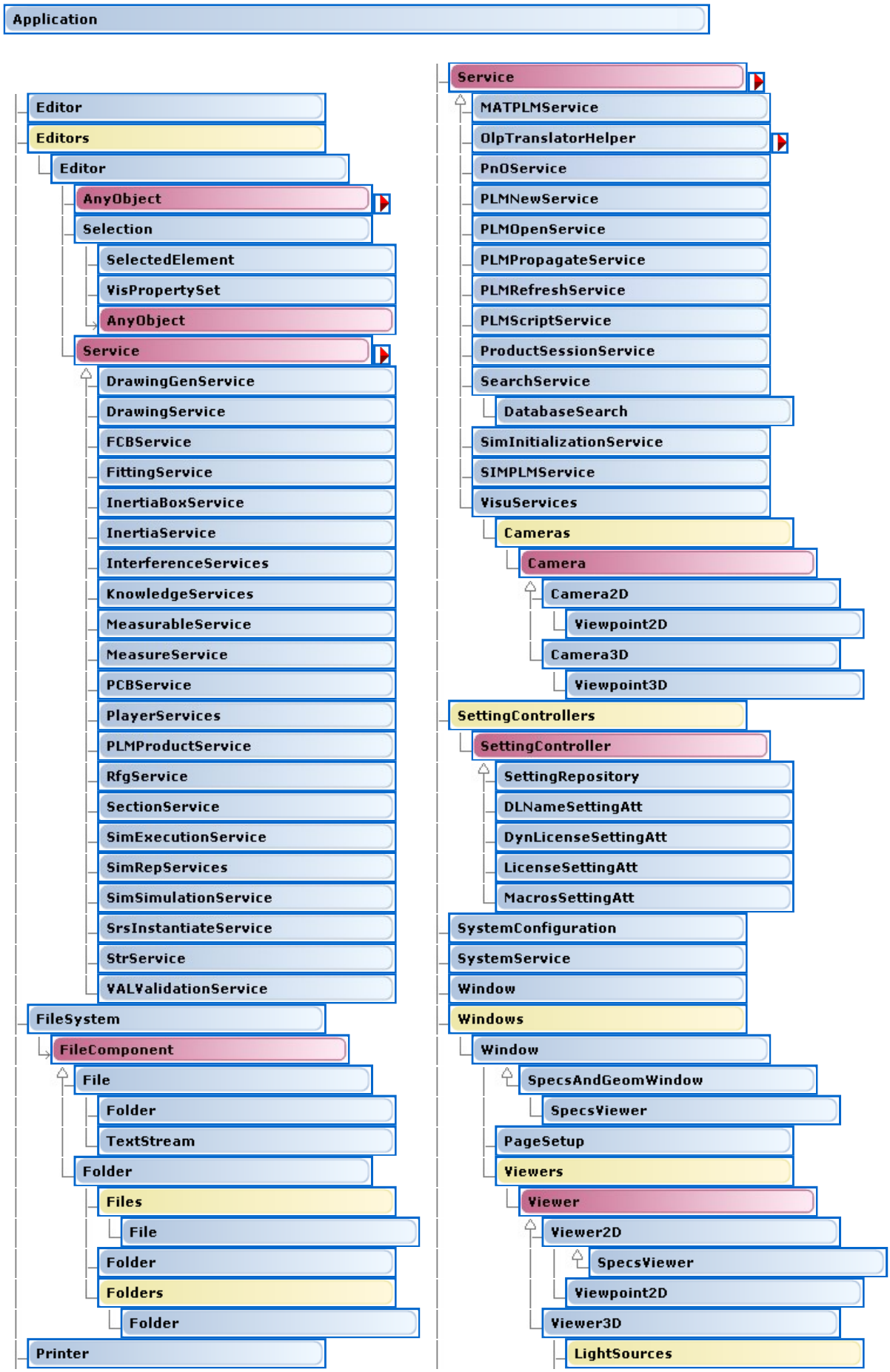
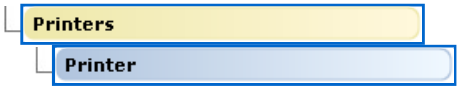
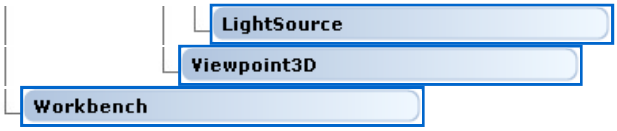
Foundation Object Model Map

See Also Legend

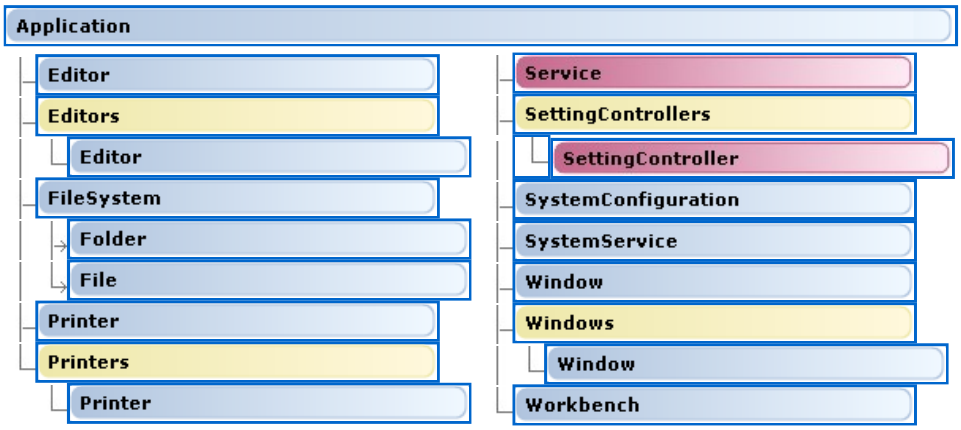






Application Object

See Also Legend Use Cases Properties Methods



Represents the entire current application and its frame window.

