are shown in different colors in the diagram.

Listed within many of the modules in the diagram are examples of types published by that module.

3.1. Doc Objects

Of particular importance is the distinction between *doc objects*, and the lower-level modeler and geometric objects. Doc objects, as the name suggests, belong to documents. They are first class objects, since they belong to a parent-child hierarchy, and they provide monikers (for persistent identifiers) and update states (for associative update).

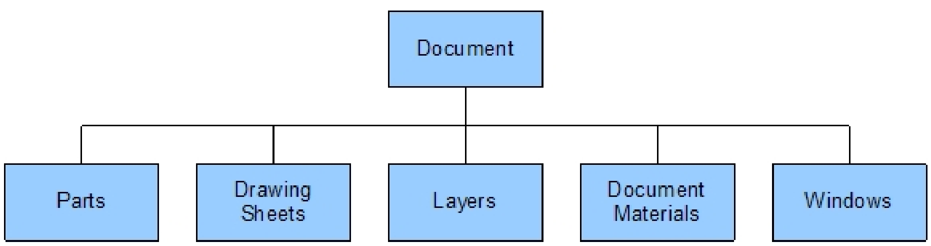
Many doc objects have references to modeler or geometry objects:

- Design bodies, design faces, and design edges are doc objects that have a reference to a corresponding modeler object: a body, face, or edge. You can create a modeler body without creating a design body, but this means no document is modified and nothing appears in the window. This can be useful if you want to perform some modeling calculation. You may or may not create a design body at the end.
- Design curves are doc objects that have a reference to a trimmed curve, which is a geometry object. Again, you can create curves and trimmed curves without ever creating a design curve, but design curves are what get displayed in windows. (You can create graphics display primitives, which reference geometry objects too, and these also get displayed in the window, although display primitives are not doc objects.)
- Datum planes are doc objects that have a reference to a plane, which is a geometry object. Again, you can create planes and other surfaces without ever creating a datum plane.

Typically, the doc object will have more properties than the modeler or geometry object that it references, such as name, layer, visibility, or color.

3.2. Document Structure

A document contains the following objects:



Parts

A document always contains at least one part, known as its *main part*, and this represents the design. If the main part has internal components (instances of other parts that belong to the same document), the document will contain other parts too.

Internal components are also used for beam profiles, mid-surface parts, and sheet metal unfolded parts.

The structure of a part is described below.

Drawing Sheets

A document contains zero or more drawing sheets.

The structure of a drawing sheet is described below.

Layers

A document contains one or more layers. There is always a *default layer*, and if you delete another layer, all its objects are moved to the default layer. You cannot delete the default layer. The default layer is not the same as the *active layer*, which is the layer to which new objects are assigned. The active layer is a property of the window.

Materials

A document contains zero or more document materials, which are materials used by parts, design bodies, or beams in that document.

Windows

A document contains one or more windows, but it may not have any windows loaded. If the document is explicitly opened, then its windows are also loaded and opened, but if a document is loaded implicitly, for example because it is referenced from another open document, then its windows are not loaded.

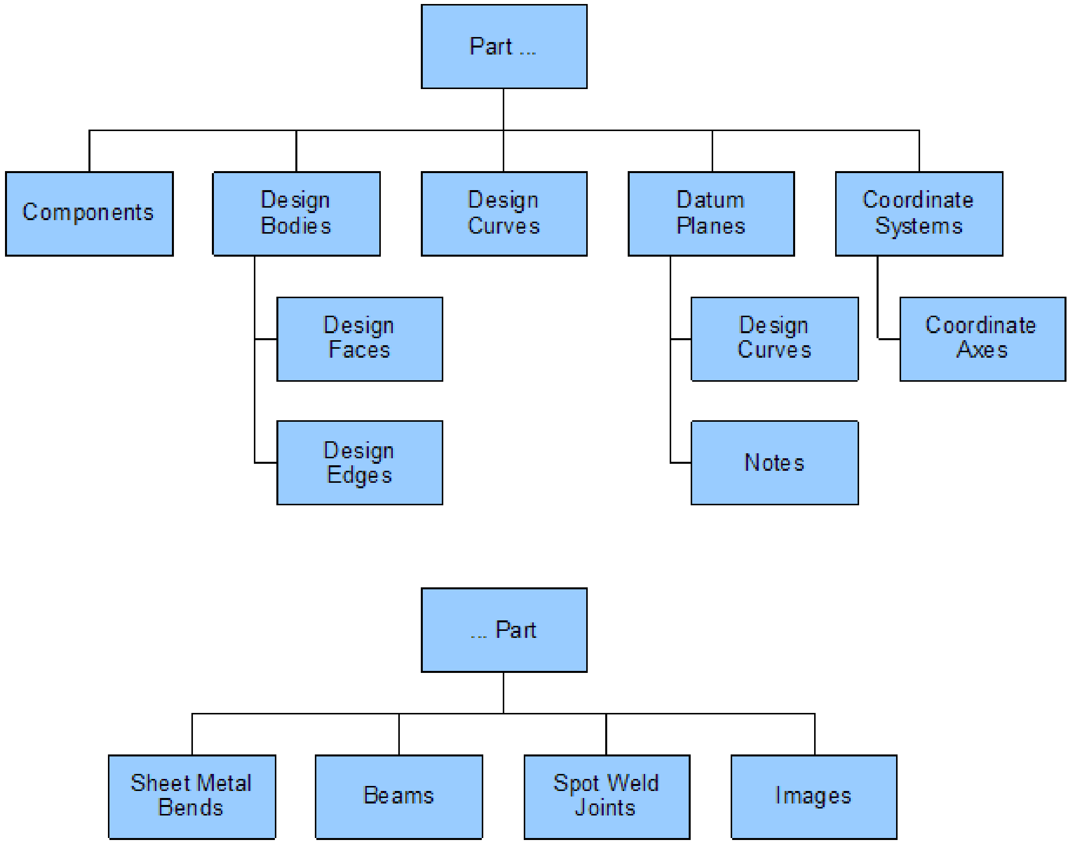
A window shows a *scene*, which is the root of the object hierarchy it displays. The window scene can be a part or a drawing sheet belonging to the same document.

The window also provides access to *interaction contexts*, which allow you to work in a specific coordinate space. The interaction context presents the current selection in that coordinate space. A useful interaction context is the *active context*, which is the context in which the user is working.

3.3. Part Structure

The Discovery object model is quite flat, with the part being a bucket for many objects:

Components



A part contains zero or more components. A component is an instance of another *template* part. The template part may belong to the same document (an internal component), or it may belong to another document (an external component).

Design Bodies

A part contains zero or more design bodies. A design body can be open (a surface body) or closed (a solid body). A design body contains design faces and design edges.

Design Curves

A part contains zero or more design curves. Design curves have 3D geometry, even though they are often sketched in a plane. For example, if you copy and paste design edges, design curves are created, and these need not lie in a plane.

Design curves can also belong to datum planes and drawing sheets.

Datum Planes

A part contains zero or more datum planes. As well as serving as construction planes, as the name suggests, datum planes can also contain design curves and text notes, which lie in the plane. When the datum plane is moved, its children are moved too.

Coordinate Systems

A part contains zero or more coordinate systems. A coordinate system contains three mutually perpendicular coordinate axes.

The world coordinate system, which can be displayed in the user interface, does not belong to any document, and is not presented through the API.

Sheet Metal Bends

If a part is a sheet metal part, then it contains zero or more sheet metal bends, which might be cylindrical or conical. If a part is a sheet metal part, then it has a sheet metal *aspect*, which is a companion object presenting sheet metal information, including bends.

Beams

A part contains zero or more beams, which have a trimmed curve path, a planar cross section, and information about the position and orientation of the cross section relative to the beam path.

Spot Weld Joints

A part contains zero or more spot weld joints. A spot weld joint has a collection of spot welds, each of which welds two or more points on design faces.

Images

A part contains zero or more images. An image is a picture or video, either positioned in space or wrapped onto a design face. An image can also belong to a drawing sheet.

4: Geometry Architecture

This section discusses the geometry architecture and includes the following sections:

- Documents and Doc Objects
- Application Integration
- Storing Custom Data
- Identifiers During Export
- Geometry and Topology
- Accuracy
- Units

4.1. Documents and Doc Objects

A Document is the unit of loading and saving. Assuming it has been saved, a document corresponds to a Discovery scdoc file on disk.

A DocObject is an object that belongs to a document. Doc objects are not the only objects that get saved when the document is saved, but they are the only objects that have a Document property. Examples of doc objects include: Part, Component, DesignBody, DesignFace, DatumPlane, and Note.

Doc objects provide:

- A parent-child tree containment hierarchy.
- Monikers for persistent identification.
- Update states to indicate that the doc object has changed.
- Custom attributes for storing 3rd party data.

