第 8 讲工程制图

目录

1 工程制图综述 3

1.1 制图的前提条件 4

1.2 制图的结构对象 4

1.3 制图的 2D 组件对象 5

1.4 制图的 2D 几何对象 6

1.5 制图的 2D 约束对象 6

1.6 制图的标注对象 6

1.7 制图修饰对象 7

1.8 制图生成视图 8

1.9 制图生成对象 8

2 在 CATDrawing 文件中创建图纸和视图 9

2.1 创建并初始化制图文件 9

2.2 获取文件中的 Drawing 特征和 drawing container 9

2.3 在文件中创建制图标准 11

2.4 给文件中的当前图纸添加格式 12

2.5 创建额外的图纸并将其放置在 drawing 特征下 13

2.6 创建创建一个视图并将其放置在图纸中 13

2.7 在视图中创建几何元素 13

2.8 保存文件并退出 14

3 创建图框和标题栏 14

3.1 创建、初始化并开始 Drawing 文件绘制 14

3.2 通过 Drawing 和图纸获取背景视图 15

3.3 通过几何工厂来创建图框 15

3.4 获取标注工厂并创建修饰 17

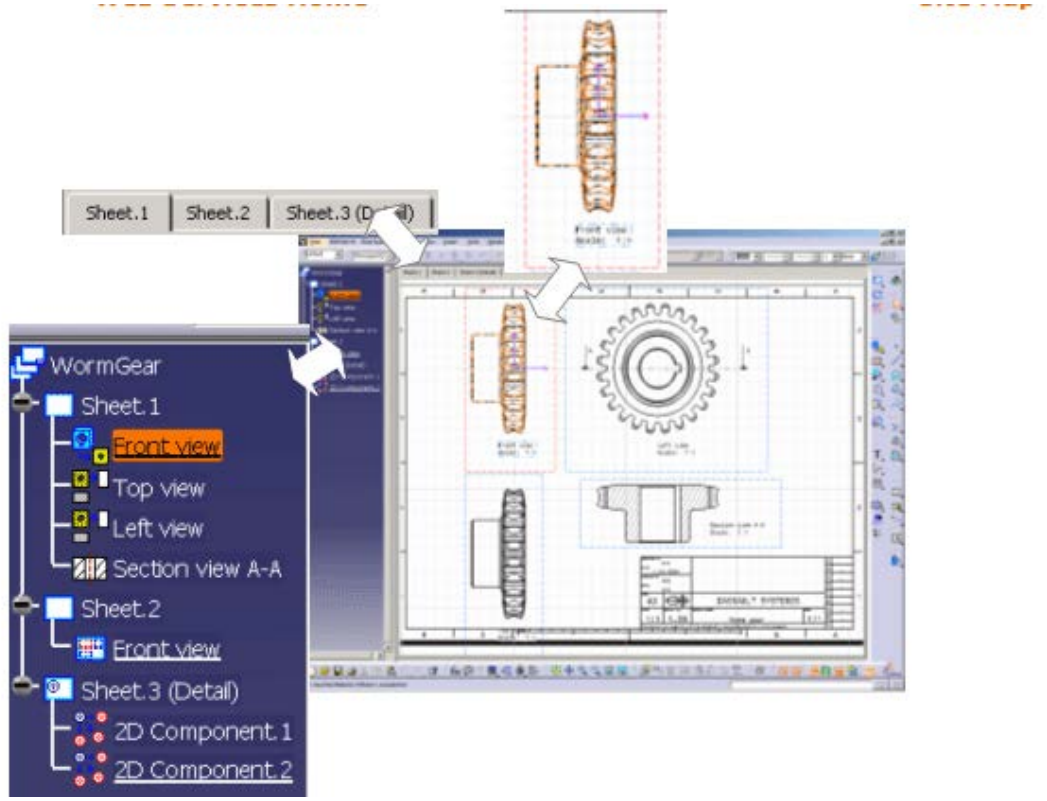
4 从 3D 模型中创建剖视图 18

4.1 创建并初始化文件 18

4.2 用 SketchForSection 草图创建剖视图 20
5 制图程序应用 23

1 工程制图综述

工程制图主要描述说明制图前提要求、制图结构、2D 几何、2D 约束、标注对象、修饰对象、制图生成对象等。
工程制图主要与 2D 设计、创成式绘图、打印、外部 CAD 图纸转化、保存图纸格式模块相关，制图能自动与相关模块进行自动连接。创成式制图模块能够从 CATIProduct、CATPart 或者 CATIA v4 模型中自动创建工程图，外部 CAD 图纸转化能够根据标准格式自动批创转化外部图纸，也可以将标准格式批转换为其他格式。2D 设计模块能够在制图环境中添加 workbench 或者 workshop。
制图模块管理几何、约束、标注、图形修饰、结构等对象（图纸、视图、详细图纸）。所有的元素都由完全集成在 CATDrawing 文件中的制图模块创建。这就是说，这些对象可以由内部的编辑器重新构造，这些编辑包括草图，尺寸，模型方向，模型定位，约束等。



制图工作台加载并且当前的制图图纸打开(Sheet.1)制图结构树在图纸的左边显示，结构树显示了结构对象之间的集合关系。“Front View”是“Sheet.1”的前视图，它是激活图纸的激活视图（视图下有下划线），这是说通过鼠标点选创建的元素都会集成在这个视图中。在文件的顶部有选项卡可以控制激活的图纸。
一个 CATDrawing 文件是由多个图纸组成，同时还包括制图标准信息（ISO，ANSI，ASME 和 JIS）。
图纸与纸张的空间紧密联系，空间由以下元素定位：1.一个主视图，这个主视图支持由几何直接创建到图纸上；2.一个背景框视图，这个视图是图框、标题栏、版本栏、和物料清单的视图。3.任意数量的交互创建视图或者其他视图（剖视图，轴测图等）。
视图会包括 5 种元素：