The **Application** object is the root object for all the other objects you can use and access from scripts. It corresponds to both the application itself and to its frame window. It directly aggregates four collections:

1. The editor collection represented by the **Editors** collection object.
   This collection contains all the editors currently managed by the application. Use the **Editors** property to return the **Editors** collection object.
2. The printer collection represented by the **Printers** collection object.
   This collection contains all the printers accessible from the application. Use the **Printers** property to return the **Printers** collection object.
3. The setting controller collection represented by the **SettingControllers** collection object.
   This collection contains a generic setting controller for each setting repository managed using an XML file, and some specific setting controllers. Use the **SettingControllers** property to return the **SettingControllers** collection object.
4. The window collection represented by the **Windows** collection object.
   This collection contains all the windows currently opened by the application, each window displaying objects managed by an editor. Use the **Windows** property to return the **Windows** collection object.

In addition to these four collections, the **Application** object also aggregates:

- An **Editor** object: at any time, an **Editor** object among those managed by the application is active and associated with the objects displayed in the active window, that is, the window the end-user is currently working in. This active editor gives access to the objects visible in the active window and to the set of operations that can be applied to those objects. This editor sets the available menus and toolbars that make it possible to edit the objects it controls, according to the root object type. Use the **ActiveEditor** property to return the active editor.
- A **FileSystem** object that enables you to manipulate folders and files. Use the **FileSystem** property to return the file system object
- A **Printer** object: the active printer. Use the **ActivePrinter** property to return the active printer
- A **Service** object: object-independent session-level **Service** objects can be retrieved from the **Application** object thanks to the **GetSessionService** method. A **Service** object has methods that can execute whatever the objects managed by the active editor. For example, creating and running a search in the database does not depend on the objects managed by the active editor in the active window. It is made possible using a method of the session-level **SearchService** object.
- A **SystemConfiguration** object, providing access to system or configuration dependent resources, such as the operating system, the current version, release, and service pack, and the available licensed products. Use the **SystemConfiguration** property to return the **SystemConfiguration** object
- A **SystemService** object, providing information about the system environment, and services to run a script or a process. For example, the **Environ** method retrieves the value of a given environment variable. Note that the **SystemService** object is not a **Service** object, that is, is retrieved thanks to the **SystemService** property of the **Application** object, while a **Service** object is retrieved using the **GetSessionService** method.
- A **Window** object: the active window. Use the **ActiveWindow** property to return the active window.
- A **Workbench** object that you can retrieve thanks to its workbench identifier using the **GetWorkbenchId** method.

The **Application** object has other properties, such as **LocalCache** to return the local cache path, and **CacheSize** to return the local cache size. As the application represents the frame window, you can retrieve or set the frame dimensions and location

using the properties **Width**, **Height**, **Left**, and **Top** respectively, with values expressed in screen pixels.

Remarks

When you create or use macros for in-process access, always refer to the **Application** object as **CATIA**. For example, to retrieve the **Windows** collection from the **Application** object, refer to the **Application** object as follows:

```
Dim cWins As Windows
Set cWins = CATIA.Windows
```

Using the Application Object

Use the **ActiveEditor** property to retrieve the active editor

```
Dim oActEditor As Editor
Set oActEditor = CATIA.ActiveEditor
```

Use the **GetSessionService** method to return a **Service** object. This example returns the **VisuServices** object.

```
Dim oService As Service
Set oService = CATIA.GetSessionService("VisuServices")
```

Cameras Collection Object

See Also Legend Use Cases Properties Methods

Represents a collection of cameras.

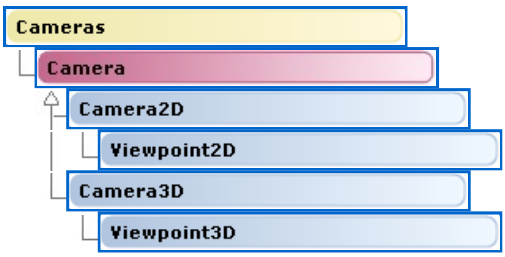
Returning the Cameras Collection Object

Use the **Cameras** property of the **VisuServices** object to return the **Cameras** collection object.

```
Dim cCameras As Cameras
Set cCameras = oVisuServices.Cameras
```

Camera Object

See Also Legend Use Cases Properties Methods



Represents a camera.

A camera is the persistent form of a viewpoint.

You can create a camera from the current viewpoint using the **NewCamera** method of the **Viewer** object.

The camera name is assigned by CATIA: **Camera00**, **Camera01**, and so forth, and the created camera is placed at the end of the camera collection and is displayed in the Named Views dialog box of the View menu.

You can rename the cameras you create with your own names using the **Name** property.

A created camera can then be assigned to a viewer to make its own viewpoint change to this of the camera.

Two kinds of cameras exist: the **Camera2D** object for 2D viewpoints, that is for Drawing Representation objects, and the **Camera3D** object for 3D viewpoints representing the real world, that is for 3D Shape Representation and Product objects.

A **Camera2D** object stores a **Viewpoint2D** object and a **Camera3D** object stores a **Viewpoint3D** object.

Creating a Camera

Use the **NewCamera** method of the **Viewer** object to create a **Camera** object.

```
Dim oViewer As Viewer
Set oViewer = CATIA.ActiveWindows.ActiveViewer

Dim oCamera As Camera
Set oCamera = oViewer.NewCamera()
```

The created camera is added to the camera collection.