4.1.1. Parent-Child Hierarchy

Doc objects are part of a parent-child containment hierarchy, where the *parent* represents the container and the *children* represent the contents. If a doc object is deleted, all its *descendants* (its children, recursively) are also deleted.

For example, a Part contains zero or more DesignBody objects, each of which contains one or more DesignFace objects. The parent of a design face is a design body, and the parent of a design body is a part. Similarly, a design body is a child of a part, and a design face is a child of a design body.

```
Public static void Example(DesignBody desBody) {
 Part part = desBody.Parent;

 // a part is a root object, so it has no parent
 Debug.Assert(part.Parent == null);
 Debug.Assert(part.Root == part);

 // GetChildren<T> returns immediate children of type T
 foreach (DocObject child in part.GetChildren<DocObject>()) {
  // Parent returns the immediate parent
  Debug.Assert(child.Parent == part);
 }
```

```
// DesignBody.Faces is equivalent to GetChildren<DesignFace>
foreach (DesignFace desFace in desBody.Faces) {
 // Root returns the topmost parent
 Debug.Assert(desFace.Root == part);
}

// GetDescendants<T> gets all nested children of type T
   // the search starts with the object itself
foreach (DesignFace desFace in part.GetDescendants<DesignFace>()) {
 // GetAncestor<T> crawls up the parent chain to find an object of type T
 Debug.Assert(desFace.GetAncestor<Part>() == part);

 // the search starts with the object itself
 Debug.Assert(desFace.GetAncestor<DesignFace>() == desFace);
}
}
```

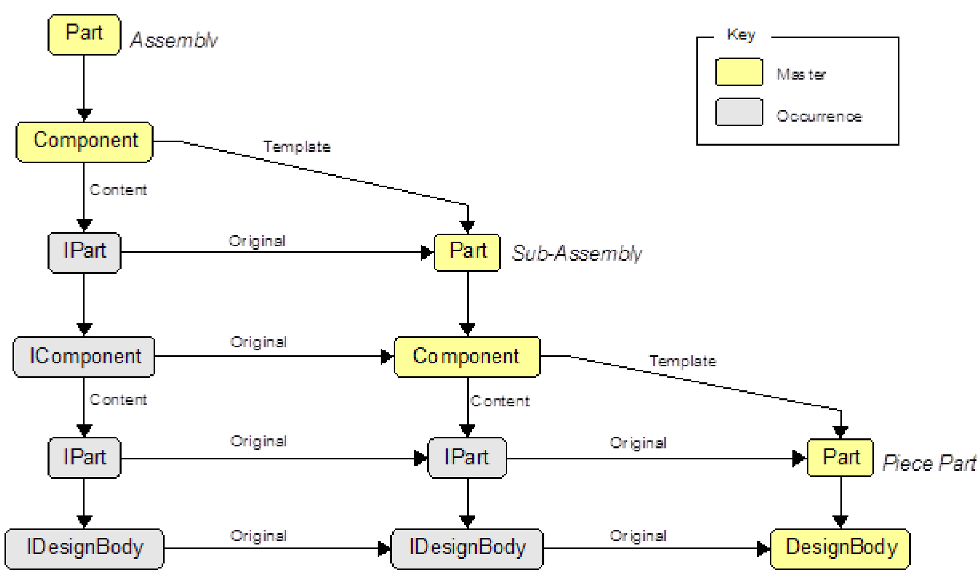
The parent chain continues up the hierarchy until the *root* object is reached. This is the topmost parent, which itself has no parent (its parent is null). Examples of root objects are: Part, DrawingSheet, and Layer.

All doc objects in the same parent-child hierarchy belong to the same document. The Document class provides properties to access its root objects: MainPart, Parts, DrawingSheets, and Layers.

4.1.2. Parts and Components

A Part contains zero or more DesignBody objects and zero or more Component objects. This means a part can contain both design bodies and components, in which case it is neither a pure piece part, nor a pure assembly. This is supported so that the interactive user can restructure design bodies into components, or vice versa.

A component is an *instance* of a *template* part. It has a *placement* matrix to position the component into assembly-space. The template is neither a child nor a parent of the component. If the parent-child hierarchy is visualized as an inverted tree structure with the root at the top and leaf nodes at the bottom, then the template is a sideways reference to another part, which is the root of another



The template part may live in the same document as the component, giving rise to what the interactive user would call an *internal component*, or it may live in another document, giving rise to an *external component*. Strictly speaking, it's the template part that is either internal or external, not the component itself.

4.1.3. Occurrence Chains

If the template object itself contains instances, occurrences of occurrences are produced. For example, this happens with a two-level assembly, since the top-level assembly contains a component that instantiates a sub-assembly part, which itself contains a component that instantiates a piece part.

4.1.4. General Objects and Masters

In general, a doc object is either an occurrence, or a *master*. When dealing with *general* doc objects (doc objects that might be occurrences or masters), interfaces are used, e.g. IPart, IComponent, and IDesignBody. These all derive from IDocObject.

There are some methods and properties that for theoretical or practical reasons are not presented by occurrences, and are therefore only presented by masters. When dealing with masters, classes are used, e.g. Part, Component, and DesignBody. These all derived from DocObject, which implements IDocObject.

Part implements IPart, Component implements IComponent, and so on, so a master supports everything that a general object supports, and often more besides.

Note that although Part is always a root object, IPart may or may not be a root object. If the IPart happens to be a master, then it is a root object, but if it happens to be an occurrence, its parent will be an IComponent or an IDrawingView.

4.1.5. Originals and Masters

IDocObject is the base for all doc object functionality. It provides members for traversing the parent-child hierarchy (Children, GetChildren<T>, GetDescendants<T>, Parent, GetAncestor<T>, Root), which we have already met, and it also provides members for traversing the occurrence structure:

- Original – Gets the original IDocObject, else null if this is a master.
- Master – Gets the ultimate original, i.e. the DocObject master at the end of the occurrence chain. If the object is a master, the object itself is returned.
- Instance – Gets the instance that produces this occurrence, else null if this is a master.

4.1.6. Transforming to Master-Space

The most common of these properties to use is Master, because the master can provide methods and properties not available on the general object. Since the master might be in a different coordinate-space to the general object, TransformToMaster can be used to get the transform that maps objects in general-space to objects in master-space.

```
Public static void Example(Icomponent comp, Frame placement) {
 // the Placement property is available on Component, but not Icomponent
 Component master = comp.Master;

 // map placement frame into master-space
 Matrix transToMaster = comp.TransformToMaster;
 Frame masterPlacement = transToMaster * placement;

 // apply master placement frame to master component
 master.Placement = Matrix.CreateMapping(masterPlacement);
}
```

4.1.7. Getting Occurrences

Having done some work in master-space, it may be necessary to obtain an object in general-space. This is common if the original object came from the window selection, and you wish to set a new selection in the window.

The window selection is always in window-space, i.e. in the same coordinate-space as the scene shown in that window (the same coordinate-system as Window.Scene). So if the window shows an assembly, then the selected objects are in assembly-space.

GetOccurrence can be used to obtain an object in general-space. It returns an occurrence that is similar to a companion object supplied. Note that the companion object is a general object, which may or may not be an occurrence. If it is an occurrence, then an equivalent occurrence is returned for the subject. If it is not an occurrence, then the subject itself is returned. This allows you to write code that works correctly without testing whether the object is in fact an occurrence.

```
Public static void Example() {
 Window window = Window.ActiveWindow;
 if (window == null)
  return;

 // the selected component is in window-space
 IComponent oldComp = window.SingleSelection as IComponent;
 if (oldComp == null)
  return;

 // copy the component master
 Component oldMaster = oldComp.Master;
 Component newMaster = Component.Create(oldMaster.Parent, oldMaster.Template);

 newMaster.Placement = oldMaster.Placement;

 // get an occurrence of the new master in window-space
 IComponent newComp = newMaster.GetOccurrence(oldComp);

 // select the newly created component
 window.SingleSelection = newComp;
}
```

4.2. Application Integration

This section covers the following topics:

- Persistent Identifiers
- Replacements
- Update States

4.2.1. Persistent Identifiers

Doc objects have persistent identifiers, called *monikers*, which can be recorded and later *resolved* to return the doc object again. The Moniker property returns a moniker for the doc object, and the Resolve method returns the doc object again.

Internally, master doc objects have an identifier, which is globally unique. Occurrences are identified by the instance, which is a master, along with the original object, which might itself be an occurrence. A moniker is an object that encapsulates the identifiers of the one or more master objects involved in the identity of the subject.

To record a moniker, you can record its string representation using ToString. The length of this string depends on the number of master objects involved. The format of this string is not documented, so you should not attempt to construct or modify such a string.