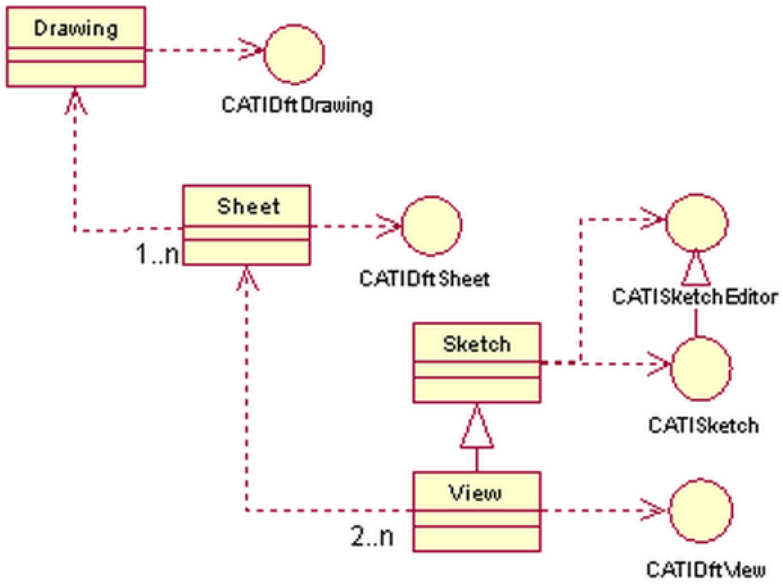
1.2D 几何，通过制图交互命令创建。包括用草图编辑创建的线，点等。
2.创成式创建结果，通过制图中的交互命令可以从三维模型中创建 2D 几何元素。
3.约束，用草图模块创建的约束。
4.标注，所有包含了文本信息的元素，尺寸，多文本标注，粗糙度符号，焊接符号等
5.修饰，包括添加的轴线，中心线，箭头，螺纹等

1.1 制图的前提条件

制图模块主要由两个产品管理：一是交互式制图，它能在需要的时候添加 2D 设计元素；二是创成式制图，它能够提供从 3D 模型和装配中定义工程图的能力。
这两个产品是给制图模块开发应用的前提条件。
制图模块包括两部分 API：一是制图界面，这个框架包含了除了 2D 几何，描述和约束对象之外的所有界面；二是草图编辑界面，这个框架这个框架致力于管理草图编辑组件，描述，2D 几何和约束对象。

1.2 制图的结构对象

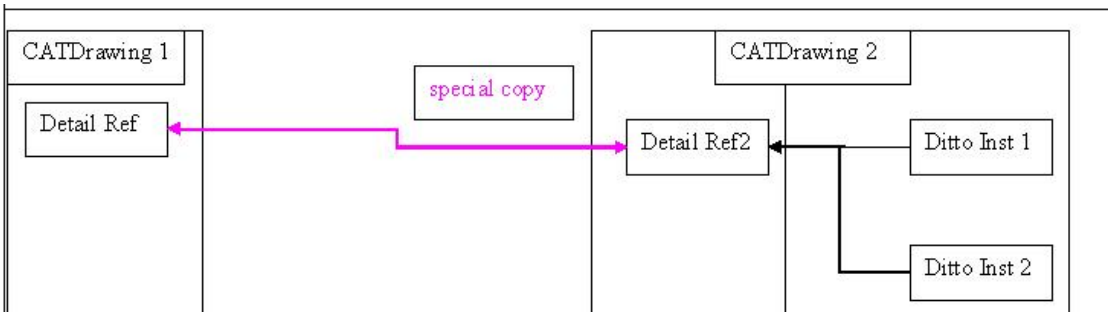
制图应用数据在应用性 Container 中，制图的 container 可以从通过方法 `GetFeatContainer` 从制图的 root 中获得。
当一个绘图文件 CATDrawing 创建的时候，Drawing container、Drawing root 、一个图纸、集成在第一个图纸中的主视图以及集成在第一个图纸中的背景视图 5 个对象同时被创建。
要完成图纸创建以下两步必不可少：引入一个绘图标准和从标准中获取一个可用的格式并应用到第一个图纸中。
当创建视图，图纸，或者其他任何对象的时候都要用到从 Drawing container 中实现的接口 *CATIDrwFactory* （制图工厂）。
注意：为了创建图纸，最好使用 *CATIDftDrawing 接口中的* AddSheet 方法，它考虑了图纸的数量管理。
图纸集成了多个视图，一个视图中集成了多个实例化的对象。在图纸中始终会有一个视图处于激活状态。为了在图纸中直接创建对象，图纸的主视图会被激活。



视图继承了草图中的管理 2D 几何和 2D 约束的能力，共有两种几何共存在一个视图中，一是 2D 几何，它通过 CAA 草图模块管理；二是创成式几何，它从 3D 模型中生成。
有两种不同的接口可以访问几何信息：
IDMXXX 接口，这些接口通过 2D 几何和创成式几何共同实现；
CATI2DXXX 接口，这些接口只能是 2D 几何可以访问。

1.3 制图的 2D 组件对象

一个 2D 组件是一个几何和标注的重用集。组件被定位在图纸中并且可以像编辑视图一样编辑它，这就是为什么在组建又被称为细节视图。2D 组件能够实例化多次，每个实例能够提供一个带有具体方向，位置和尺度的组件。细节视图可以和与其相关的组件实例在同一个 CATDrawing 文件中，也可以在分开的 CATDrawing 文件中。
Catalog 浏览器命令容许从一个外部的 CATDrawing 文件 CATDrawing1 中为 CATDrawing2 文件实例化一个视图，为了实现这样做，创建了 Detail Ref 的特殊拷贝(Detail Ref2)，和这个拷贝的实例化 (Ditto Inst 1)。如图所示：



Detail Ref2 不能直接修改，可以通过修改 Detail Ref 来实现修改拷贝体。
实例中文本是可以修改的，当一个带有具体属性的模板文本在实例化的时候就可以文件可修改文本。以下的代码就可以对该文本进行修改。

```
// GetModifiableObjects 方法能够获取模板中的可修改文字
CATIADrawingComponent * piMyDrawComp = NULL;
HRESULTrc = piMyDitto -> QueryInterface
(IID_CATIADrawingComponent,(void**)&piMyDrawComp);
if ( SUCCEEDED(rc) )
{
   long Count = 0;
   piMyDrawComp -> GetModifiableObjectsCount (Count);
   for ( int Idx = 1; Idx<=Count; Idx++ )
   {
      CATVariant Variant;
      rc = BuildVariant ((long)Idx, Variant);
      if ( SUCCEEDED(rc) )
      {
         CATIABase * piABase = NULL;
         rc = piMyDrawComp -> GetModifiableObject (Variant, piABase);
         if ( SUCCEEDED(rc) )
```

{

1.4 制图的 2D 几何对象

2D 几何对象由 2D 线框工厂中的 SketcherInterface 定义。可以通过对视图进行 QueryInterface 来获取 CATI2DWFFactory 接口的指针。
注意：
1.处理 2D 几何时必须激活视图。当激活视图，与视图相关联的草图会打开编辑，就可以创建几何。
2.如果用交互命令处理几何，在 2D 几何创建步奏的最后，*CATISketchEditor* 接口中的 SaveEdition 方法会被自动执行。*CATISketch* 派生自 *CATISketchEditor，所以*接口 *CATISketch* 也可以执行 SaveEdtion 方法。