Note : Created Camera for 3DPart scenario is no more located under the 3DShape.

Retrieving an Existing Camera

Use the **VisuServices** object to retrieve a **Cameras** collection object.

```
Dim oVisuServices As VisuServices
Set oVisuServices = CATIA.GetSessionService("VisuServices")

Dim cCameras As Cameras
Set cCameras = oVisuServices.Cameras
```

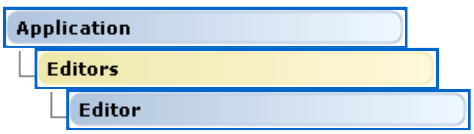
From this collection, you can either retrieve a built-in camera representing a standard view, or a camera you have created.

```
Dim oCamera As Camera
Set oCamera = cCameras.Item("Camera05")
```

This retrieves the front camera.

Editors Collection Object

See Also Legend Use Cases Properties Methods



A collection of the **Editor** objects that are currently held by the application.

Retrieving the Editors Collection Object

Use the **Editors** property of the **Application** object to return the **Editors** collection object.

```
Dim cEditors As Editors
Set cEditors = CATIA.Editors
```

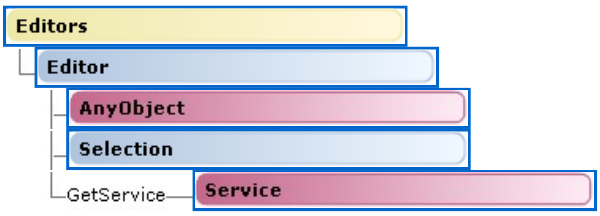
Using the Editors Collection Object

Use the **Item** method of the **Editors** collection object to return an editor among the ones hold by the **Application** object.

```
Dim oEditor As Editor
Set oEditor = CATIA.Editors.Item(2)
```

Editor Object

See Also Legend Use Cases Properties Methods



Represents an editor.

In the Model View Controller paradigm, the editor plays the Controller role. The editor federates all the objects that can be interactively edited in the same window. It holds the current workbench and thus maintains the list of all the commands that can be launched from menus and toolbars when this window is active to edit the objects it controls according to the root object PLM type.

The active editor is the editor associated with the current window, whatever the active object in that window.

Retrieving the Editor Object

Use the **ActiveEditor** property of the **Application** object to return the active editor.

```
Dim oActiveEditor As Editor
Set oActiveEditor = CATIA.ActiveEditor
```

Use the **Item** method of the **Editors** collection to return an editor among the ones hold by the **Application** object.

```
Dim oEditor As Editor
Set oEditor = CATIA.Editors.Item(2)
```

Using the Editor Object

Use the **ActiveObject** property to return the active object associated with the active **Editor** object. The active object is the root of the data being currently edited. Depending on this data, it can be a **Part** object, a **DrawingRoot** object, or a **VPMReference** object.

```
Dim oActiveObject As AnyObject
Set oActiveObject = CATIA.ActiveEditor.ActiveObject
```

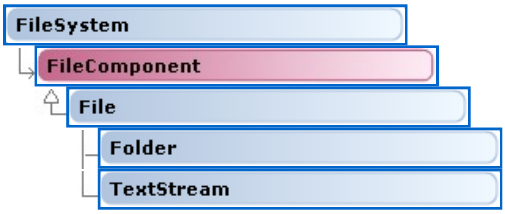
Use the **Selection** property to return the **Selection** object associated with the active **Editor** object.

```
Dim oSelection As Selection
Set oSelection = CATIA.ActiveEditor.Selection
```

Use the **GetService** method to return a **Service** object applying to the Editor. Refer to the Service Object.

File Object

See Also Legend Use Cases Properties Methods



Represents a file.

The **File** object is operating system independent. Using it instead of those available with the platform you use makes your macro portable.

The **File** object derives from the **FileComponent** object from which it inherits the **Path** property. It aggregates:

- A **Folder** object: its parent folder accessible using the **ParentFolder** property also inherited from the **FileComponent** object
- A **TextStream** object that represents the file contents as a text stream, returned thanks to the **OpenAsTextStream** method.

Retrieving the File Object

Use the **GetFile** method of the **FileSystem** object, returned thanks to the **FileSystem** property of the **Application** object, to return an existing file.

```
Dim oFile As File
Set oFile = CATIA.FileSystem.GetFile("C:\tmp\myFile.txt")
```

In the same way, use the other properties and methods of the **FileSystem** object to create, copy, delete, and check for the existence of a file.

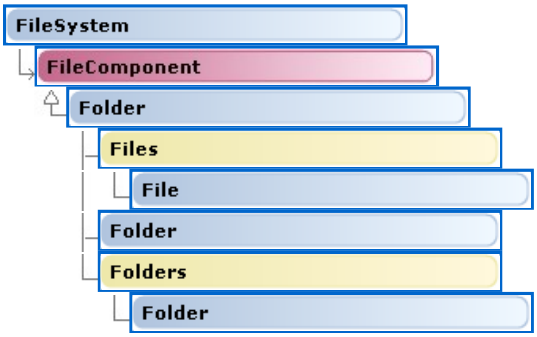
Using the File Object

Use the **Size** property to return the size of a file. The size is expressed in bytes.

```
Dim oSize As Long
Set oSize = oFile.Size
```

Folder Object

See Also Legend Use Cases Properties Methods



Represents a folder.

The **Folder** object derives from the **FileComponent** object from which it inherits the **Path** property. It aggregates:

- A **Files** collection object: it contains its files, returned thanks to the Files property.
- A **Folder** object: its parent folder accessible using the **ParentFolder** property also inherited from the **FileComponent** object.
- A **Folders** collection object: it contains its subfolders, returned thanks to the **SubFolders** property.