To convert the string representation back into a moniker, you can use `FromString`.

```
public static void Example(IDesignFace desFace) {
 Document doc = desFace.Document;

 // all doc objects provide a moniker
 Moniker<IDesignFace> desFaceMonikerA = desFace.Moniker;

 // resolve the moniker in the document to obtain the original object
 Debug.Assert(desFaceMonikerA.Resolve(doc) == desFace);

 // the string representation can be recorded
 string monikerText = desFaceMonikerA.ToString();

 // the moniker can be reconstructed from the string
 Moniker<IDesignFace> desFaceMonikerB =
Moniker<IDesignFace>.FromString(monikerText);
 Debug.Assert(desFaceMonikerB.Resolve(doc) == desFace);
 }
```

To resolve a moniker, a document must be provided as the context. Discovery allows more than one version of the same scdoc file to be loaded at the same time, so the same moniker could potentially be resolved in more than one document.

Since the internal identifiers involved are globally unique, there is no danger of resolving the moniker in an unrelated document. If you attempt to do so, `null` will be returned to indicate that the object was not found.

4.2.2. Replacements

If the doc object has been deleted, Resolve returns null to indicate that the object was not found. Sometimes doc objects can get replaced during a command. For example, if a design face is split during a modeling operation, it will be replaced by two new design faces. Perhaps one of these new design faces gets split itself, or perhaps one of them gets deleted.

Behind the scenes, replacements are recorded, and when Resolve is called, if the object has been replaced, the moniker automatically returns one of the survivors, or null if there are no survivors.

- If it is important to know whether the object is a survivor, rather than the original object, you can compare its moniker with the moniker you already have, using the == operator. If the object is a survivor, it will have a different moniker.
- If it is important to obtain all survivors, ResolveSurvivors can be used instead. Note that this method only returns surviving replacements, so if the object hasn't been replaced at all, no survivors are returned.

4.2.3. Update States

A doc object master has an *update state*, which tells you if the object has changed.

Each time the doc object master is changed, its update state changes. Conversely, if the update state has not changed, then the object has not changed. When an object is changed due to an undo (or redo) operation, its update state is undone (or redone) too.

The update state is persistent, so you can store its string representation and use it in another Discovery session.

```
public static void Example(DesignFace desFace) {
 // doc object masters provide an update state
 UpdateState beforeStateA = desFace.UpdateState;

 // the string representation can be recorded
 string stateText = beforeStateA.ToString();

 ModifyDesignBody();

 // test whether the design face was changed
 if (desFace.UpdateState != beforeStateA)
  Debug.WriteLine("Design face was changed.");

 // the update state can be reconstructed from the string
 UpdateState beforeStateB = UpdateState.FromString(stateText);
 Debug.Assert(beforeStateA == beforeStateB);
}
```

Update states are not provided for occurrences, but you can store the update states of the instances involved in the occurrence chain, along with the update state of the master. PathToMaster returns these instances.

The update state only represents the state of the object itself, and not the state of other objects it contains or references. For example, the update state of a part is not changed if a child design body is modified. Similarly, although the update state of a component will change if its placement is modified (since this is a property of the component itself), the update state will not change if its template part is modified.

4.3. Storing Custom Data

This section covers the following topics:

- Document Properties
- Custom Attributes
- Attribute Propagation

4.3.1. Document Properties

Documents have two types of properties:

- *Core properties* cover standard fields such as *description*, *subject*, *title*, and *creator*. The set of core properties is fixed. You cannot create new core properties.
- *Custom properties* allow 3rd party applications to store data with the document. Each custom property is a name-value pair.

So that custom property names do not clash when different applications choose a custom property name, the name should be prefixed with the application or add-in name, as in the following example:

```
public static void Example(Document doc) {
 CustomProperty.Create(doc, "BananaWorks.ApplicationVersion", 14);

 CustomProperty property;
 if (doc.CustomProperties.TryGetValue("BananaWorks.ApplicationVersion", out
property))
  Debug.Assert((double) property.Value == 14);
}
```

Note that a document can contain more than one part, so if you want to store data for a part, this is best done by storing a custom attribute on the part master (see next topic).

4.3.2. Custom Attributes

Doc object masters provide custom attributes so that 3rd party applications can store data. Two types of attribute are provided: text attributes and number attributes. An attribute is a name-value pair. A doc object can have a text attribute and a number attribute with the same name.

So that attribute names do not clash when different applications choose an attribute name, the name should be prefixed with the application or add-in name, as in the following example:

```
public static void Example(DesignBody desBody) {
 desBody.SetTextAttribute("BananaWorks.SkinCondition", "Ripe");

 string skinType;
 if (desBody.TryGetTextAttribute("BananaWorks.SkinCondition", out skinType))
  Debug.Assert(skinType == "Ripe");
}
```

Multiple values can be stored as multiple attributes with distinct names, or they can be formatted into a single text string using String.Format or an XML serializer.

4.3.3. Attribute Propagation

Attributes applied to doc object masters are propagated if the object is replaced. For example, if a design face has a text attribute, and this face is split during a modeling operation, the replacement design face fragments will also carry the same text attribute.

4.4. Identifiers During Export

When an ACIS or Parasolid file is written, either by the user or by calling Part.Export, *name* attributes are attached to face and edge entities in the file to indicate which design face or design edge master they came from. This is useful if the model is changed and then a new file is exported, since corresponding faces and edges will have the same *name* attributes.

- An ACIS *name* attribute is a "named attribute" (ATTRIB_GEN_NAME) with the attribute name, "ATTRIB_XACIS_NAME".
- A Parasolid *name* attribute is a system attribute with the name, "SDL/TYSA_NAME".

Design face and design edge masters have an ExportIdentifier property, which returns a string containing the value of the name attribute that is written when the object is exported.

4.4.1. Identifying Objects in ACIS and Parasolid Files

This section covers the following topics:

- Identifiers During Export
- Foreign Identifiers During Import and Export

4.4.2. Foreign Identifiers During Import and Export

There may be a requirement to import a model from another system, modify it in Discovery, and then export it again, such that the other system can track the identity of faces and edges during this round trip.

When importing an ACIS or Parasolid file, if any body, face, or edge entities have *id* attributes, these are converted to Discovery text attributes on the resulting design body, design face, or design edge masters. These text attributes have the reserved name, "@id".

- An ACIS *id* attribute is a "named attribute" (ATTRIB_GEN_NAME) with the attribute name, "ATTRIB_XACIS_ID".
- A Parasolid *id* attribute has the name, "ATTRIB_XPARASOLID_ID", and has an attribute definition, which is described in the documentation for the Part.Export method.

Attributes applied to doc object masters are propagated if the object is replaced. For example, if a design face has a text attribute, and this face is split during a modeling operation, the replacement design face fragments will also carry the same text attribute.

When exporting an ACIS or Parasolid file, if any design bodies, design faces, or design edges have text attributes with the name, "@id", these are written as *id* attributes applied to resulting ACIS or Parasolid entities.

4.5. Geometry and Topology

This section addresses the following topics:

- Unbounded Geometry and Trimmed Objects
- Topology
- Doc Objects and Geometry Design
- Curves
- Design Bodies
- Shape in General

4.5.1. Unbounded Geometry and Trimmed Objects

In Discovery, geometry is conceptually unbounded, e.g. a Plane extends indefinitely, a Sphere is complete, and a Line is infinite in length. On top of this there are *trimmed* objects, which are bounded and therefore provide additional properties:

- ITrimmedCurve – a bounded curve. It provides Length and parametric Bounds.
- ITrimmedSurface – a bounded surface. It provides Area, Perimeter, and a parametric BoxUV.
- ITrimmedSpace – a bounded region of 3D space. It provides Volume and Surface Area.

All of these inherit from IBounded, which provides GetBoundingBox.

A trimmed curve has a Geometry property that returns the Curve, and a trimmed surface has a Geometry property that returns the Surface. A trimmed region even has a Geometry property that returns a Space object representing all of Cartesian 3D space.

Trimmed curves and trimmed surfaces also have an IsReversed property, which tells you whether the sense of the object is the opposite of the sense of its geometry. The sense of a trimmed curve is its direction, and the sense of a trimmed surface is which way its normals face.

4.5.2. Topology

The topology of a model is made of Body, Face, and Edge objects, along with other objects (shells, loops, fins, and vertices) that describe in more detail how they connect up.

- Body inherits from ITrimmedSpace. It also provides Faces and Edges.
- Face inherits from ITrimmedSurface. It also provides surrounding Edges.
- Edge inherits from ITrimmedCurve. It also provides adjacent Faces.

Topology classes have more information than the trimmed object interfaces that they implement:

- Trimmed object interfaces have no concept of connectivity.
- Although they can return area and volume, respectively, ITrimmedSurface and ITrimmedSpace say nothing about the shape of their boundary. (With ITrimmedCurve, the boundary has no shape as such, since the curve is simply bounded by parameter values.)