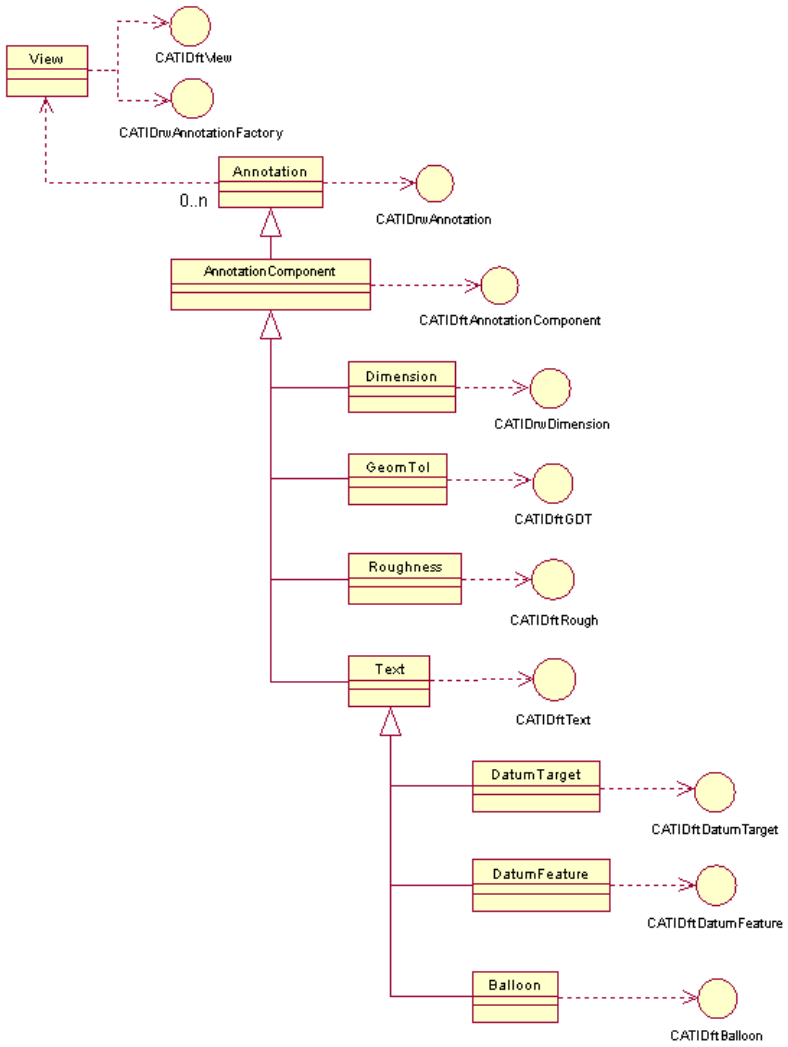
1.5 制图的 2D 约束对象

这些对象用在 SketcherInterface 中定义的 2D 约束工厂创建。CATI2DConstraintFactory 接口的指针可以从视图 QueryInterface 获得。这个工厂中 CreateConstraint 方法允许在视图中创建约束。

```
  --
Cst2DType_ERROR,
Cst2DType_Distance,
Cst2DType_Angle,
Cst2DType_Concentric,
Cst2DType_Radius,
Cst2DType_Parallel,
Cst2DType_Perpend,
Cst2DType_On,
Cst2DType_Tangent,
Cst2DType_Symmetry,
Cst2DType_Length,
Cst2DType_Reference,
Cst2DType_Horizontal,
Cst2DType_Vertical,
Cst2DType_MajorRadius,
Cst2DType_MinorRadius,
Cst2DType_Begin,
Cst2DType_End,
Cst2DType_MiddlePoint,
Cst2DType_Depend,
Cst2DType_EquidistantPoint,
Cst2DType_CylinderRadius,
Cst2DType_SetDepend
```

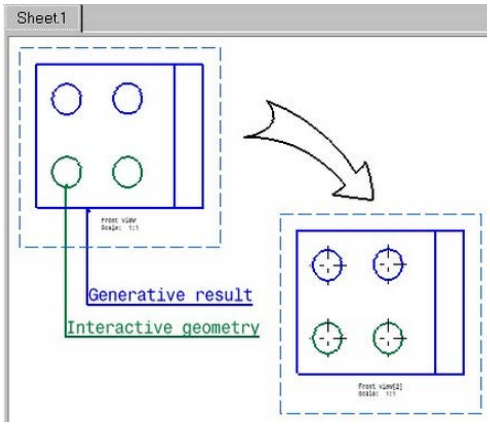
1.6 制图的标注对象

标注对象通过制图标注工厂来完成创建。CATIDrwAnnotationFactory 接口的指针可以通过在视图中使用 QueryInterface 来获取。所有的标注继承自标注组件对象，而标注组件对象又继承自标注对象。CATIDrwAnnotation 接口管理标注之间或者标注与几何之间的位置、方向相关的内容。CATIDrwAnnotationComponent 接口专门用来管理与标注文本相关的全局信息。
当一个标注被修改，标注将会重建并获取结果。



1.7 制图修饰对象

这些对象使用制图标注工厂可以创建，CATIDrwAnnotationFactory 接口的指针可以通过在视图中使用 QueryInterface 来获取。有时这些元素要求几何先被创建。



CATIDrwAnnotationFactory 接口中的方法如图所示：

- **CreateDatumFeature**(double,double,double,double,CATUnicodeString&)
  Creates a DatumFeature.
- **CreateDatumTarget**(double,double,double,double,CATUnicodeString&,CATUnicodeString&,int)
  Creates a DatumTarget.
- **CreateDftArrow**(double[2],double[2],CATIDftArrow**,int,int,IUnknown*,IUnknown*)
  Creates a drawing arrow.
- **CreateDftBalloon**(double[2],double[2],CATUnicodeString&,CATIDftBalloon**)
  Creates a drawing balloon.
- **CreateDftGDT**(int,double[2],double[2],CATIDftLeader**,CATIDftGDT**)
  Creates an empty GDT.
- **CreateDftRoughness**(double[2],CATIDftRough**)
  Creates a roughness symbol.
- **CreateDftText**(double[2],CATIDftText**)
  Creates a text.
- **CreateDimSystem**(CATIDrwDimDimension*,CATDimSystemDefinition*,CATIDrwDimSystem**)
  Creates a dimension system.
- **CreateDimension**(CATIUnknownList*,double**,CATDrwDimType,CATDimDefinition*,CATIDrwDimD
  Creates a dimension from selected elements.
- **CreateDrwAreaFill**(CATLISTV(CATISpecObject_var)&,CATISpecObject_var,CATUnicodeString)
  Creates an area fill.
- **CreateDrwAxisLine**(CATBaseUnknown_var,CATBaseUnknown_var)
  Creates an Axis Line.
- **CreateDrwCenterLine**(CATBaseUnknown_var,CATBaseUnknown_var)
  Creates a Center Line.
- **CreateDrwCoordDimension**(double*,CATBaseUnknown*,double*,CATIDrwCoordDimension_var&,
  Creates a coordinates dimension.
- **CreateDrwThread**(CATBaseUnknown_var,CATBaseUnknown_var,int,CATDftThreadTypeEnum)
  Creates a Thread.
- **CreatePicture**(CATPixelImage*,double,double,CATIDrwPicture**,CATBoolean)
  Creates a drawing picture.
- **CreatePicture**(CATVectorImage*,double,double,CATIDrwPicture**)
  Creates a drawing picture.