Retrieving the Folder Object

Use the **GetFolder** method of the **FileSystem** object, returned thanks to the **FileSystem** property of the **Application** object, to return an existing folder.

```
Dim oFolder As Folder
Set oFolder = CATIA.FileSystem.GetFolder("C:\tmp")
```

In the same way, use the other properties and methods of the **FileSystem** object to create, copy, delete, and check for the existence of a folder.

Using the Folder Object

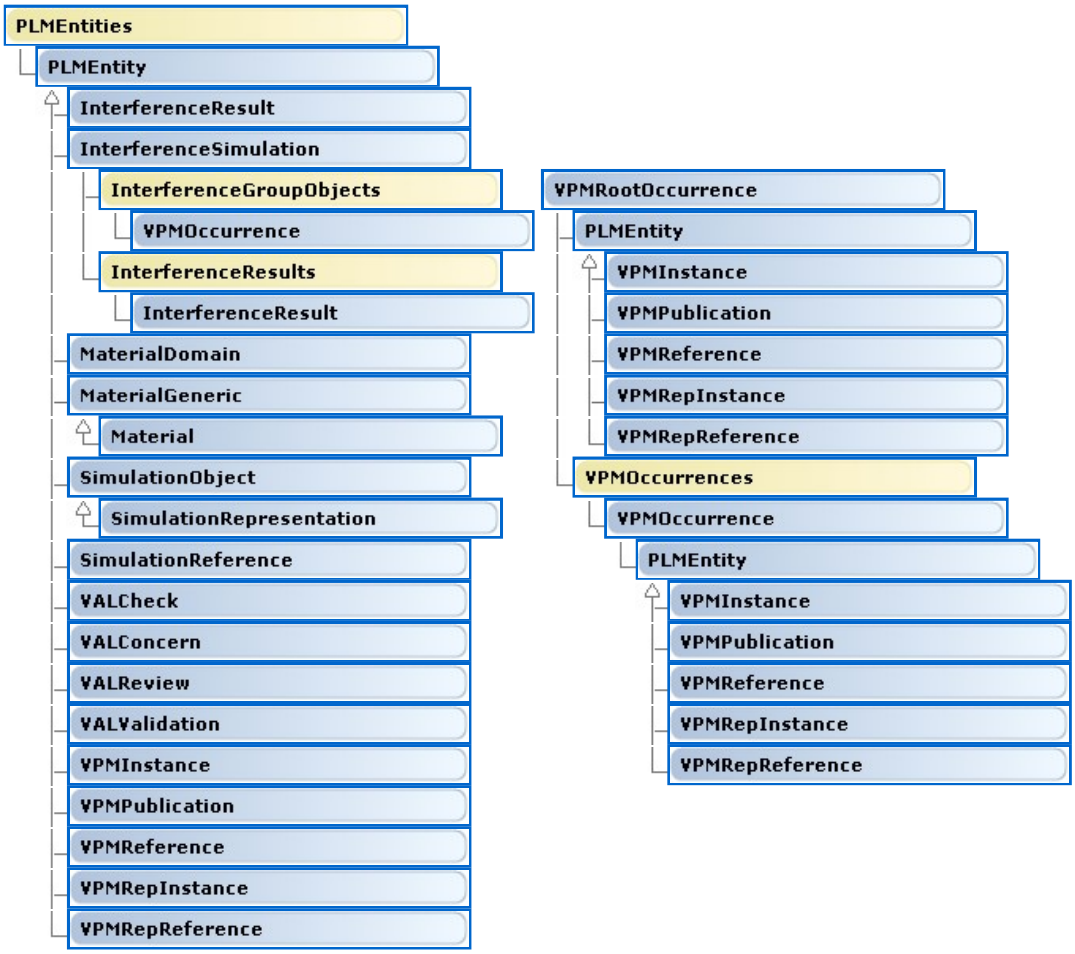
Use the **Files** property to return the **Files** collection object containing the files in the folder.

```
Dim cFiles As Files
```

```
Set cFiles = oFolder.Files
For Each oFile In cFiles
  msgbox "File path: " & oFile.Path & _
         "File type: " & oFile.Type & _
         "File size: " & oFile.Size
Next
```

PLMEntity Object

See Also Legend Use Cases Properties Methods



Represents the parent object of a PLM object either in the database or in session.

Retrieving a PLMEntity Object

A **PLMEntity** object is retrieved either:

- From the **PLMEntities** collection object that is itself retrieved from either Search Results. In this case, any kinds of **PLMEntity** objects can be retrieved, depending on what is searched. You can get the properties of these objects, but not set them as long as they are not opened in session.
- From the **PLMOccurrence** object, that is either the **VPMRootOccurrence** or a **VPMOccurrence** object that both enable you to handle the occurrence model, to which it is aggregated in the product model. In this case, the possible kinds of **PLMEntity** objects are **VPMInstance**, **VPMPublication**, **VPMReference**, **VPMRepInstance**, and **VPMRepReference** objects. These objects enable you to manipulate the instance-reference model. You can get/set the properties of these objects, since they are already opened in session. See Product Modeler Overview.

Use the **Results** property of the **DatabaseSearch** object to return a **PLMEntities** collection object as a Search Results, from which you can retrieve one or several **PLMEntity** objects.

```
Dim cPLMEntities As PLMEntities
Set cPLMEntities = oDBSearch.Results

For i = 1 To cPLMEntities.Count
  Dim oPLMEntity As PLMEntity
  Set oPLMEntity = cPLMEntities.Item(i)
  MsgBox "This PLMEntity object name is: " & oPLMEntity.GetAttributeValue("PLM_ExternalID")
```

```
Next
```

Use the **PLMEntity** property of the **VPMRootOccurrence** or a **VPMOccurrence** object, inherited from the **PLMOccurrence** object, to return the **PLMEntity** object aggregated to it in a product model.

```
Dim oVPMOccurrence As VPMOccurrence
... 'Navigate the product model to find the appropriate object
Dim oPLMEntity As PLMEntity
Set oPLMEntity = oVPMOccurrence.PLMEntity
...
```

Refer to Navigating Product Structure and to Browsing Occurrence Model.