4.5.3. Doc Objects and Geometry

Topology objects (Body, Face, Edge, etc.) and geometry objects (Plane, Cylinder, Line, etc.) are not doc objects. They are not part of a parent-child hierarchy, and they do not have monikers or update states. They are lower level objects, since they might be referenced by a doc object, but they have no knowledge of documents and doc objects themselves.

You can construct geometry, trimmed objects, and even solid bodies, in order to perform geometric calculations, without modifying a document:

```
public static void Example() {
 // create infinite line with zero parameter at the origin
 Line line = Line.Create(Point.Origin, Direction.DirX);

 // create line segment from (-0.01, 0, 0) to (0.01, 0, 0)
 ITrimmedCurve trimmedCurve = CurveSegment.Create(line, Interval.Create(-0.01,
 0.01));
 Debug.Assert(Accuracy.EqualLengths(trimmedCurve.Length, 0.02));

 // find closest point to (0.05, 0.05, 0)
 CurveEvaluation eval = trimmedCurve.ProjectPoint(Point.Create(0.05, 0.05,
0));

 // closest point on line segment should be (0.01, 0, 0)
 Debug.Assert(eval.Point == Point.Create(0.01, 0, 0));
}
```

The document is modified when you create doc objects. For example, you might create a design curve from a trimmed curve.

4.5.4. Design Curves

Design curves are what the end user refers to as *sketch curves*. They are called *design curves* in the API for consistency with design bodies, design faces, and design edges.

A DesignCurve is a doc object, which has a trimmed curve Shape.

```
public static void Example(Part part) {
 Line line = Line.Create(Point.Origin, Direction.DirX);
 ITrimmedCurve shape = CurveSegment.Create(line, Interval.Create(-0.01, 0.01));
```

```
 // create a design curve
 DesignCurve desCurve = DesignCurve.Create(part, Plane.PlaneXY, shape);

 // the Shape property returns the trimmed curve
 Debug.Assert(Accuracy.EqualLengths(desCurve.Shape.Length, 0.02));

 // override the layer color
 desCurve.SetColor(null, Color.DarkSalmon);
}
```

A design curve has other properties that are outside the concept of a trimmed curve, such as layer, color override, and name.

4.5.5. Design Bodies

Just as a DesignCurve is a doc object, which has a Shape of type ITrimmedCurve, there is a similar pattern for bodies:

- A DesignBody has a Shape of type Body.
- A DesignFace has a Shape of type Face.
- A DesignEdge has a Shape of type Edge.

This is only true for these doc object masters. For the corresponding general objects, less information is available:

- IDesignBody has a Shape of type ITrimmedSpace.
- IDesignFace has a Shape of type ITrimmedSurface.
- IDesignEdge has a Shape of type ITrimmedCurve.

This means, you can only access detailed topology information, such as loops and fins, from masters:

```
public static void Example(IDesignFace desFace) {
 // we can get the area from the ITrimmedSurface shape
 double area = desFace.Shape.Area;

 // but to access loops, we need to use the master
 DesignFace desFaceMaster = desFace.Master;
 // if we access geometry, remember we are now in master-space
 Matrix transToMaster = desFace.TransformToMaster;

 DesignBody desBodyMaster = desFaceMaster.Parent;

 // the master Shape is a Face rather than an ITrimmedSurface
 Face face = desFaceMaster.Shape;
 foreach (Loop loop in face.Loops)
  foreach (Fin fin in loop.Fins) {
   Edge edge = fin.Edge;

   // get from shape back to doc object master
   DesignEdge desEdgeMaster = desBodyMaster.GetDesignEdge(edge);

   // from master to occurrence equivalent to desFace
   IDesignEdge desEdge = desEdgeMaster.GetOccurrence(desFace);
  }
}
```

However, the general interfaces do provide some convenient connectivity traversals at the doc object level:

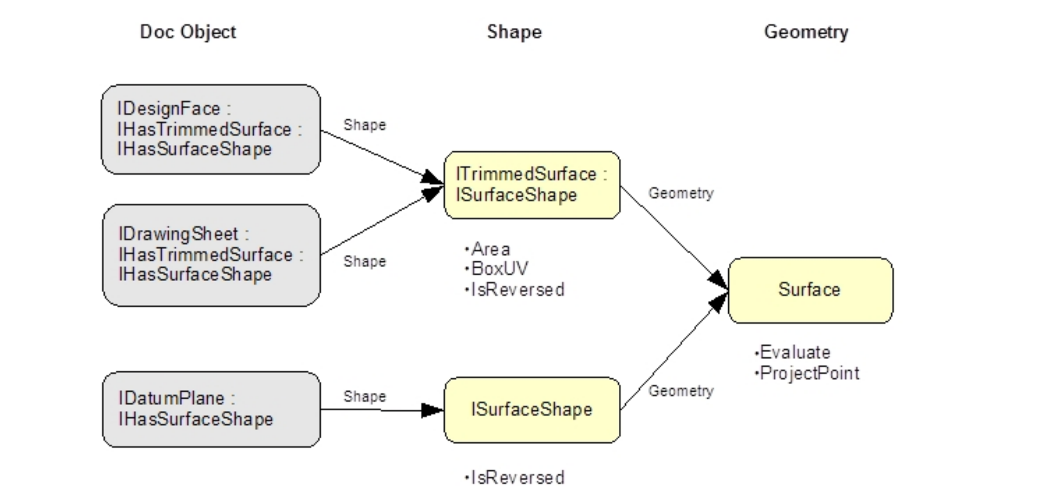
```
public static void Example(IDesignFace desFace) {
 IDesignBody desBody = desFace.Parent;

 // the Edge property returns the edges in the face boundary
 foreach (IDesignEdge desEdge in desFace.Edges) {
  Debug.Assert(desEdge.Parent == desBody);

  // the Faces property returns the faces that meet at this edge
  Debug.Assert(desEdge.Faces.Contains(desFace));
 }
}
```

4.5.6. Shape in General

Shape also applies to doc objects that have untrimmed geometry, e.g. a datum plane. DatumPlane implements IHasSurfaceShape and its Shape, not surprisingly, is a ISurfaceShape. Compare this to DesignFace, which implements IHasTrimmedSurface and has a Shape of type ITrimmedSurface. This parallel path exists because IHasTrimmedSurface is derived form IHasSurfaceShape, and ITrimmedSurface is derived from ISurfaceShape. Therefore, whether the geometry is untrimmed or trimmed, there is always a two step traversal, first to Shape, and then to Geometry:



4.6. Accuracy

This section covers the following topics:

- Linear and Angular Resolution
- Comparing Lengths and Angles
- Comparing XYZ Objects
- Comparing UV Objects
- Comparing Geometry

Published: 2024-01-05T21:40:16.301-05:00

4.6.1. Linear and Angular Resolution

Internally, geometric calculations are performed to machine double precision. When comparing the results of calculations, values should always be compared to within a specific resolution. These resolutions are provided by the Accuracy class:

- Two lengths are considered equal if their difference is less than LinearResolution.
- Two angles are considered equal if their difference is less than AngularResolution.

For example, when ContainsPoint is called to determine whether a point lies in a surface, internally the distance from the surface might be calculated, and the result is true if this distance is less than LinearResolution.

4.6.2. Comparing Lengths and Angles

The Accuracy class provides methods for comparing lengths and angles.

The CompareLengths method takes two arguments, *lengthA* and *lengthB*, and returns an integer result:

- -1 *lengthA* is less than *lengthB*.
- 0 *lengthA* is equal to *lengthB* to within linear resolution.
- +1 *lengthA* is greater than *lengthB*.

This method provides general comparison of two lengths, but for common situations, such as comparing with zero, or testing whether two values are equal, simpler and more readable methods can be used:

```
public static void Example(double lengthA, double lengthB) {
 // same as CompareLengths(lengthA, lengthB) == 0
 bool equalLengths = Accuracy.EqualLengths(lengthA, lengthB);

 // same as CompareLengths(lengthA, 0) == 0
 bool lengthIsZero = Accuracy.LengthIsZero(lengthA);

 // same as CompareLengths(lengthA, 0) > 0
 bool lengthIsPositive = Accuracy.LengthIsPositive(lengthA);

 // same as CompareLengths(lengthA, 0) < 0
 bool lengthIsNegative = Accuracy.LengthIsNegative(lengthA);
}
```

Corresponding methods are provided for angles: CompareAngles, EqualAngles, AngleIsZero, AngleIsPositive, and AngleIsNegative.