1.8 制图的生成视图

这些视图是通过来自 Part 或者 Product 文件的 3D 数据创建的，为了创建一个正视图，一个带有视图框的视图会被创建，然后这个视图的类型可以通过 *CATIView* 接口的 SetViewType 方法进行设置。视图的类型包括正视图，前视图，后视图，剖视图，轴测图等。SetDoc 方法可以创建生成视图和 3D 文件之间的连接关系。最后，使用 *CATIGenerSpec* 接口的 SetProjectionPlane 方法定义一个投影平面，通过使用 CATIView 接口的 GetGenerSpec 方法可以获取 *CATIGenerSpec* 接口指针，对于任何视图，审查视图必须使用 CATIDftSheet 接口的 AddView 方法集成到图纸通过。为了在视图中获取 3D 投影的结果，调用 CATIView 接口的 Update 方法即可。

特别的生成视图可以通过 *CATIDftGenViewFactory* 接口轻松创建，这个接口通过图纸对象实现：

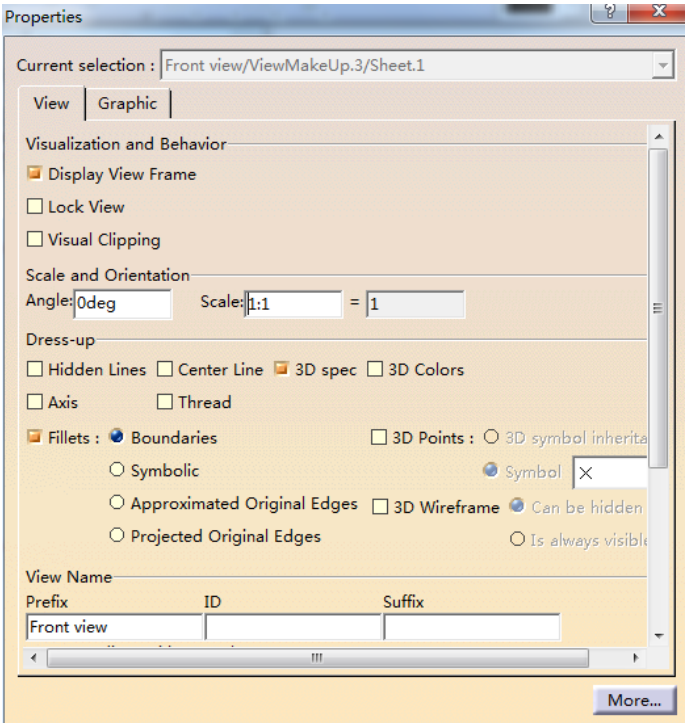
1，CreateViewFrom3D ,用来生成一个制图，通过 TPS 视图。（TPS 视图是一个通过 Functonal tolerancing and annotation 工作台创建的 3D 视图）

2，CreateSectionView，从一个定义了剖面轮廓的生成视图中创建一个剖视图。

3，CreateStandAloneSectionView ，创建一个单独的剖视图，这个剖视图是直接同剖切的 3D 元素相关的，3D 剖切元素可以为一个平面，一个草图平面，一个视图的平面表面。剖切的轮廓是和 3D 元素息息相关。

1.9 制图生成对象

这些对象是创建或者更新视图时自动产生的，生成几何是从 3D 文件中关联并定义的。生成几何是不能编辑的，只有图形属性可以修改。



2 在 CATDrawing 文件中创建图纸和视图

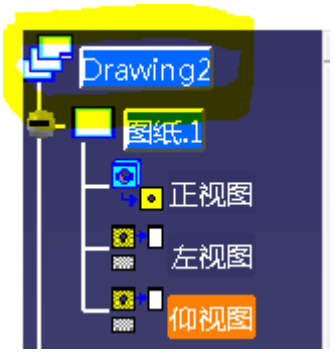
2.1 创建并初始化制图文件

```
// creates a session
   CATSession *pSampleSession = NULL;
   HRESULT hr = ::Create_Session("SampleSession",pSampleSession);
   if (FAILED(hr)) return 1;
   CATDocument* pDoc = NULL;
   hr = CATDocumentServices::New("CATDrawing", pDoc);
   if (FAILED(hr)) return 2;
```

这段代码是按照常规顺序创建一个 CATIA 文件的。

2.2 获取文件中的 Drawing 特征和 drawing container

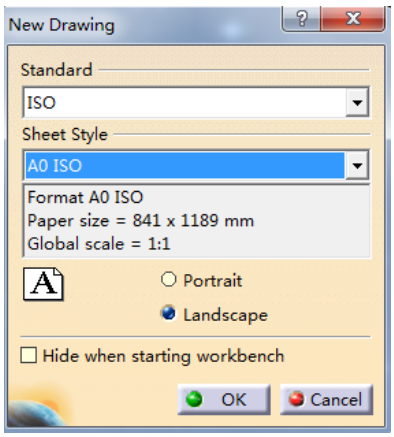
在之前的 part 文件中讲到一个 part 文件的最高特征是 part 特征，这儿也是这个道理，在制图文件中的最高特征就是 Drawing 特征。获取代码如下所示：




```
// 获取 drawing 特征.
CATIDftDrawing *piDftDrawing = NULL;
CATIDftDocumentServices *piDftDocServices = NULL;
CATIContainer_var spDrwCont;
if    (SUCCEEDED(pDoc->QueryInterface(IID_CATIDftDocumentServices,
(void **)&piDftDocServices)))
{
  if    (SUCCEEDED(piDftDocServices->GetDrawing(IID_CATIDftDrawing,
(void **)&piDftDrawing)))
  {
    if (piDftDrawing)
    {
      // 获取 drawing container.
      CATISpecObject *piSpecObj=NULL;
      if
(SUCCEEDED(piDftDrawing->QueryInterface(IID_CATISpecObject,(void
**)&piSpecObj)))
      {
        spDrwCont = piSpecObj->GetFeatContainer();
        piSpecObj->Release();
        piSpecObj=NULL;
      }
    }
  }
  piDftDocServices->Release();
  piDftDocServices=NULL;
}
```