Using a PLMEntity Object

Each **PLMEntity** derived object shares with the others the methods inherited from the **PLMEntity** object.

Use the **GetAttributeValue** and **SetAttributeValue** to get/set the value of a PLM attribute.

```
Dim oPLMEntity As PLMEntity
... 'Retrieve the PLMEntity object as described above
MsgBox "This PLMEntity object title is: " & oPLMEntity.GetAttributeValue("V_Name")
oPLMEntity.SetAttributeValue("V_Name") = "New Title"
```

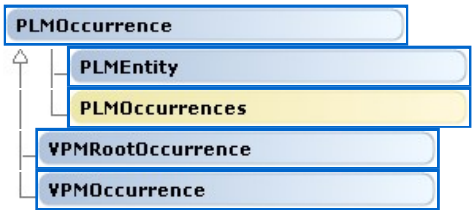
Use the **GetCustomType** to return the actual customization, type of a **PLMEntity** object.

```
Dim oPLMEntity As PLMEntity
... 'Retrieve the PLMEntity object as described above
MsgBox "This PLMEntity object customization type is: " & oPLMEntity.GetCustomType
```

Refer to each **PLMEntity** derived object to know its own properties and methods.

PLMOccurrence Object

See Also Legend Use Cases Properties Methods



Represents the parent object of the occurrence model objects.

The **PLMOccurrence** object enables you to manipulate the occurrence model. See Product Modeler Overview.

Retrieving a PLMOccurrence Object

You can retrieve a **PLMOccurrence** object through its derived objects only. See VPMRootOccurrence Object and VPMOccurrence Object.

Using a PLMOccurrence Object

Use the **PLMEntity** property to return the instance/reference model object associated with the current **PLMOccurrence** object..

```
Dim oPLMOccurrence As PLMOccurrence
... 'Retrieve either the VPMRootOccurrence or a VPMOccurrence object
Dim oPLMEntity As PLMEntity
Set oPLMEntity = oPLMOccurrence.PLMEntity
```

The **PLMOccurrences** property returns the **PLMOccurrences** collection object aggregated to the **PLMOccurrence** object. This collection object contains all the **PLMOccurrence** objects that are children of the **PLMOccurrence** object. If this collection object is empty, the **PLMOccurrence** object is a leaf node in the occurrence model tree. Prefer using the **Occurrences** properties of either the VPMRootOccurrence Object and the VPMOccurrence Object to navigate the occurrence model.

Selection Object

See Also Legend Use Cases Properties Methods

Represents the selection.

Retrieving the Selection Object

Use the **Selection** property to return the **Selection** object associated with the active **Editor** object.

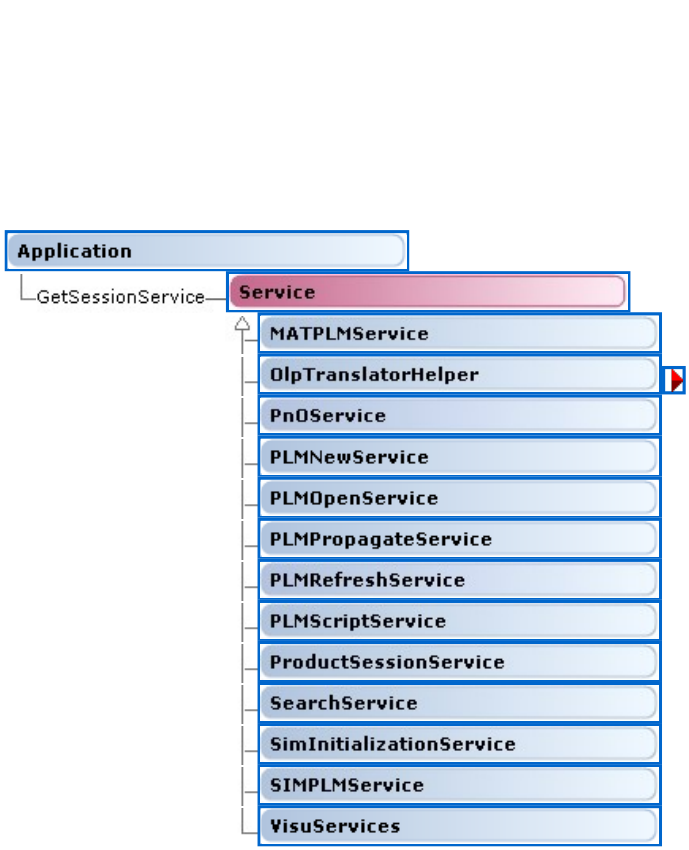
```
Dim oSelection As Selection
Set oSelection = CATIA.ActiveEditor.Selection
```

Using the Selection Object

Refer to the Selection object.

Service Object

See Also Legend Use Cases Properties Methods



Represents a service.

The **Service** object gives access to a set of object-independent operations, that either apply to the session and are editor and data independent, or globally apply to the objects controlled by the active editor and aggregated to the PLM root object. Because the **Service** object is an abstract object, you can manipulate its derived objects only.

A session-level service includes operations that either apply to any PLM type root object, or without any PLM root object active. On the opposite, an editor-level service applies only to a dedicated PLM type root object.