4.6.3. Comparing XYZ Objects

The basic XYZ types, Point, Vector, Direction, Box, and Frame, have resolution tests build in, so you can compare objects using the == operator. For example, two points are equal if the distance between them is less than the linear resolution, and two directions are equal if the angle between them is less than then angular resolution.

```
public static void Example(Plane plane, Point point) {
 // project point onto plane
 SurfaceEvaluation eval = plane.ProjectPoint(point);
 Point pointOnPlane = eval.Point;

 // points are the same if less than linear resolution apart
 bool planeContainsPoint = point == pointOnPlane;
```

```
    // ContainsPoint is more efficient, but gives the same result
    Debug.Assert(planeContainsPoint == plane.ContainsPoint(point));
}
```

4.6.4. Comparing UV Objects

The same is not true for the surface parameter UV types, PointUV, VectorUV, DirectionUV, and BoxUV, or for the curve parameter type, Interval. These types do not know whether the parameterization they represent is linear, angular, or some other type. For example, for a plane, the U parameterization is linear, but for a cylinder, the U parameterization is angular. For a NURBS surface, the U parameterization is neither linear nor angular.

Therefore, you should not use the == operator for these types. When comparing parameters, you should use the appropriate length or angle comparison method for each of the U and V values. For NURBS parameterization, angular comparison could be used, but it is safest to evaluate points and compare these instead.

4.6.5. Comparing Geometry

To say that two surfaces or two curves are equal is ambiguous, since there is more than one interpretation of the concept. For example, with two planes:

- They could be coplanar, but the normals could be opposed.
- They could be coplanar, with normals having the same sense, but their frames and hence their parameterization might be different.
- They might be identical in every way.

You should not use the == operator with surface and curve types, since that will only test for reference equality.

Geometry.IsCoincident and ITrimmedCurve.lsCoincident are provided to make comparisons. They only test for coincidence, which means any of the above cases would pass.

4.7. Units

This section covers the following topics:

- System Units and User Units
- Outputting Values
- Inputting Values
- Custom Conversions

4.7.1. System Units and User Units

Internally Discovery works in SI units: meters, kilograms, and seconds. The API also works in SI units.

The user may be working in some other units, but internally the units are still SI units. Conversions are done when values are presented to the user, or input by the user.

The Window class provides conversions between system units (SI units) and user units.

4.7.2. Outputting Values

Window.Units.Length.Format produces a length string that can be presented to the user, which includes the units symbol. As well as performing units conversion, this method also formats the output according to whether the user is working in decimals or fractions.

Window.Units.Angle.Format provides the same functionality for angles.

4.7.3. Inputting Values

To parse a length string entered by the user, you can use Window.Units.Length.TryParse. As well as converting to system units, this method also handles expressions, and values with explicit units stated:

- "(1 + 2) * 3 ^ (3/3 + 3)" = 243
- "1cm + 1 1/2 mm" = 0.0115

Window.Units.Angle.TryParse provides the same functionality for angles.

4.7.4. Custom Conversions

If you need more control over the formatting, you can use Window.Units.Length.ConversionFactor and Window.Units.Length.Symbol for lengths, or Window.Units.Angle.ConversionFactor and Window.Units.Angle.Symbol for angles.

Published: 2024-01-05T21:40:16.301-05:00

5: Release Notes

Available with this release: **Release Notes V24.1**.

5.1. V24.1

The **V24.1** version of the **Discovery APIs** contains the following changes and additions:

Distinction between Core APIs and APIs for customizing the user interface

- In versions preceding v24.1, the *Discovery.Api.Vxx.dll* encompassed the complete set of Discovery APIs. The *Discovery.Api.V241.dll* now exclusively contains the APIs relevant to Core functionality. The user interface customization APIs have been isolated into a distinct .dll named *Discovery.API.V241.UserInterface.dll*. Notably, referencing the *Discovery.API.V241.UserInterface.dll* automatically incorporates a reference to the Core API.

Refactoring namespaces from "Solver" to "Stages"

- As stages and solver fidelity can be defined by add-ins within their custom stages, the following APIs, originally accessible under the **Solver** class, are now accessible through the **Stages** class:
  - **SetStage**
  - **GetCurrentStageIdentifier**
  - **GetFidelity**
  - **SetFidelity**

SIDCustomization

- SID customization has been improved to enable add-ins to incorporate the **Notification hexagon**, unless overridden by add-in-defined functionality. Additionally, it is now feasible to integrate a customized **Selection Button**, similar to the one found in the standard **Discovery** application.

Scripting APIs support new functionality

- Commands within the **DetectTool** namespace for identifying and collecting problematic areas in geometry.
- Commands within the **EnvironmentalProperty** namespace for monitoring the environmental impact of selected materials.
- Commands within the **SubdivisionSurfaces** namespace related to the utilization of subdivision surfaces in geometry.
- Commands and properties within the **Welds** namespace for modeling welding processes.

6.1.1. Copyright and Trademark Information

Disclaimer Notice

U.S. Government Rights

Third-Party Software

Some ANSYS-SpaceClaim products may contain Autodesk® RealDWG by Autodesk, Inc., Copyright © 1998-2010 Autodesk, Inc. All rights reserved. Autodesk, AutoCAD, and Autodesk Inventor are registered trademarks and RealDWG is a trademark of Autodesk, Inc.

CATIA is a registered trademark of Dassault Systèmes.

Portions of this software Copyright © 2013 Trimble. SketchUp is a trademark of Trimble Navigation Limited.

This software is based in part on the work of the Independent JPEG Group.

Portions of this software Copyright © 1999-2006 Intel Corporation. Licensed under the Apache License, Version 2.0. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0

Contains DotNetBar licensed from devcomponents.com.

Portions of this software Copyright © 1988-1997 Sam Leffler and Copyright (c) 1991-1997 Silicon Graphics, Inc.

KeyShot is a trademark of Luxion ApS.

MatWeb is a trademark of Automation Creations, Inc.

2010 Microsoft ® Office System User Interface is licensed from Microsoft Corporation. Direct3D, DirectX, Microsoft PowerPoint, Excel, Windows/Vista/Windows 7/Windows 8/Windows 10 and their respective Start Button designs are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Portions of this software Copyright © 2005 Novell, Inc. (Licensed at http://stuff.mit.edu/afs/athena/software/mono_v3.0/arch/i386_linux26/mono/mcs/class/Managed.Windows.Forms/System.Windows.Forms.RTF/)

Pro/ENGINEER and PTC are registered trademarks of Parametric Technology Corporation.

POV-Ray is available without charge from http://www.pov-ray.org. No charge is being made for a grant of the license to POV-Ray.

POV-Ray License Agreement

DISTRIBUTOR'S LICENCE AGREEMENT

Persistence of Vision Raytracer(tm) (POV-Ray(tm))

13 August 2004

Licensed Versions: Versions 3.5 and 3.6

Please read through the terms and conditions of this license carefully. This is a binding legal agreement between you, the "Distributor" and Persistence of Vision Raytracer Pty. Ltd. ACN 105 891 870 ("POV"), a company incorporated in the state of Victoria, Australia, for the product known as the "Persistence of Vision Raytracer(tm)", also referred to herein as "POV-Ray(tm)". The terms of this agreement are set out at http://www.povray.org/distribution-license.html ("Official Terms"). The Official Terms take precedence over this document to the extent of any inconsistency.

1. INTRODUCTION

1.1. In this agreement, except to the extent the context requires otherwise, the following capitalized terms have the following meanings:

(a) Distribution means:

(i) a single item of a distribution medium, including a CD Rom or DVD Rom, containing software programs and/or data;

(ii) a set of such items;

(iii) a data file in a generally accepted data format from which such an item can be created using generally available standard tools;

(iv) a number of such data files from which a set of such items can be created; or

(v) a data file in a generally accepted data storage format which is an archive of software programs and/or data;

(b) Derived Code means all software which is derived from or is an adaptation of any part of the Software other than a scene file;

(c) Intellectual Rights means:

(i) all copyright, patent, trade mark, trade secret, design, and circuit layout rights;

(ii) all rights to the registration of such rights; and

(iii) all rights of a similar nature which exist anywhere in the world;

(d) Licensed Version means the version set out at the top of this agreement against the heading "Licensed Version" and all minor releases of this version (ie releases of the form x.y.z);

(e) POV Associate means any person associated directly or indirectly with POV whether as a director, officer, employee, subcontractor, agent, representative, consultant, licensee or otherwise;

(f) Modification Terms means the most recent version from time to time of the document of that name made available from the Site (g) Revocation List means the list of that name linked to from the Official Terms;

(h) Site means www.povray.org;

(i) Software means the Licensed Version of the Persistence of Vision Raytracer(tm) (also known as POV-Ray(tm)) (including all POV-Ray program source files, executable (binary) files, scene files, documentation files, help files, bitmaps and other POV-Ray files associated with the Licensed Version) in a form made available by

POV on the Site;

(j) User Licence means the most recent version from time to time of the document of that name made available from the Site.