制图文件的根特征就是 Drawing 特征，它可以用来实现 CATIDrawing 接口。我们可以使用在文件中实现的CATIDftDocumentServices 接口来获取CATIDrawing接口指针，GetDrawing 的方法的第一个参数就是要获取的 CATIDrawing 接口指针。

2.3 在文件中创建制图标准




```
// 从 drawing container 中获取标准
CATIDftStandardManager *piStdmgr = NULL;
if (SUCCEEDED(spDrwCont->QueryInterface(IID_CATIDftStandardManager,(void**)&piStdmgr)))
{
   // 从标准列表中找到一个标准
   CATIStringList *piListstd = NULL;
   if ( SUCCEEDED(piStdmgr->GetAvailableStandards(&piListstd)) && piListstd )
   {
      unsigned int nbrstd = 0;
      piListstd->Count(&nbrstd);
      for (unsigned int indice = 0; indice < nbrstd; indice ++)
      {
         wchar_t *wstd = NULL;
         if ( SUCCEEDED ( piListstd->Item ( indice, &wstd ) ) && wstd )
         {
      const CATUnicodeString ANSI_UncS = "ANSI";
      CATUnicodeString stdname;
      stdname.BuildFromWChar(wstd);
      if ( stdname == ANSI_UncS )
      {
         //给文件中导入 ANSI 标准
               piStdmgr->ImportStandard (wstd);
               break;
            }
            delete[] wstd; wstd = NULL;
         }
      }
      piListstd->Release(); piListstd=NULL;
   }
   piStdmgr->Release (); piStdmgr=NULL;
}
```

CATIDftStandardManager 接口是由是制图应用 container 实现，GetAvailableStandards 方法返回可以用的标准列表，，ImportStandard 方法能够将一个标准应用到制图文件中。

2.4 给文件中的当前图纸添加格式

```
// 从图纸中获取可用的格式
CATIDftDrawingFormats *piDftFormats = NULL;
CATUnicodeString myFormatName;
if (SUCCEEDED(piDftDrawing->QueryInterface(IID_CATIDftDrawingFormats,(void
**)&piDftFormats)))
{
   CATLISTV(CATISpecObject_var) spListFormat;
   if (SUCCEEDED(piDftFormats->GetAvailableFormats(spListFormat)))
   {
      int nbformats= spListFormat.Size();
      // 获取列表中第一个格式
      if (nbformats >= 1)
      {
         CATIDftFormat_var spFormat = spListFormat[1];
            spFormat->GetFormatName(myFormatName)
      }
   }
}
// 给当前工作图纸设置格式
CATIUnknownList *piListOfSheet=NULL;
CATIDftSheetFormat *piDftSheetFormat=NULL;
if (SUCCEEDED(piDftDrawing->GetSheets(&piListOfSheet)))
{
   IUnknown * item = NULL;
   unsigned int nbSheet = 0;
piListOfSheet->Count(&nbSheet);
   // Loop on all Generated Geometry of the view.
   for(unsigned int i=0 ; i<nbSheet ; i++)
   {
      if( SUCCEEDED( piListOfSheet->Item(i, &item) ) )
      {
         if (item)
         {
            if (SUCCEEDED(item->QueryInterface (IID_CATIDftSheetFormat,(void
**)&piDftSheetFormat)))
            {
                  piDftSheetFormat->SetSheetFormat(myFormatName)
}
         }
```

```
    }
  }
}
```

CATIDftDrawingFormat 接口是从 Drawing 特征实现的。GetAvailableFormats 方法返回可以用的所有格式， SetSheetFormat 方法给图纸的格式进行初始化.

2.5 创建额外的图纸并将其放置在 drawing 特征下

```
CATIDftSheet *piDftNewSheet = NULL;
wchar_t *pSheetName= L"MyNewSheet";
if (SUCCEEDED(piDftDrawing->AddSheet(&piDftNewSheet,pSheetName)))
```

CATIDftDrawing 接口的 AddSheet 方法能够创建图纸并将其添加到 drawing 特征下，如果这个方法是在交互环境中被调用，那么图纸选项页也会同时更新。

2.6 创建一个视图并将其放置在图纸中

```
// 从 drawing container 可以实现 CATIDrwFactory，
CATIDrwFactory_var spDrwFact = spDrwCont;
// 创建一个带有框架的视图
CATIDftViewMakeUp *piNewViewMU = NULL;
if (NULL_var != spDrwFact && SUCCEEDED(spDrwFact ->
CreateViewWithMakeUp(IID_CATIDftViewMakeUp, (void **)&piNewViewMU)))
{
  if (piNewViewMU)
  {
    //从框架中获取视图
    CATIView *piNewView = NULL;
    if (SUCCEEDED(piNewViewMU->GetView(&piNewView)))
    {
      if (piNewView)
      {
        // 设置视图类型
        piNewView->SetViewType(FrontView);
        piNewViewMU->SetAxisData(100.0,50.0);
       // 添加视图到图纸
       if (piDftNewSheet) piDftNewSheet->AddView(piNewViewMU);
```

CATIDrwFactory 是由 DrawingContainer 实现的，视图框架对象和视图紧密相关，所有通过 CraeteViewWithMakeUp 方法创建了两个对象，最后，还应该设置视图类型，并将其添加在图纸中。

2.7 在视图中创建几何元素

```
// 激活要创建几何元素的视图
CATIDftView *piDftNewView=NULL;
```

```
if (SUCCEEDED(piNewView->QueryInterface(IID_CATIDftView,(void **)&piDftNewView)))
{
   piDftNewSheet->SetDefaultActiveView(piDftNewView);
   piDftNewView->Release();
   piDftNewView=NULL;
}
// 获取线框工厂以创建几何
CATI2DWFFactory_var spGeomFactory(piNewView);
// 创建一个圆
double center[2];
center[0]=50.0;
center[1]=60.0;
double radius = 50.0;
CATISpecObject_var spCercle;
if (NULL_var != spGeomFactory)
{
   spCercle = spGeomFactory->CreateCircle(center,radius);
```

只能在当前视图中创建几何元素，线框工厂也是从当前视图中获取。piNewView 就是刚刚建立的视图，刚建立的默认为当前视图。

2.8 保存文件并退出

```
// 保存文件
   hr = CATDocumentServices::SaveAs(*pDoc, (char *)fileName);
   if (FAILED(hr)) return 6;
   // 关闭文档并退出
   CATDocumentServices::Remove (*pDoc);
   ::Delete_Session("SampleSession");
```