- Session-level services:
  - The **MATPLMService** object to create materials, set or retrieves core or covering materials, remove applied materials, and retrieves all materials in session.
  - The **OlpTranslatorHelper** object used to access OLP data objects.
  - The **PnOService** object to access a person.
  - The **PLMNewService** object to create a PLM root object.
  - The **PLMOpenService** object to open a PLM root object.
  - The **PLMPropagateService** object to save changes held by the editor.
  - The **PLMRefreshService** object to manage the refresh operation.
  - The **PLMScriptService** object to handle scripts stored in the database.
  - The **ProductSessionService** object that aggregates a collection of **Shape3D** objects.
  - The **SearchService** object to to search for PLM objects.
  - The **SimInitializationService** object to initialize the simulation.
  - The **SIMPLMService** object that manages **SimulationReference** and other Simulation objects.
  - The **VisuServices** object dealing with layers, layer filters, windows and cameras.
- Editor-level services:
  - The **DrawingGenService** object and the **DrawingService** object that apply to a Drawing representation.
  - The **FCBService** object to create or retrieve flexible boards.
  - The **FittingService** object to create or manage tracks and Tpoints.
  - The **InertiaBoxService** object to retrieve the **InertiaBox** object representing the bounding box of a given geometric object.
  - The **InertiaService** object to retrieve the **Inertia** object for a given geometric object.
  - The **InterferenceServices** object to create an interference simulation.
  - The **KnowledgeServices** object to return the **Units** collection object or the **KnowledgeCollection** collection object.
  - The **MeasurableService** object to return objects to measure curves and surfaces.
  - The **MeasureService** object to return objects to measure curves and surfaces.
  - The **PCBServices** object to create Circuit Board Design objects.
  - The **PlayerServices** object to use the play methods.
  - The **PLMProductService** object that returns root objects from the **Editor** object.
  - The **RfgService** object to manage reference planes and surfaces.
  - The **SectionService** object that manages **Section** objects from the current review.
  - The **SimComposeLinkServices** object to create a link memory object.
  - The **SimExecutionService** object to perform simulation execution.
  - The **SimPublicationServices** object to manage publications.
  - The **SimRepServices** object to determine whether the representation reference is loaded, and to load it if it is not.
  - The **SimLinkServices** object to retrieve the occurrence, the representation instance, or the target of a link.
  - The **SimSimulationService** object to return the root occurrence of the product referred by the simulation.
  - The **SrsInstantiateService** object that manages Space Reference System objects.
  - The **StrService** object that applies to Structure objects.
  - The **VALValidationService** object that applies to validation objects.

Retrieving a Service Object

Use the **GetSessionService** method of the **Application** object to return a session-level **Service** object.

```
Dim oVisuServices As VisuServices
Set oVisuServices = CATIA.GetSessionService("VisuServices")
```

The table below lists the service identifier to pass to the **GetSessionService** method to return a session-level service.

Service	Identifier
MATPLMService	MATPLMService
OlpTranslatorHelper	OlpTranslatorHelper
PnOService	PnOService
PLMNewService	PLMNewService
PLMOpenService	PLMOpenService
PLMPropagateService	PLMPropagateService

PLMRefreshService	PLMRefreshService
PLMScriptService	PLMScriptService
ProductSessionService	ProductSessionService
SearchServices	Search
SimInitializationService	SimInitializationService
SIMPLMService	SimPLMService
VisuServices	VisuServices

Use the **GetService** method of the **Editor** object to return an editor-level **Service** object.

```
Dim oDrawingService As DrawingService
Set oDrawingService = CATIA.ActiveEditor.GetService("CATDrawingService")
```

The table below lists the service identifier to pass to the **GetService** method to return an editor-level service.

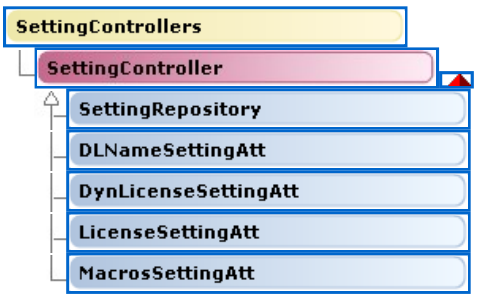
Service	Identifier
DrawingGenService	CATDrawingGenService
DrawingService	CATDrawingService
FCBService	FCBService
FittingService	FittingService
InertiaBoxService	InertiaBoxService
InertiaService	InertiaService
InterferenceServices	InterferenceServices
KnowledgeServices	KnowledgeServices
MeasurableService	MeasurableService
MeasureService	MeasureService
PCBService	PCBService
PlayerServices	PlayerServices
PLMProductService	PLMProductService
RfgService	RfgService
SectionService	SectionService
SimComposeLinkServices	SimComposeLinkServices
SimExecutionService	SimExecutionService
SimLinkServices	SimLinkServices
SimPublicationServices	SimPublicationServices
SimRepServices	SimRepServices
SimSimulationService	SimSimulationService
SrsInstantiateService	SrsInstantiateService
StrService	StrService
VALValidationService	ValidationService

Using a Service Object

Refer to each derived **Service** object to know the proposed operations and how to use them.

SettingController Object

See Also Legend Use Cases Properties Methods



Represents a setting controller.

The **SettingController** object enables you to access the setting attributes stored in setting repository files, and arranged in the different property pages of the Options dialog box you can display thanks to the **Options** command of the **Tools** menu. There are two kinds of setting controllers.

1. The **SettingRepository** object is the generic setting controller. It applies to any setting repository having an XML definition. Refer to Setting Repository Reference to have the list of such setting repositories
2. Specific setting controllers, each dedicated to a given setting repository that is not described using an XML file. For example, the **DynLicenseSettingAtt** object deals with the dynamic licensing setting repository that stores the list of available configurations and products for which you can request a dynamic license.