2. OPEN SOURCE DISTRIBUTIONS

2.1. In return for the Distributor agreeing to be bound by the terms of this agreement, POV grants the Distributor permission to make a copy of the Software by including the Software in a generally recognised Distribution of a recognised operating system where the kernel of that operating system is made available under licensing terms:

(a) which are approved by the Open Source Initiative (www.opensource.org) as complying with the "Open Source Definition" put forward by the Open Source Initiative; or

(b) which comply with the "free software definition" of the Free Software Foundation (www.fsf.org). 2.2. As at June 2004, and without limiting the generality of the term, each of the following is a "generally recognised Distribution" for the purposes of clause 2.1: Debian, Red Hat (Enterprise and Fedora), SuSE, Mandrake, Xandros, Gentoo and Knoppix Linux distributions, and officially authorized distributions of the FreeBSD, OpenBSD, and NetBSD projects.

2.3. Clause 2.1 also applies to the Software being included in the above distributions 'package' and 'ports' systems, where such exist;

2.4. Where the Distributor reproduces the Software in accordance with clause 2.1:

(a) the Distributor may rename, reorganise or repackage (without omission) the files comprising the Software where such renaming, reorganisation or repackaging is necessary to conform to the naming or organisation scheme of the target operating environment of the Distribution or of an established package management system of the target operating environment of the Distribution; and (b) the Distributor must not otherwise rename, reorganise or repackage the Software.

3. DISTRIBUTION LICENCE

3.1. Subject to the terms and conditions of this agreement, and in return for Distributor agreeing to be bound by the terms of this agreement, POV grants the Distributor permission to make a copy of the Software in any of the following circumstances:(a) in the course of providing a mirror of the POV-Ray Site (or part of it), which is made available generally over the internet to each person without requiring that person to identify themselves and without any other restriction other than restrictions designed to manage traffic flows;

(b) by placing it on a local area network accessible only by persons authorized by the Distributor whilst on the Distributor's premises;

(c) where that copy is provided to a staff member or student enrolled at a recognised educational institution;

(d) by including the Software as part of a Distribution where:

(i) neither the primary nor a substantial purpose of the distribution of the Distribution is the distribution of the Software. That is, the distribution of the Software

is merely incidental to the distribution of the Distribution; and

(ii) if the Software was not included in the Distribution, the remaining software and data included within the Distribution would continue to function effectively and

according to its advertised or intended purpose;

(e) by including the Software as part of a Distribution where:

(i) there is no data, program or other files apart from the Software on the Distribution;

(ii) the Distribution is distributed by a person to another person known to that person; or

(iii) the Distributor has obtained explicit written authority from POV to perform the distribution, citing this clause number, prior to the reproduction being

made.

3.2. In each case where the Distributor makes a copy of the Software in accordance with clause 3.1, the Distributor must, unless no payment or other consideration of any type is received by Distributor in relation to the Distribution:

(a) ensure that each person who receives a copy of the Software from the Distributor is aware prior to acquiring that copy:

(i) of the full name and contact details of the Distributor, including the Distributor's web site, street address, mail address, and working email address;

(ii) that the Software is available without charge from the Site;

(iii) that no charge is being made for the granting of a licence over the Software.

(b) include a copy of the User Licence and this Distribution License with the copy of the Software. These licences must be stored in the same subdirectory on the distribution medium as the Software and named in such a way as to prominently identify their purpose;

3.3. The Distributor must not rename, reorganise or repackage any of the files comprising the Software without the prior written authority of POV.

3.4. Except as explicitly set out in this agreement, nothing in this agreement permits Distributor to make any modification to any part of the Software.

4. RESTRICTIONS ON DISTRIBUTION

4.1. Nothing in this agreement gives the Distributor: (a) any ability to grant any licence in respect of the use of the Software or any part of it to any person;

(b) any rights or permissions in respect of, including rights or permissions to distribute or permit the use of, any Derived Code;

(c) any right to bundle a copy of the Software (or part thereof), whether or not as part of a Distribution, with any other items, including books and magazines. POV may, in response to a request, by notice in writing and in its absolute discretion, permit such bundling on a case by case basis. This clause 4.1(c) does not apply to Distributions permitted under clause 2;

(d) any right, permission or authorisation to infringe any Intellectual Right held by any third party.

4.2. Distributor may charge a fee for the making or the provision of a copy of the Software.

4.3. Where the making, or the provision, of a copy of the Software is authorised under the terms of clause 3 but not under those of clause 2 of this agreement, the total of all fees charged in relation to such making or provision and including all fees (including shipping and handling fees) which are charged in respect

of any software, hardware or other material provided in conjunction with or in any manner which is reasonably connected with the making, or the provision, of a copy of the Software must not exceed the reasonable costs incurred by the Distributor in making the reproduction, or in the provision, of that copy for which the fee

is charged.

4.4. Notwithstanding anything else in this agreement, nothing in this agreement permits the reproduction of any part of the Software by, or on behalf of:

(a) Any person currently listed on the Revocation List from time to time;

(b) Any related body corporate (as that term is defined in section 50 of the Corporations Law 2001 (Cth)) of any person referred to in clause 4.4(a);

(c) Any person in the course of preparing any publication in any format (including books, magazines, CD Roms or on the internet) for any of the persons identified in paragraph (a);

(d) Any person who is, or has been, in breach of this Agreement and that breach has not been waived in writing signed by POV; or

(e) Any person to whom POV has sent a notice in writing or by email stating that that person may not distribute the Software.

4.5. From the day two years after a version of the Software more recent than the Licensed Version is made available by POV on the Site clause 3 only permits reproduction of the Software where the Distributor ensures that each recipient of such a reproduction is aware, prior to obtaining that reproduction, that that reproduction of the Software is an old version of the Software and that a more recent version of the Software is available from the Site.

5. COPYRIGHT AND NO LITIGATION

5.1. Copyright subsists in the Software and is protected by Australian and international copyright laws.

5.2. Nothing in this agreement gives Distributor any rights in respect of any Intellectual Rights in respect of the Software or which are held by or on behalf of POV. Distributor acknowledges that it does not acquire any rights in respect of such Intellectual Rights.

5.3. Distributor acknowledges that if it performs out any act in respect of the Software without the permission of POV it will be liable to POV for all damages POV may suffer (and which Distributor acknowledges it may suffer) as well as statutory damages to the maximum extent permitted by law and that it may also be liable to

criminal prosecution.

5.4. Distributor must not commence any action against any person alleging that the Software or the use or distribution of the Software infringes any rights, including Intellectual Rights of the Distributor or of any other person. If Distributor provides one or more copies of the Software to any other person in accordance with the agreement, Distributor waives all rights it has, or may have in the future, to bring any action, directly or indirectly, against any person to the extent that such an action relates to an infringement of any rights, including Intellectual Rights of any person in any way arising from, or in relation to, the use, or distribution, (including through the authorisation of such use or distribution) of:(a) the Software;

(b) any earlier or later version of the Software; or

(c) any other software to the extent it incorporates elements of the software referred to in paragraphs (a) or (b) of this clause

5.4.

6. DISCLAIMER OF WARRANTY

6.1. To the extent permitted by law, all implied terms and conditions are excluded from this agreement. Where a term or condition is implied into this agreement and that term cannot be legally excluded, that term has effect as a term or condition of this agreement. However, to the extent permitted by law, the liability

of POV for a breach of such an implied term or condition is limited to the fullest extent permitted by law.

6.2. To the extent permitted by law, this Software is provided on an "AS IS" basis, without warranty of any kind, express or implied, including without limitation, any implied warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property of any third party. The Software has inherent limitations including design faults and programming bugs.

6.3. The entire risk as to the quality and performance of the Software is borne by Distributor, and it is Distributor's responsibility to ensure that the Software fulfils Distributor's requirements prior to using it in any manner (other than testing it for the purposes of this paragraph in a non-critical and non-production environment), and prior to distributing it in any fashion.

6.4. This clause 6 is an essential and material term of, and cannot be severed from, this agreement. If Distributor does not or cannot agree to be bound by this clause, or if it is unenforceable, then Distributor must not, at any time, make any reproductions of the Software under this agreement and this agreement gives the

Distributor no rights to make any reproductions of any part of the Software.

7. NO LIABILITY

7.1. When you distribute or use the Software you acknowledge and accept that you do so at your sole risk. Distributor agrees that under no circumstances will it have any claim against POV or any POV Associate for any loss, damages, harm, injury, expense, work stoppage, loss of business information, business interruption,

computer failure or malfunction which may be suffered by you or by any third party from any cause whatsoever, howsoever arising, in connection with your use or distribution of the Software even where POV was aware, or ought to have been aware, of the potential of such loss.