3 创建图框和标题栏

3.1 创建、初始化并开始 Drawing 文件绘制

```
CATDocument* pDoc = NULL;
   if (!SUCCEEDED(CATDocumentServices::OpenDocument(fileName, pDoc)))
   {
      return 2;
   }
   //用 CATIDftDocumentServices 接口获取 drawing 特征 using the
   CATIDrawing *piDrawing = NULL;
```

```
CATIDftDocumentServices *piDftDocServices = NULL;
if (SUCCEEDED(pDoc->QueryInterface(IID_CATIDftDocumentServices, (void **)&piDftDocServices)))
{
  piDftDocServices->GetDrawing(IID_CATIDrawing, (void **)&piDrawing);
}
if (NULL == piDrawing)
  return 1;
```

drawing 文件的根特征是 drawing 特征，这个特征采用 CATIDrawing 接口来实现，使用从文件中获取的 *CATIDftDocumentServices 接口来*获取 CATIDrawing 接口指针，GetDrawing 方法的第一个参数就是 CATIDrawing 接口的 IID。此处 pDoc 指针是 CATDrawing 文件的指针。

3.2 通过 Drawing 和图纸获取背景视图

```
// 获取当前图纸
CATISheet_var spSheet = piDrawing->GetCurrentSheet();
// 获取当前图纸的背景视图
CATIView_var spBgView = spSheet->GetBackgroundView();
```

一个制图文件可以包含多个图纸，但只有一个图纸是当前图纸，当前图纸是包含激活视图的图纸，激活视图即是当前可以编辑的视图。CATISheet 和 CATIView 接口的方法可以返回当前视图和当前图纸。

3.3 通过几何工厂来创建图框

```
// 通过以下 2 个步骤之后可以在视图中创建几何:
//   - 设置视图为当前视图
//   - 获取视图的几何工厂
spSheet->SetCurrentView(spBgView);
CATI2DWFFactory_var spGeomFactory = spBgView;
double X[8] = { 936.0, 949.0, 963.0, 1010.0, 1062.0, 1093.0, 1157.0, 1176.0};
double Z[11] ={ 17.0, 22.0, 35.0, 48.0, 53.0, 58.0, 63.0, 68.0, 73.0, 78.0, 88.0};
double PtTmp[6];
double startPoint[2], endPoint[2];
// 创建水平线
PtTmp[0]=Z[0];
PtTmp[1]=Z[1];
PtTmp[2]=Z[3];
PtTmp[3]=Z[5];
PtTmp[4]=Z[7];
PtTmp[5]=Z[10];
for (int i= 0;i < 6; i++)
{
  startPoint[0] = X[0];
  startPoint[1] = PtTmp[i];
```

```
    endPoint[0] = X[7];
    endPoint[1] = PtTmp[i];
    spGeomFactory->CreateLine(startPoint, endPoint);
}
startPoint[0] = X[3];
startPoint[1] = Z[2];
endPoint[0] = X[7];
endPoint[1] = Z[2];
spGeomFactory->CreateLine(startPoint, endPoint);

startPoint[0] = X[0];
startPoint[1] = Z[9];
endPoint[0] = X[5];
endPoint[1] = Z[9];
spGeomFactory->CreateLine(startPoint, endPoint);

PtTmp[0]=Z[4];
PtTmp[1]=Z[6];
PtTmp[2]=Z[8];
for (i= 0;i < 3; i++)
{
    startPoint[0] = X[3];
    startPoint[1] = PtTmp[i];
    endPoint[0] = X[5];
    endPoint[1] = PtTmp[i];
    spGeomFactory->CreateLine(startPoint, endPoint);
}

//创建垂直线
PtTmp[0]=X[0];
PtTmp[1]=X[7];
for (i= 0;i < 2; i++)
{
    startPoint[0] = PtTmp[i];
    startPoint[1] = Z[0];
    endPoint[0] = PtTmp[i];
    endPoint[1] = Z[10];
    spGeomFactory->CreateLine(startPoint, endPoint);
}
startPoint[0] = X[3];
startPoint[1] = Z[1];
endPoint[0] = X[3];
endPoint[1] = Z[10];
spGeomFactory->CreateLine(startPoint, endPoint);
```

```
  startPoint[0] = X[5];
  startPoint[1] = Z[3];
  endPoint[0] = X[5];
  endPoint[1] = Z[10];
  spGeomFactory->CreateLine(startPoint, endPoint);
  PtTmp[0]=X[4];
  PtTmp[1]=X[6];
  for (i= 0;i < 2; i++)
  {
    startPoint[0] = PtTmp[i];
    startPoint[1] = Z[1];
    endPoint[0] = PtTmp[i];
    endPoint[1] = Z[2];
    spGeomFactory->CreateLine(startPoint, endPoint);
  }
  startPoint[0] = X[1];
  startPoint[1] = Z[5];
  endPoint[0] = X[1];
  endPoint[1] = Z[7];
  spGeomFactory->CreateLine(startPoint, endPoint);
  startPoint[0] = X[2];
  startPoint[1] = Z[5];
  endPoint[0] = X[2];
  endPoint[1] = Z[9];
  spGeomFactory->CreateLine(startPoint, endPoint);
...
```

在使用几何工厂之前，必须确定当前视图，视图用来实现 2D 几何工厂接口 *CATI2DWFFactory*，这个工厂同时也是草图工厂，所以可以使用与草图工厂中的相同方法在视图中绘制几何对象。

3.4 获取标注工厂并创建修饰

```
// 获取视图的标注工厂
  CATIDrwAnnotationFactory_var spAnnFactory = spBgView;
  // 创建文本编辑
  CATIDftText *piDftText = NULL;
  const double txtpos1[2] = {1013.,45.};
  if (SUCCEEDED(spAnnFactory->CreateDftText(txtpos1,&piDftText)))
  {
    // 设置字符串
    CATUnicodeString textString("TITLE BLOCK PERFORMED BY CAA2 APPLICATION");
    wchar_t *ptxtChar = new wchar_t[textString.GetLengthInChar()+1];
    textString.ConvertToWChar(ptxtChar);
    piDftText->SetString(ptxtChar);
```

```
    delete [] ptxtChar;
    ptxtChar = NULL;
    // 设置字符串字体大小
    CATIDftTextProperties *piDftTextProp = NULL;
    if (SUCCEEDED(piDftText->GetTextProperties(&piDftTextProp)))
    {
      piDftTextProp->SetBold(TRUE);
      piDftTextProp->SetItalic(TRUE);
    }
  }
  const double txtpos2[2] = {940., 40.};
  if (SUCCEEDED(spAnnFactory->CreateDftText(txtpos2,&piDftText)))
  {
    // 设置字符串
    CATUnicodeString textString("DASSAULT \nSYSTEMES");
    wchar_t *ptxtChar = new wchar_t[textString.GetLengthInChar()+1];
    textString.ConvertToWChar(ptxtChar);
    piDftText->SetString(ptxtChar);
    delete [] ptxtChar;
    ptxtChar = NULL;
    CATIDftTextProperties *piDftTextProp = NULL;
    //设置字体，加粗，斜体
    if (SUCCEEDED(piDftText->GetTextProperties(&piDftTextProp)))
    {
      piDftTextProp->SetBold(TRUE);
      piDftTextProp->SetItalic(TRUE);
    }
  }
  const double txtpos3[2] = {940., 54.};

  }
  // 轴线和中心线创建
  CATIDrwAxisLine_var axisline = spAnnFactory->CreateDrwAxisLine(Line1,Line2);
  CATIDrwCenterLine_var centerline = spAnnFactory->CreateDrwCenterLine(Cercle1);
```

4 从 3D 模型中创建剖视图

4.1 创建并初始化文件

首先应该用程序打开要创建视图的 Part 文件模型，并获取其中的剖切面。

```
CATDocument *pDocPart = NULL;
 if( SUCCEEDED(CATDocumentServices::OpenDocument(pfileNamePart, pDocPart)) &&
pDocPart)
```

```
 {
    CATInit_var spInitOnDoc(pDocPart);
    if(NULL_var != spInitOnDoc)
    {
      // 获取根 container
      CATIPrtContainer * piPrtCont = (CATIPrtContainer*)
spInitOnDoc->GetRootContainer("CATIPrtContainer");
      if (piPrtCont)
      {
        // 获取 Part 特征
        CATIPrtPart_var spPart = piPrtCont->GetPart();

        // 获取用来做剖视图的草图用名称来匹配
        CATIDescendants *piDescPart=NULL;
        if (SUCCEEDED(spPart->QueryInterface(IID_CATIDescendants,(void**)&piDescPart)))
        {
          CATListValCATISpecObject_var listFeatures;

          piDescPart->GetAllChildren ("CATISketch",listFeatures) ;

          int nbChilds = listFeatures.Size();
          CATISketch* piSketch = NULL;
          CATISpecObject_var spFeat;
          for (int i = 1; i <= nbChilds; i++)
          {
            spFeat = listFeatures[i];
            if (NULL_var != spFeat)
            {
              if (SUCCEEDED(spFeat->QueryInterface(IID_CATISketch, (void**)&piSketch)))
              {
                CATIAlias *piSketchAlias = NULL;
                if (SUCCEEDED(piSketch->QueryInterface(IID_CATIAlias,
(void**)&piSketchAlias)))
                {
                  CATUnicodeString SketchName = piSketchAlias->GetAlias();
          const CATUnicodeString SketchSection_UC = "SketchForSection";
          if (SketchName == SketchSection_UC)
          hr = CreateSectionViewFromSketchInDrawingDoc(pNewDoc, piSketch);
...
```

通过 *CATIDescendants* 接口采用 GetAllChildren 获取所有的 CATISketch 类型接口。再采用 GetAliasmethod 方法获取草图名称。用名称匹配得到要创建剖视图的草图。

4.2 用 SketchForSection 草图创建剖视图

```
// 从 3D 草图平面创建一个生成视图
HRESULT CreateSectionViewFromSketchInDrawingDoc(CATDocument *ipNewDoc, CATISketch *ipi3DSketch)
{
  HRESULT hr = E_FAIL;
  if (ipNewDoc && ipi3DSketch)
  {
    // 创建绘图标准
    // 通过 CATIDftDocumentServices 接口获取 Drawing 特征
    CATIDftDrawing *piDftDrawing = NULL;
    CATIDftDocumentServices *piDftDocServices = NULL;
    CATIContainer_var spDrwCont;
    if (SUCCEEDED(ipNewDoc->QueryInterface(IID_CATIDftDocumentServices, (void **)&piDftDocServices)))
    {
      piDftDocServices->GetDrawing(IID_CATIDftDrawing, (void **)&piDftDrawing);
      piDftDocServices->Release();
      piDftDocServices = NULL;
      if (piDftDrawing)
      {
        CATISpecObject *piDrawingSO=NULL;
        if (SUCCEEDED(piDftDrawing->QueryInterface(IID_CATISpecObject,(void **)&piDrawingSO)))
        {
          spDrwCont = piDrawingSO->GetFeatContainer();
          if (NULL_var != spDrwCont)
          {
            CATIDftStandardManager *piStdmgr = NULL;
            hr = spDrwCont->QueryInterface(IID_CATIDftStandardManager,(void**)&piStdmgr);
            if (SUCCEEDED(hr))
            {
              // 获取标准列表
              CATIStringList *piListstd = NULL;
              if ( SUCCEEDED(piStdmgr->GetAvailableStandards(&piListstd)) && piListstd )
              {
                unsigned int nbrstd = 0;
                piListstd->Count(&nbrstd);
                for (unsigned int indice = 0; indice < nbrstd; indice ++)
                {
                  wchar_t *wstd = NULL;
                  if ( SUCCEEDED ( piListstd->Item ( indice, &wstd ) ) && wstd )
```

```
                    {
                CATUnicodeString stdname;
                const CATUnicodeString ISO_UncS = "ISO";
                stdname.BuildFromWChar(wstd);
                if ( stdname == ISO_UncS )
                {
                        // 导入 ISO 标准
                        piStdmgr->ImportStandard (wstd);
                        break;
                    }
                    delete[] wstd; wstd = NULL;
                  }
                }
                piListstd->Release(); piListstd=NULL;
              }
              piStdmgr->Release (); piStdmgr=NULL;
            }
          }
          // 给 CATDrawing 文件的当前图纸从 3D 草图中创建新的剖视图
          CATIDftView *piDftSectionViewFrom3D = NULL;
          CATIDftSheet *piDftSheet = NULL;
          piDftDrawing->GetActiveSheet(&piDftSheet);
          // 固定视图指针定义
          double ptOrigin[2] = {150.0,150.0};
           CATMathVector normalSketch;
          CATI2DLine_var spFirstLine;
      double pOrthoDirection[2];
          CATIDftGenViewFactory *piDftGenViewFact = NULL;
          if (piDftSheet &&
SUCCEEDED(piDftSheet->QueryInterface(IID_CATIDftGenViewFactory,(void
**)&piDftGenViewFact)))
          {
            // vecPro 参数
            // VecPro 是垂直于草图截面轮廓的一个向量（垂直于剖切线的方向）
            // 这个方向向量确定了哪个 part 文件将在投影时会看到。
            CATLISTV (CATI2DWFGeometry_var ) GeomList;
            if
(SUCCEEDED(ipi3DSketch->GetComponents(CATI2DWFGeometry::ClassName(),GeomList)))
            {
              if (GeomList.Size() > 1)
              {
                int indice=1;
                while (indice < GeomList.Size())
                {
```

```
                    spFirstLine = GeomList[indice];
                    if (NULL_var != spFirstLine)
                    {
                       double pOrigin[2],pDirection[2];
                       spFirstLine->GetLineData(pOrigin, pDirection);
                       pOrthoDirection[0] = pDirection[1];
                       pOrthoDirection[1] = -pDirection[0];
                       break;
                    }
                    indice++;
                 }
              }
           }
           CATISpecObject_var spPlanarSupport;
           if (SUCCEEDED(ipi3DSketch->GetPlanarSupport(spPlanarSupport )))
           {
              CATPlane_var spPlane= spPlanarSupport;
              if(NULL_var != spPlane)
              {
                 CATMathPlane mathPlaneSk;
                 spPlane->GetAxis(mathPlaneSk);
                 CATMathPoint ThePoint;
                 mathPlaneSk.EvalPoint(pOrthoDirection[0],pOrthoDirection[1],ThePoint);
                 double XCoord = ThePoint.GetX();
                 double YCoord = ThePoint.GetY();
                 double ZCoord = ThePoint.GetZ();
                 CATMathDirection vecPro;
                 vecPro.SetCoord(XCoord,YCoord,ZCoord);
                 //偏移轮廓
                 int viewProfile = 0;
                 //  草图平面定义轮廓
                 CATCell *piPlaneElem = NULL;
                 CATBody *piBody= NULL;
                 CATMathPoint iLimitPoints[2];
                 CATIProduct *piProduct= NULL;
                 //创建与草图相关联的剖视图
                 hr        =        piDftGenViewFact->CreateStandAloneSectionView(ptOrigin,
DftSectionView, vecPro, viewProfile, ipi3DSketch,piPlaneElem,piBody,iLimitPoints,piProduct,
&piDftSectionViewFrom3D);
                 piDftGenViewFact->Release();piDftGenViewFact=NULL;
              }
           }
        }
        piDrawingSO->Release();
```

```
            piDrawingSO=NULL;
          }
        piDftDrawing->Release();
        piDftDrawing = NULL;
      }
    }
  }
  return hr;
}
...
```

这段程序 CreateSectionViewFromSketchInDrawingDoc 通过 3D 草图创建了一个剖视图，其中使用了 CATIDftGenViewFactory 接口中的 CreateStandAloneSectionView 方法。这个接口是由草图接口进行实现的，制图标准是在剖视图创建之前设置完成的。主要的参数是：

ptOrigin: 剖视图的固定指针

DftSectionView: 剖视图的类型

VecPro: m 垂直于剖切线的方向，这个方向确定了哪些零件可以看到。

ipi3DSketch: 3D 草图.

从帮助文档中可以看到 **CreateStandAloneSectionView 的创建剖视图用到的具体参数，**而草图并非必须要用的参数。

o **CreateStandAloneSectionView**

```
public virtual HRESULT CreateStandAloneSectionView(double*          iptOrigin,
                                                   CATDftViewType   iViewType,
                                                   CATMathDirection iVecpro,
                                                   int              iViewProfile,
                                                   CATISketch*      ipi3DSketch,
                                                   CATCell*         ipiCell,
                                                   CATBody*         ipiBody,
                                                   CATMathPoint[2]  iLimitPoints,
                                                   CATIProduct*     ipiProduct,
                                                   CATIDftView**    opiSectionView) = 0
```

Create a Standalone Section View. **Role**: Create a standalone section view with associative cutting profile to 3D element. The authorised 3D elements are either a sketch element or plannar face or a plan.

Parameters:

iPtOrigin[2]
: Coordinates of anchor point of the section view created.

iViewType
: Type of the View:
Legal values: DftSectionView,DftSectionCutView

iVecPro
: The direction of projection of the view: This direction must be contained in the plane of the section and must be: Perpandicular to the first line of the Sketch for profile defined by a Sketch. Perpandicular to the line created by two limit points of the profile for profile defined by a Plane or a planar face.

iViewProfile
: Type of the profile:
Legal values: 0: Offset, 1: Aligned

i3DSketch
: The cutting profile is defined by a 3D sketch containing the geometry's description of the section profile
Legal values: if NULL, The profile must be defined by a plan or a plane face in the drawing

ipiCell
: The cutting profile is defined by a plan or plane face.
Legal values: if NULL, The profile must be defined by a sketch

ipiBody
: Associated body if the cutting profile is defined by a plan or plane face.
Legal values: NULL, if the profile is defined by a sketch.

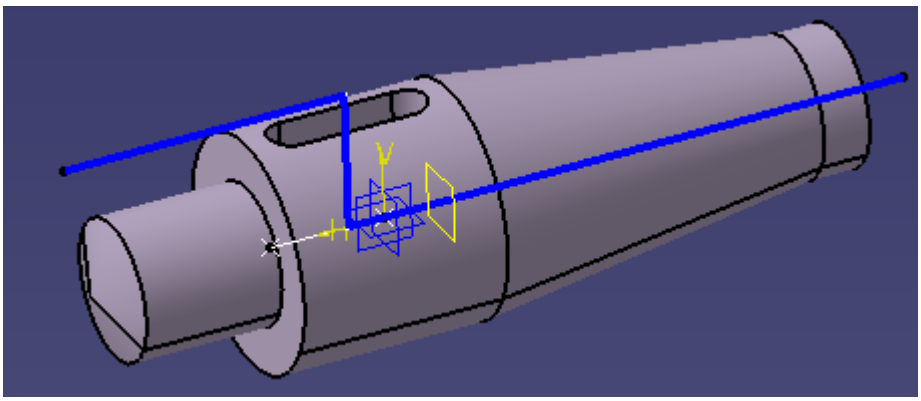
iLimitPoints
: 2 points limits of the cutting profile, if the cutting profile is defined by a plan or a plane face.

iProduct

5 制图程序应用

源程序编译运行步骤：

A.手动打开 part 文件 PartWithPlaneAndSketchForSectionView.CATPart



B.打开对话框 TrainDraftDlgCmd

C.点击创建剖视图，程序将会创建 D:\NewDrawing.CATDrawing 文件，并创建剖视图。

D.手动打开 D:\NewDrawing.CATDrawing 文件，打开对话框，点击“创建视图几何对象”按钮。完成标题栏的创建。