The generic setting controller and all of the specific ones share the five methods of the **SettingController** object to deal with the whole set, or a subset of the setting attributes of a setting repository:

- **Commit** to make a memory copy of the setting attribute values
- **Rollback** to restore the last memory copy of the setting attribute values
- **ResetToAdminValues** to restore the administered values of all the attributes
- **ResetToAdminValuesByName** to restore the administered values of a subset of the attributes
- **SaveRepository** to make a persistent copy of the setting attribute values in a file.

Retrieving the Generic Setting Controller

The generic setting controller can be retrieved for any of the setting repositories listed in Setting Repository Reference, provided the configuration or product it belongs to is installed on your computer. Use the **SettingControllers** property of the **Application** object to return the **SettingControllers** collection. Then use its **Item** method to return the generic setting controller applying to the GeneralGeneral setting repository.

```
Dim cSettingCtrls As SettingControllers
Set cSettingCtrls = CATIA.SettingControllers
Dim oSettingCtrl As SettingRepository
Set oSettingCtrl = cSettingCtrls.Item("GeneralGeneral")
```

Using the Generic Setting Controller

Use the **GetAttr** or **PutAttr** methods of the **SettingRepository** object to return or set the value of an attribute. The example below returns the value of the **DragAndDrop** attribute of the GeneralGeneral setting repository, and if this value is returned true, set it to false.

```
Dim oDragAndDrop As Boolean
oDragAndDrop = oSettingCtrl.GetAttr("DragAndDrop")
If oDragAndDrop Then
  oSettingCtrl.PutAttr "DragAndDrop", False
End If
```

Retrieving a Specific Setting Controller

Use the **SettingControllers** property of the **Application** object to return the **SettingControllers** collection. Then use its **Item** method with the appropriate identifier to return the specific **DynLicenseSettingAtt** object.

```
Dim oSettingCtrls As SettingControllers
Set oSettingCtrls = CATIA.SettingControllers
Dim oSettingCtrl As DLNameSettingAtt
Set oSettingCtrl = oSettingCtrls.Item("CATSysDynLicenseSettingCtrl")
```

The names to pass to the **Item** method are:

Setting Controller Objects	Identifiers to Pass to the Item Method
DLNameSettingAtt	CATSysDLNameSettingCtrl
DynLicenseSettingAtt	CATSysDynLicenseSettingCtrl
LicenseSettingAtt	CATSysLicenseSettingCtrl
MacrosSettingAtt	CATScriptMacrosSettingCtrl

Using a Specific Setting Controller

A specific setting controller manages the setting attributes located in a setting repository. A property and two methods enable you to manipulate each setting attribute:

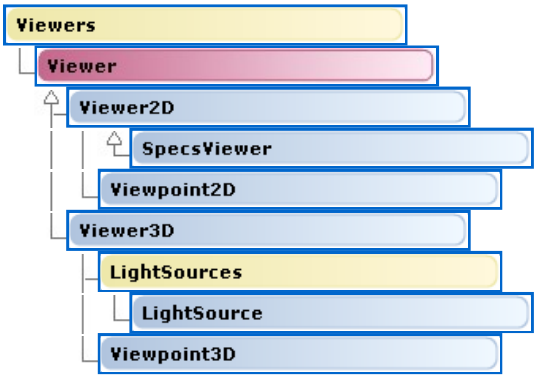
1. A read-write property to return or set the setting attribute value
2. A method to retrieve information about the setting attribute:
    - The level of administration of the setting attribute
    - The lock status of the setting attribute (Locked or Unlocked)
3. A method to set a lock onto the setting attribute

Depending on each setting attribute, when a property is not suitable, the read-write property can be replaced with a couple of methods. For example, a setting attribute managing a color must return or set three integer values. This cannot be performed by a property, and a method with three parameters must be used instead.

Refer to each derived specific **SettingController** object to know the setting attributes it manages and how to manipulate them.

Viewer Object

See Also Legend Use Cases Properties Methods



Represents a viewer.

A viewer is used to display a objects according to a given viewpoint and display options. Depending on the PLM type of the root object, the following viewers can be found in a window:

- 3D Shape Representation: a **Viewer3D** object for the part 3D objects and/or a **SpecsViewer** object for the part specification tree
- Product: a **Viewer3D** object for the assembly 3D objects and/or a **SpecsViewer** object for the assembly specification tree
- Drawing Representation: a **Viewer2D** object for the drawing sheets and/or a **SpecsViewer** object for the drawing specification tree.

When the window displays both a **Viewer3D** object and a **SpecsViewer** object, or one or several **Viewer2D** objects and a **SpecsViewer** object , it is a **SpecsAndGeomWindow** object. You can activate a given viewer in a multi-viewer window, fit all the scene in the viewer, update the display, zoom in and out, and capture the contents of the viewer as an image file.

Display options depend on the viewer type. All viewers share display options such as the background color and the display on the whole screen or in a smaller window. In addition, **Viewer3D** objects allow for different lighting modes and for modifying lighting intensity, and for depth effects, navigation styles, rendering modes, and clipping modes.

Viewpoint3D Object

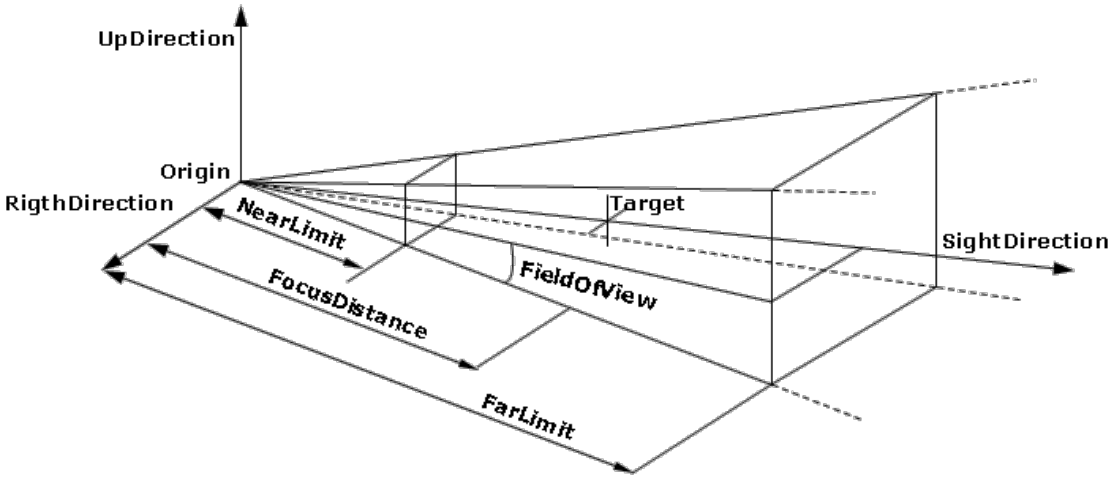
See Also Legend Use Cases Properties Methods

Represents a 3D viewpoint.

A **Viewpoint3D** object is aggregated to the **Viewer3D** object. It holds data to define how objects are seen to enable their display by a 3D viewer:

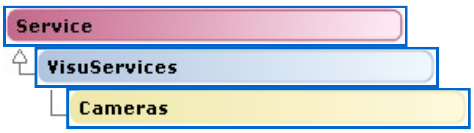
- The eye location, also named the origin of the scene to display, expressed in model units
- The distance from the eye location to the target, that is, to the looked at point in the scene.
- The sight, up, and right directions, defining a 3D coordinate system with the eye location as origin.
- The projection type: perspective (conic) or parallel (cylindrical).
- The zoom factor to apply for display.

The 3D viewpoint is the object that stores data which defines how your objects are seen to enable their display by a 3D viewer. This data includes namely the eye location, also named the origin, the distance from the eye to the target, that is to the looked at point in the scene, the sight, up, and right directions, defining a 3D axis system with the eye location as origin, the projection type chosen among perspective (conic) and parallel (cylindrical), and the zoom factor. The right direction is not exposed in a property, and is automatically computed from the sight and up directions.



VisuServices Object

See Also Legend Use Cases Properties Methods



Represents a service for visualization purposes.

The **VisuServices** object lets you manage:

- The **Cameras** collection object.
- The layers and the layer filters.
- The hidden elements visibility.
- A new window for the data displayed in the active one.

Retrieving the VisuServices Object

Refer to the Service Object.

Using the VisuServices Object

Use the **Cameras** property to return the **Cameras** collection object.

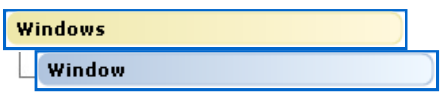
```
Dim cCameras As Cameras
Set cCameras = oVisuServices.Cameras
```

Use the **NewWindow** method to create a new **Window** object.

```
Dim oWindow As Window
Set oWindow = oVisuServices.NewWindow()
```

Windows Collection Object

See Also Legend Use Cases Properties Methods



A collection of **Window** objects that represent of all the windows currently managed by the application.

The **Windows** collection object gathers **Window** objects which make the link with the windowing system and display objects in a viewable form, mainly in 3D or 2D modes, or as a specification tree in graph or tree mode. Windows of the collection can be arranged in the frame.

Retrieving the Windows Collection

Use the **Windows** property of the **Application** object to return the **Windows** collection object.

```
Dim cWindows As Windows
Set cWindows = CATIA.Windows
```

Using the Windows Collection

Use the **Item** method to return a window from the **Windows** collection object, namely below the third one.

```
Dim oWindow As Windows
Set oWindow = CATIA.Windows.Item(3)
```

Note that you can also use the window name in place of the window range. The example below returns the search window the name of which is "Name Like Part".

```
Dim oWindow As Windows
Set oWindow = CATIA.Windows.Item("Name Like Part")
```

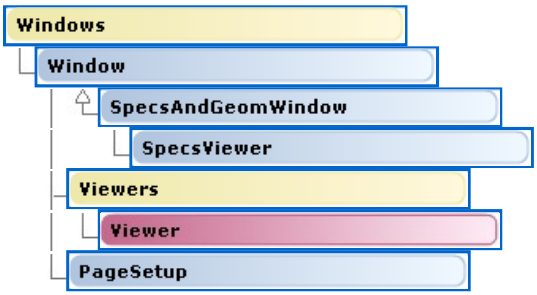
Use the **Arrange** method to arrange the windows. You can arrange the windows, according to the values of the CatArrangeStyle enumeration:

- As horizontal tiles.
- As vertical tiles.
- As a cascade.

```
CATIA.Windows.Arrange(catArrangeCascade)
```

Window Object

See Also Legend Use Cases Properties Methods



Represents a window.

A **Window** object aggregates a **Viewers** collection object that enables the window to display the application data in the appropriate modes using viewers. A **SpecsAndGeomWindow** object features altogether a 2D or a 3D viewer and a specification tree viewer. A window can be activated, that is becomes the active one, using the **Activate** method. This implies that the editor that controls the objects displayed in this window is also activated if it was not, and that subsequent interactions will affect it until another window is activated instead. The **Viewers** property returns the aggregated **Viewers** collection object, and the **ActiveViewer** property returns the active viewer in the window.