7.2. Neither POV nor any POV Associate has any liability to Distributor for any indirect, general, special, incidental, punitive and/or consequential damages arising as a result of a breach of this agreement by POV or which arises in any way related to the Software or the exercise of a licence granted to Distributor under this

agreement.

7.3. POV's total aggregate liability to the Distributor for all loss or damage arising in any way related to this agreement is limited to the lesser of: (a) AU$100, and (b) the amount received by POV from Distributor as payment for the grant of a licence under this agreement.

7.4. Distributor must bring any action against POV in any way related to this agreement or the Software within 3 months of the cause of action first arising. Distributor waives any right it has to bring any action against POV and releases POV from all liability in respect of a cause of action if initiating process in relation to that action is not served on POV within 3 months of the cause of action arising. Where a particular set of facts give rise to more than one cause of action this clause 7.4 applies as if all such causes of action arise at the time the first such cause of action arises.

7.5. This clause 7 is an essential and material term of, and cannot be severed from, this agreement. If Distributor does not or cannot agree to be bound by this clause, or if it is unenforceable, then Distributor must not, at any time, make any reproductions of the Software under this agreement and this agreement gives the Distributor no rights to make any reproductions of any part of the Software.

8. INDEMNITY

8.1. Distributor indemnifies POV and each POV Associate and holds each of them harmless against all claims which arise from any loss, damages, harm, injury, expense, work stoppage, loss of business information, business interruption, computer failure or malfunction, which may be suffered by Distributor or any other

party whatsoever as a consequence of:

(a) any act or omission of POV and/or any POV Associate, whether negligent or not;

(b) Distributor's use and/or distribution of the Software; or

Published: 2024-01-05T21:40:16.301-05:00

(c) any other cause whatsoever, howsoever arising, in connection with the Software. This clause 8 is binding on Distributor's estate, heirs, executors, legal successors, administrators, parents and/or guardians.

8.2. Distributor indemnifies POV, each POV Associate and each of the authors of any part of the Software against all loss and damage and for every other consequence flowing from any breach by Distributor of any Intellectual Right held by POV.

8.3. This clause 8 constitutes an essential and material term of, and cannot be severed from, this agreement. If Distributor does not or cannot agree to be bound by this clause, or if it is unenforceable, then Distributor must not, at any time, make any reproductions of the Software under this agreement and this agreement gives the Distributor no rights to make any reproductions of any part of the Software.

9. HIGH RISK ACTIVITIES

9.1. This Software and the output produced by this Software is not fault-tolerant and is not designed, manufactured or intended for use as on-line control equipment in hazardous environments requiring fail-safe performance, in which the failure of the Software could lead or directly or indirectly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). POV specifically disclaims all express or implied warranty of fitness for High Risk Activities and, notwithstanding any other term of this agreement, explicitly prohibits the use or distribution of the Software for such purposes.

10. ENDORSEMENT PROHIBITION

10.1. Distributor must not, without explicit written permission from POV, claim or imply in any way that:

(a) POV or any POV Associate officially endorses or supports the Distributor or any product (such as CD, book, or magazine) associated with the Distributor or any reproduction of the Software made in accordance with this agreement; or(b) POV derives any benefit from any reproduction made in accordance with this agreement.

11. TRADEMARKS

11.1. "POV-Ray(tm)", "Persistence of Vision Raytracer(tm)" and "POV-Team(tm)" are trademarks of Persistence of Vision Raytracer Pty. Ltd. Any other trademarks referred to in this agreement are the property of their respective holders. Distributor must not use, apply for, or register anywhere in the world, any word, name

(including domain names), trade mark or device which is substantially identical or deceptively or confusingly similar to any of Persistence of Vision Raytracer Pty. Ltd's trade marks.

12. MISCELLANEOUS

12.1. The Official Terms, including those documents incorporated by reference into the Official Terms, and the Modification Terms constitute the entire agreement between the parties relating to the distribution of the Software and, except where stated to the contrary in writing signed by POV, supersedes all previous

negotiations and correspondence in relation to it.

12.2. POV may modify this agreement at any time by making a revised licence available from the Site at http://www.povray.org/distribution-license.html.

This agreement is modified by replacing the terms in this agreement with those of the revised licence from the time that the revised licence is so made available. It is your responsibility to ensure that you have read and agreed to the current version of this agreement prior to distributing the Software.

12.3. Except where explicitly stated otherwise herein, if any provision of this Agreement is found to be invalid or unenforceable, the invalidity or unenforceability of such provision shall not affect the other provisions of this agreement, and all provisions not affected by such invalidity or unenforceability shall remain in

full force and effect. In such cases Distributor agrees to attempt to substitute for each invalid or unenforceable provision a valid or enforceable provision which achieves to the greatest extent possible, the objectives and intention of the invalid or unenforceable provision.

12.4. A waiver of a right under this agreement is not effective unless given in writing signed by the party granting that waiver. Unless otherwise stipulated in the waiver, a waiver is only effective in respect of the circumstances in which it is given and is not a waiver in respect of any other rights or a waiver in respect of future rights or actions.

12.5. The validity and interpretation of this agreement is governed by the laws in force in the State of Victoria, Australia. Distributor submits to the exclusive jurisdiction of the courts of that State and courts located within that State exercising federal jurisdiction.

12.6. References in this agreement to "written" and "writing" mean on paper or by fax and expressly exclude email and other forms of electronic communication.

13. CONTACT INFORMATION

13.1. This clause 13 does not form part of the agreement. License inquiries can be made via email; please use the following address (but see 13.2 below prior to emailing) : team-coord-[three-letter month]-[four-digit year]@povray org. for example, team-coord-jun-2004@povray.org should be used if at the time you send the email it is the month of June 2004. The changing email addresses are necessary to combat spam. Old email addresses may be deleted at POV's discretion.

13.2. Note that the address referred to in 13.1 may change for reasons other than those referred to in that clause; please check the current version of this document at http://www.povray.org/distribution-license.html. for the current address. Your inability or failure to contact us is no excuse for violating the licence.

13.3. Do NOT send any email attachments of any sort other than by prior arrangement. Do not send email in HTML format. EMAIL MESSAGES INCLUDING ATTACHMENTS WILL BE DELETED UNREAD.

13.4. The following postal address is only for official license business. Please note that it is preferred that initial queries about licensing be made via email; postal mail should only be used when email is not possible, or when written documents are being exchanged by prior arrangement. While it is unlikely this address will change in the short term it would be advisable to check http://www.povray.org/distribution-license.html for the current one prior to sending postal mail.

Persistence of Vision Raytracer Pty. Ltd.

PO Box 407

Williamstown,

Victoria 3016

Australia

POV-Ray Licence Agreement

GENERAL LICENSE AGREEMENT

FOR PERSONAL USE

Persistence of Vision Ray Tracer (POV-Ray)

Version 3.6 License and Terms & Conditions of Use

version of 1 February 2005

(also known as POVLEGAL.DOC)

Please read through the terms and conditions of this license carefully. This license is a binding legal agreement between you, the 'User' (an individual or single entity) and Persistence of Vision Raytracer Pty. Ltd. ACN 105 891 870 (herein also referred to as the "Company"), a company incorporated in the state of Victoria, Australia, for the product known as the "Persistence of Vision Ray Tracer", also referred to herein as 'POV-Ray'.

YOUR ATTENTION IS PARTICULARLY DRAWN TO THE DISCLAIMER OF WARRANTY AND NO LIABILITY AND INDEMNITY PROVISIONS. TO USE THE PERSISTENCE OF VISION RAY TRACER ("POV-RAY") YOU MUST AGREE TO BE BOUND BY THE TERMS AND CONDITIONS SET OUT IN THIS DOCUMENT. IF YOU DO NOT AGREE TO ALL THE TERMS AND CONDITIONS OF USE OF POV-RAY SET OUT IN THIS LICENSE AGREEMENT, OR IF SUCH TERMS AND CONDITIONS ARE NOT BINDING ON YOU IN YOUR JURISDICTION, THEN YOU MAY NOT USE POV-RAY IN ANY MANNER. THIS GENERAL

LICENSE AGREEMENT MUST ACCOMPANY ALL POV-RAY FILES WHETHER IN THEIR OFFICIAL OR CUSTOM VERSION FORM. IT MAY NOT BE REMOVED OR MODIFIED. THIS GENERAL LICENSE AGREEMENT GOVERNS THE USE OF

POV-RAY WORLDWIDE. THIS DOCUMENT SUPERSEDES AND REPLACES ALL PREVIOUS GENERAL LICENSES.

INTRODUCTION

This document pertains to the use of the Persistence of Vision Ray Tracer (also known as POV-Ray). It applies to all POV-Ray program source files, executable (binary) files, scene files, documentation files, help files, bitmaps and other POV-Ray files contained in official Company archives, whether in full or any part thereof, and are herein referred to as the "Software". The Company reserves the right to revise these rules in future versions and to make additional rules to address new circumstances at any time. Such rules, when made, will be posted in a revised license file, the latest version of which is available from the Company website at

http://www.povray.org/povlegal.html.

USAGE PROVISIONS

Subject to the terms and conditions of this agreement, permission is granted to the User to use the Software and its associated files to create and render images. The creator of a scene file retains all rights to any scene files they create, and any images generated by the Software from them. Subject to the other terms of this license, the User is permitted to use the Software in a profit-making enterprise, provided such profit arises primarily from use of the Software and not from distribution of the Software or a work including the Software in whole or part.

Please refer to http://www.povray.org/povlegal.html for licenses covering distribution of the Software and works including the Software. The User is also granted the right to use the scene files, fonts, bitmaps, and include files distributed in the INCLUDE and SCENES\INCDEMO sub-directories of the Software in their own scenes. Such permission does not extend to any other files in the SCENES directory or its sub-directories. The SCENES files are for the User's enjoyment and education but may not be the basis of any derivative works unless the file in question explicitly grants permission to do such.

This licence does not grant any right of re-distribution or use in any manner other than the above. The Company has separate license documents that apply to other uses (such as re-distribution via the internet or on CD) ; please visit http://www.povray.org/povlegal.html for links to these. In particular you are advised that the sale, lease, or rental of the Software in any form without written authority from the Company is explicitly prohibited. Notwithstanding anything in the balance of this licence agreement, nothing in this licence agreement permits the installation or use of the Software in conjunction with any product (including software) produced or distributed by any party who is, or has been, in violation of this licence agreement or of the distribution licence (http://www.povray.org/distribution-license.html)

(or any earlier or later versions of those documents) unless:

a. the Company has explicitly released that party in writing from the consequences of their non compliance; or

b. both of the following are true:

i. the installation or use of the Software is without the User being aware of the abovementioned violation; and

ii. the installation or use of the Software is not a result (whether direct or indirect) of any request or action of the abovementioned party (or any of its products), any agent of that party (or any of their products), or any person(s) involved in supplying any such product to the User.

COPYRIGHT

Copyright © 1991-2003, Persistence of Vision Team.

Copyright © 2003-2004, Persistence of Vision Raytracer Pty. Ltd.

Windows version Copyright © 1996-2003, Christopher Cason.

Copyright subsists in this Software which is protected by Australian and international copyright laws. The Software is NOT PUBLIC DOMAIN. Nothing in this agreement shall give you any rights in respect of the intellectual property of the Company and you acknowledge that you do not acquire any rights in respect of such intellectual property rights. You acknowledge that the Software is the valuable intellectual property of the Company and that if you use, modify or distribute the Software for unauthorized purposes or in an unauthorized manner (or cause or allow the forgoing to occur), you will be liable to the Company for any damages it may suffer (and which you acknowledge it may suffer) as well as statutory damages to the maximum extent permitted by law and also that you may be liable to

criminal prosecution. You indemnify the Company and the authors of the Software for every single consequence flowing from the aforementioned events.

DISCLAIMER OF WARRANTY

express or implied, including without limitation, any implied warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property of any third party. This Software has inherent limitations including design faults and programming bugs. The entire risk as to the quality and performance of the Software is borne by you, and it is your responsibility to ensure that it does what you require it to do prior to using it for any purpose (other than testing it), and prior to distributing it in any fashion. Should the Software prove defective, you agree that you alone assume the entire cost resulting in any way from such defect.

This disclaimer of warranty constitutes an essential and material term of this agreement. If you do not or cannot accept this, or if it is unenforceable in your jurisdiction, then you may not use the Software in any manner.

NO LIABILITY

When you use the Software you acknowledge and accept that you do so at your sole risk. You agree that under no circumstances shall you have any claim against the Company or anyone associated directly or indirectly with the Company whether as employee, subcontractor, agent, representative, consultant, licensee or otherwise ("Company Associates") for any loss, damages, harm, injury, expense, work stoppage, loss of business information, business interruption, computer failure or malfunction which may be suffered by you or by any third party from any cause whatsoever, howsoever arising, in connection with your use or distribution of the Software even where the Company were aware, or ought to have been aware, of the potential of such loss. Damages referred to above shall include direct, indirect, general, special, incidental, punitive and/or consequential. This disclaimer of liability constitutes an essential and material term of this agreement. If you do not or cannot accept this, or if it is unenforceable in your jurisdiction, then you may not use the Software.

INDEMNITY

You indemnify the Company and Company Associates and hold them harmless against any claims which may arise from any loss, damages, harm, injury, expense, work stoppage, loss of business information, business interruption, computer failure or malfunction, which may be suffered by you or any other party whatsoever as a consequence of any act or omission of the Company and/or Company Associates, whether negligent or not, arising out of your use and/or distribution of the Software, or from any other cause whatsoever, howsoever arising, in connection with the Software. These provisions are binding on your estate, heirs, executors, legal successors, administrators, parents and/or guardians.

This indemnification constitutes an essential and material term of this agreement. If you do not or cannot accept this, or if it is unenforceable in your jurisdiction, then you may not use the Software.

HIGH RISK ACTIVITIES

This Software and the output produced by this Software is not fault-tolerant and is not designed, manufactured or intended for use as on-line control equipment in hazardous environments requiring fail-safe performance, in which the failure of the Software could lead or directly or indirectly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). The Company specifically disclaims any express or implied warranty of fitness for High Risk Activities and explicitly prohibits the use of the Software for such purposes.

CRYPTOGRAPHIC SIGNING OF DOCUMENTS

Changes to this Agreement and documents issued under its authority may be cryptographically signed by the POV-Ray Team Co-ordinator's private PGP key.

In the absence of evidence to the contrary, such documents shall be considered, under the terms of this Agreement, to be authentic provided the signature is valid. The master copy of this Agreement at http://www.povray.org/povlegal.html will also be signed by the current version of the team-coordinator's key.

The public key for the POV-Ray Team-coordinator can be retrieved from the location https://secure.povray.org/keys/. The current fingerprint for it is

B4DD 932A C080 C3A3 6EA2 9952 DB04 4A74 9901 4518.

MISCELLANEOUS

This Agreement constitutes the complete agreement concerning this license. Any changes to this agreement must be in writing and may take the form of

notifications by the Company to you, or through posting notifications on the Company website. THE USE OF THIS SOFTWARE BY ANY PERSON OR ENTITY IS

EXPRESSLY MADE CONDITIONAL ON THEIR ACCEPTANCE OF THE TERMS SET FORTH HEREIN. Except where explicitly stated otherwise herein, if any provision of this

Agreement is found to be invalid or unenforceable, the invalidity or unenforceability of such provision shall not affect the other provisions of this agreement, and all provisions not affected by such invalidity or unenforceability shall remain in full force and effect. In such cases you agree to attempt to substitute for each invalid or unenforceable provision a valid or enforceable provision which achieves to the greatest extent possible, the objectives and intention of the invalid or unenforceable

provision. The validity and interpretation of this agreement will be governed by the laws of Australia in the state of Victoria (except for conflict of law provisions).

CONTACT INFORMATION

License inquiries can be made via email; please use the following address (but see below prior to emailing) : team-coord-[three-letter month]-[four-digit year]@povray.org for example, team-coord-jun-2004@povray.org should be used if at the time you send the email it is the month of June 2004. The changing email addresses are necessary to combat spam and email viruses. Old email addresses may be deleted at our discretion.

Note that the above address may change for reasons other than that given above; please check the version of this document at http://www.povray.org/povlegal.html for the current address. Note that your inability or failure to contact us for any reason is not an excuse for violating this licence.

Do NOT send any attachments of any sort other than by prior arrangement.

EMAIL MESSAGES INCLUDING ATTACHMENTS WILL BE DELETED UNREAD.

The following postal address is only for official license business. Please note that it is preferred that initial queries about licensing be made via email ; postal mail should only be used when email is not possible, or when written documents are being exchanged by prior arrangement.

Persistence of Vision Raytracer Pty. Ltd.

PO Box 407

Williamstown,

Victoria 3016

Australia

Portions of this software are owned by Siemens PLM © 1986-2013. All Rights Reserved. Parasolid, Unigraphics, and SolidEdge are registered trademarks and JT is a trademark of Siemens Product Lifecycle Management Software, Inc.

SolidWorks is a registered trademark of SolidWorks Corporation.

Portions of this software are owned by Spatial Corp. © 1986-2013. All Rights Reserved. ACIS, SAT and SAB are registered trademarks of Spatial Corp.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition,

"Submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License.

You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution

intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely

responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS