Generative Knowledge

- Preface
- Getting Started
- Workbench and Interactive Tasks
- Reference Library

P1

P2

Preface

The CATIA Generative Knowledge product provides you with an easy way to describe the data used to create documents and to store this data in a script file so that it can be re-used later on. Here are the benefits of using the generative script:

- the script code itself is small in size (a few K)
- the syntax is easy to grasp
- it executes rapidly from CATIA
- it allows you to add knowledgeware information in the form of expert rules and checks to a document and store the whole data in a readable file.

Moreover, it offers a full compatibility with external documents by providing you with an import capability. Features belonging to external documents can be referred to and instantiated in a script. Of course, a document which has been generated from a script can be modified interactively.

The present version of the product covers a broad range of mechanical features as well as some knowledgeware features.

Getting Started

The scenario which is developed below helps you begin learning a new area of the CATIA knowledgeware capabilities. It is broken down into steps and all the instructions required by the user are supplied for each steps.

This scenario is just a quick look at the Generative Knowledge product capabilities. From any stage of this basic scenario you can access in-depth information by using the links to the Reference Library.

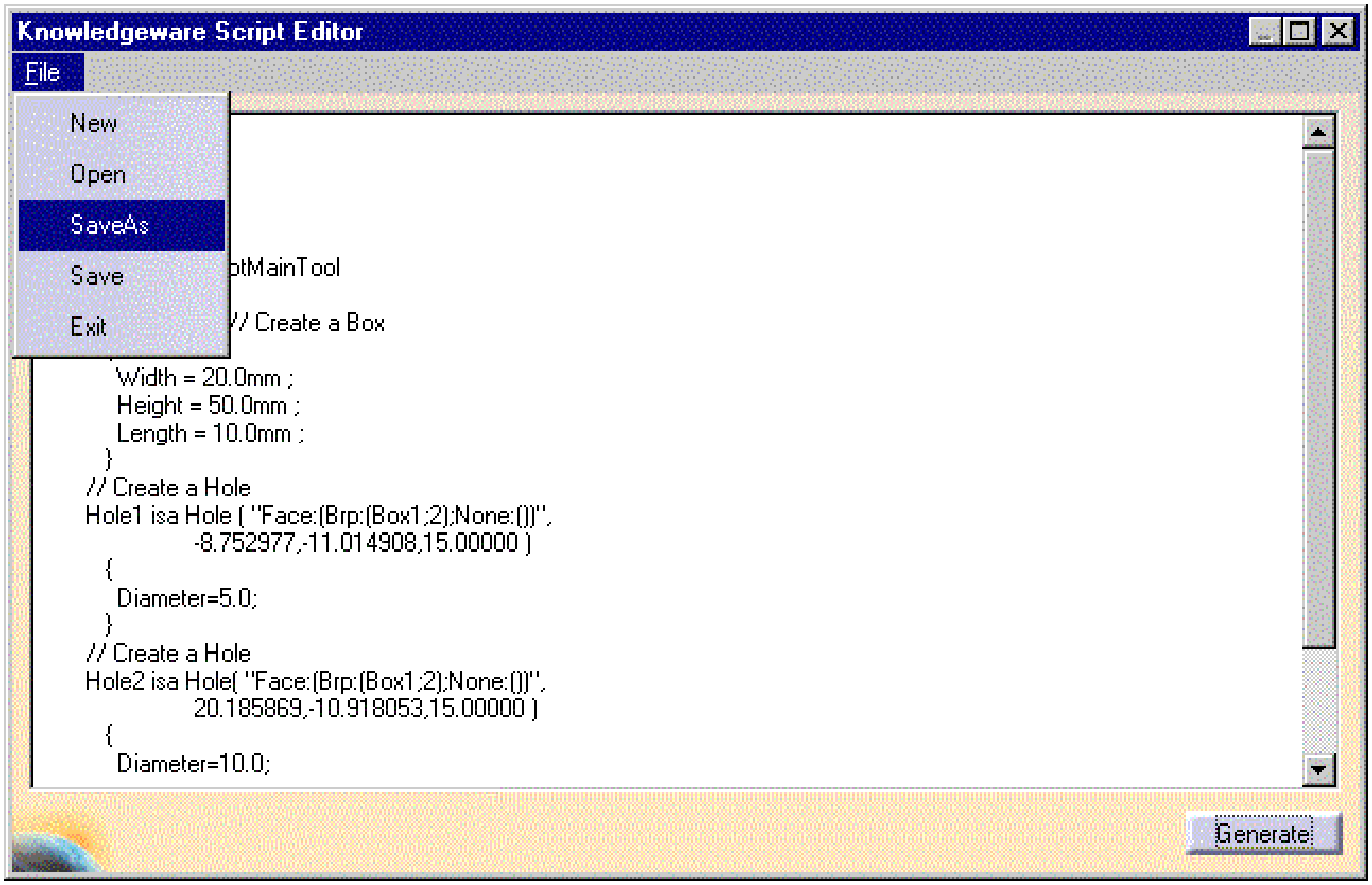
To carry out the scenario below, you don't need to have any initial sample on hand as you create your initial data from a script.

The basic methods for writing a script are conceptually simple and straightforward, they don't require any prerequisite knowledge but it is better if you already dabble in some other script languages manipulating objects and properties.

1. Select the Infrastructure->Generative Knowledge command from the Start menu, then click the icon. The Generative Script Editor is displayed.
2. Copy/Paste the script below from your browser to the Script editor:

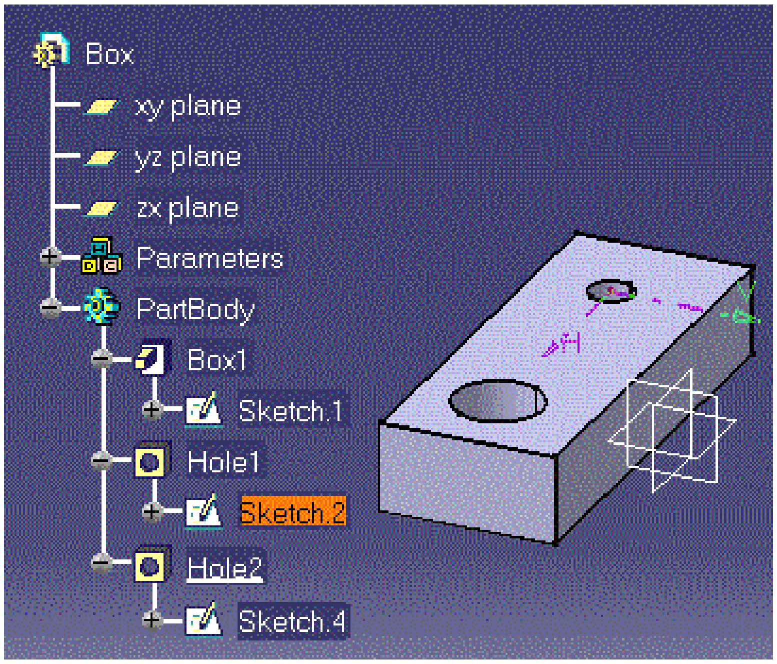
```
Box isa CATPart {
  BoxPart isa Part
   {
     PartBody isa Body
      {
        Box1 isa Box  // Create a Box
           {
             Width = 20.0mm ;
             Height = 50.0mm ;
             Length = 10.0mm ;
           }
        // Create a Hole
        Hole1 isa Hole ( "Face:(Brp:(Box1;2);None:())",
                         -8.752977,-11.014908,15.00000 )
           {
             Diameter=5.0;
           }
        // Create a Hole
        Hole2 isa Hole( "Face:(Brp:(Box1;2);None:())",
                         20.185869,-10.918053,15.00000 )
           {
             Diameter=10.0;
           }
      }
   }
```

3. Save your script file by using the File->SaveAs function of the script editor (this step is optional but recommended).



```
                     ptMainTool
                     // Create a Box
       Width = 20.0mm ;
       Height = 50.0mm ;
       Length = 10.0mm ;
     }
   // Create a Hole
   Hole1 isa Hole ( "Face:(Brp:(Box1;2);None:())",
                 -8.752977,-11.014908,15.00000 )
     {
       Diameter=5.0;
     }
   // Create a Hole
   Hole2 isa Hole( "Face:(Brp:(Box1;2);None:())",
                 20.185869,-10.918053,15.00000 )
     {
       Diameter=10.0;
```

4. Click Generate. The following document is generated in the geometry area.



Three geometric features have been created: Box1, Hole1 and Hole2. You can double-click the features created to display their parameter values and compare them with the values specified in the script. Here are a few comments about the resulting document with respect to the initial script:

. Objects are created with the isa function.
2. At creation, features are assigned default parameter values.
3. Parameter values can be (re-) defined in the script in statement blocks surrounded by curly braces.
4. In the Hole creation instruction, the complicated expression within the parentheses defines a point. Don't worry about it, you won't have to type such an expression. It is captured from the geometry area. See Capturing Data from the Geometry Area.

Workbench and Interactive Tasks

The Generative Script workbench contains only one icon which allows you to open the script editor. In most CATIA products, the dialog behind an icon is related to a feature creation and the feature created is specific to the application.

In the Generative Script product, when you click the script editor icon, you are provided with a means to create a script. This script is NOT a feature, you won't see it in the specification tree. But from this script, you can generate features that will be displayed in the specification tree.

Using the script editor is quite straightforward. Here is a help for those of you which are not already familiar with CATIA editing windows.

Creating a Script

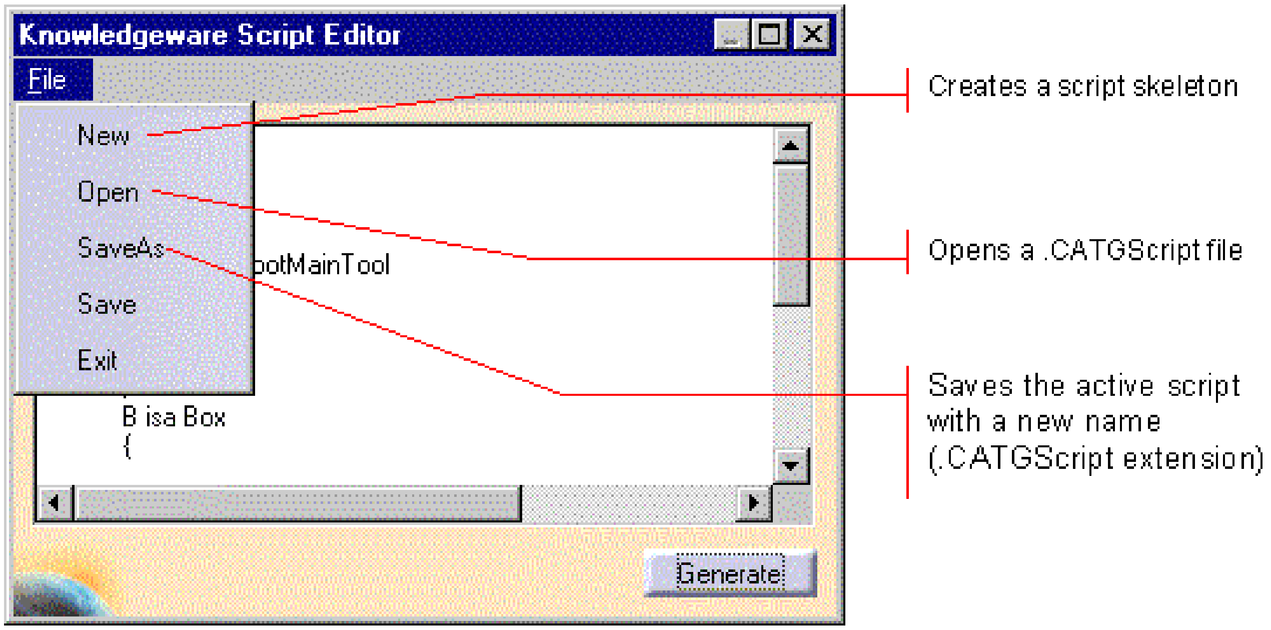
Using a Script Skeleton

Generating a Document from a Script

Creating a Script

This task describes how to create a script for an interactive standpoint. The main task is to write the proper statements. To do so, refer to Reference Library

1. Access the Generative Knowledge workbench by selecting the Generative Knowledge function from the Start Menu.
2. Click the icon. The Generative Script Editor is displayed.

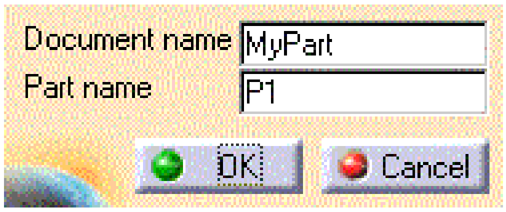

Knowledgeware Script Editor
File
New
Open
SaveAs
Save
Exit
potMainTool
B isa Box
{
Generate
Creates a script skeleton
Opens a .CATGScript file
Saves the active script with a new name (.CATGScript extension)

3. Enter your script in the editor. Refer to the Reference Library for information on the language syntax as well as samples illustrating how to code your script.
4. Save your script by using the File->Save or File->SaveAs command. Your script to be saved under a file with .CATGScript extension.

Using a Script Skeleton

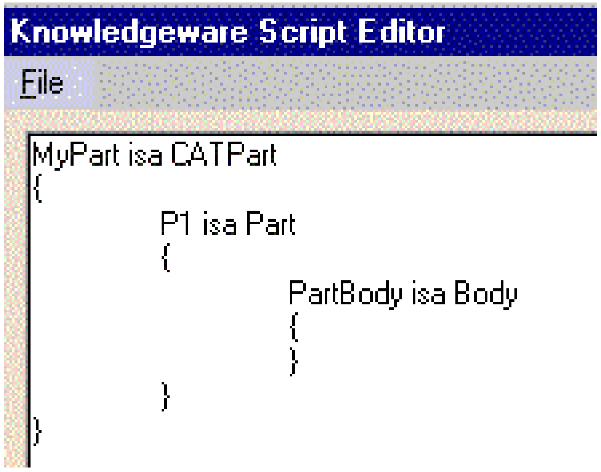
To help you write your script, the Generative Script Editor provides you with a way to create a skeleton which reflects the structure of a CATPart document.

1. Access the Generative Knowledge workbench by selecting the Infrastructure-> Generative Knowledge function from the Start menu.
2. Click the icon. The Generative Script Editor is displayed.
3. In Generative Script Editor, select the File->New->CATPart document function. The dialog box below is displayed.



Document name: MyPart
Part name: P1

4. Fill in the fields and click OK. The associated script is created



```
MyPart isa CATPart
{
        P1 isa Part
        {
                PartBody isa Body
                {
                }
        }
}
```

5. To enrich this script, refer to Reference Library.
6. Save your script or generate the related document.

Generating a Document from a Script

The task which consists in generating a document from a script is quite straightforward. This is how you have to proceed. Note that only CATPart document can be created in this CATIA version.

1. Access the Generative Knowledge workbench by selecting the Infrastructure-> Generative Knowledge function from the Start menu.
2. Click the icon. The Generative Script Editor is displayed.
3. Enter your script in the editor or open an already existing .CATGScript file.
4. Click Generate. The related document is created in the geometry area. The Reference Library gives you a number of examples.
5. If need be, save your script before exiting the editor.

You can add a new feature to an already existing part provided the name of the part in the script describing this new feature is the same as the name of the part you want to add the new feature to. This capability applies to parts generated either from a script or from the Part Design workbench.

Reference Library

The reference library describes:

- the language syntax including the code structure, how to use comments, variables and so on.
- the objects as well as their properties. They are divided into two categories: the geometric features and the knowledgeware features
- the functions to create objects or instantiate objects from an imported file
- the operators used to specify where a parameter value is to be read in the script
- the constraints and transformations that can be applied to objects to specify their location in space.

Geometric Features

- body
- box
- chamfer
- cone
- cylinder
- fillet
- pad
- hole
- pattern
- shaft
- shell
- sphere
- torus

Knowledgeware Features

- expert check
- expert rule
- formulas

Functions

- import
- isa

Special Characters & Operators

- ?
- ..\..

Constraints & Positioning

- coaxiality
- coincidence
- distance
- position
- rotate
- translate

The Generative Script Language

Overview

The CATIA Generative Knowledge Script is a simple programming language dedicated to the CATIA feature creation. It provides you with a means to generate most geometric and knowledgeware features making up CATIA documents from a scripting code. The filename extension for generative scripts is .CATGScript

- The Script Structure
- Comments
- Variables
- Capturing Data from the Geometry View
- Specifying Input Data
- Limitations

A Few Basics

The Generative Knowledge Script is quite simple to learn and you can learn about this language in little time.To get started programming, take a look at some examples by clicking on some items of the geometric feature list in the Reference Library. Like other scripting languages, the Generative Knowledge script manipulates objects and object properties. In addition, it provides you with a means to import the definitions of documents, then instantiate objects from an import file.

The Script Structure

A generative script is written in text format and is organized into statements, blocks consisting of related sets of statements, and comments. Within a statement, you can use variables and to define variables, you can use formulas.

Generally a block consists of an instruction to create an object followed by a set of statements surrounded by braces (**{ }**). Statement blocks can be nested and the most enclosing one corresponds to the document creation. When defining properties, the semicolon (;) is a terminator.

In the outermost statement block, you must create the document intended to contain all the features to be created later on. Right below, you must create a part. Within a part, you can create the following objects:

- a RootMainTool, i.e. the object which represents the part body feature
- one or more bodies
- a rule base.

```
mydocument isa CATPart
 {
  Part isa Part
   {
    PartBody isa Body
       {
        myObject isa objectType
          {
           property1 = ... ;
           property2 =  ... ;
            ...
          }
        ...
      }
     ...
   }
 }
```

Note that CATPart documents are the only document types you can create at present with a script.

You can add a new feature to an already existing part provided the name of the part in the script describing this new feature is the same as the name of the part you want to add the new feature to. This capability applies to parts generated either from a script or from the Part Design workbench.

Comments

Multiline comments (/* ... */) are not supported. A single-line comment begins with a pair of forward slashes(//).

```
   Sphere1 isa Sphere // Creates a sphere
        {
            // Valuates the Radius property
            Radius = 15.0 ;
        }
```

Variables

You declare variables explicitly in your script as follows:

ALPHA = 45 deg **;**

Unlike in most script languages, a variable's scope is not really determined by where you declare it. From anywhere in your script, you can access a variable by using the ..\.. and ? operators. After the script is finished running, the variable declared in your script still exists as a document parameter.

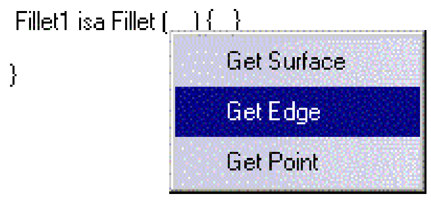
Capturing Data from the Geometry View

When creating an object or defining a property within your script, you may have to define objects such as surfaces, edges or points. The Script Editor provides you with a contextual menu whereby you can capture the required data from the geometry view. To capture data from the geometry view:

1. type the statement skeleton with the required parentheses and braces. For example:

```
 Fillet1 isa Fillet ( ) { }
```

2. right-click anywhere inside the parentheses and select the 'Get Edge', 'Get Surface' or 'Get Point' function from the contextual menu.



```
Fillet1 isa Fillet ( ) { }
}
```

3. in the geometry area, select the edge (or the face/point) whose data is to be captured.
   The object data is inserted in your script:

```
 Fillet1 isa Fillet("Face:(Brp:(Box1;2);None:())")
{ // object properties }
```

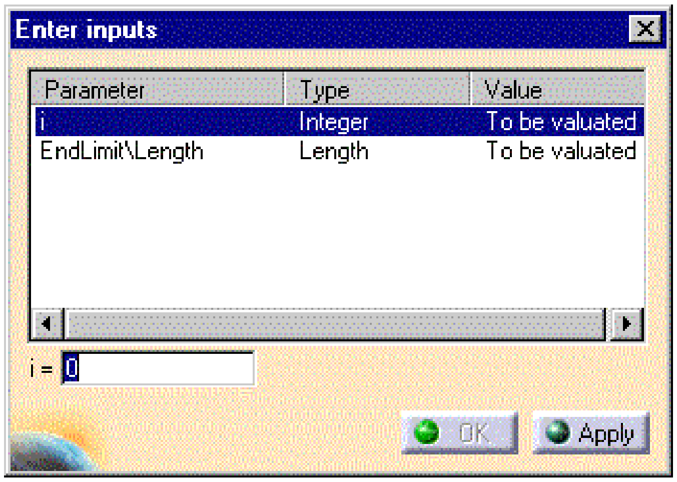
If need be, you can define or redefine the object properties within the braces.

Specifying Input Data

Parameters can be declared as input data.

```
Pad1 isa CATPart
 {
   Part isa Part
    {
      i = 0, Input : Integer;
      PartBody isa Body
       {
         P isa Pad
          {
            EndLimit\Length = 8mm, Input :Length;
          }
       }
    }
 }
```

Clicking 'Generate' displays a dialog box which prompts you to enter the data values.



Parameter	Type	Value
i	Integer	To be valuated
EndLimit\Length	Length	To be valuated

i = 0

Enter one-by-one a value for each parameter and click Apply. Once all the data is entered, click OK to generate the document.

Limitations

You should be aware of some restrictions:

- Features as well as constraints can only be created in CATPart documents.
- When a rotate or translate transformation is applied to a body, only the body first feature is rotated or translated.
- Instances of sketch-based features cannot be moved apart from their prototype.
- Any parameter used as an argument in a formula should be preceded by the ? symbol. The syntax X = 2 * Y is invalid and should be replaced with X = 2 * **?** Y.
- Unless a formula-defined parameter has not been initialized with the proper units, the value calculated from the formula is dimensionless.
  Y = 0 kg ;
  Y = 2 * ? X ;
- A script error stops the reading and the execution of the script.

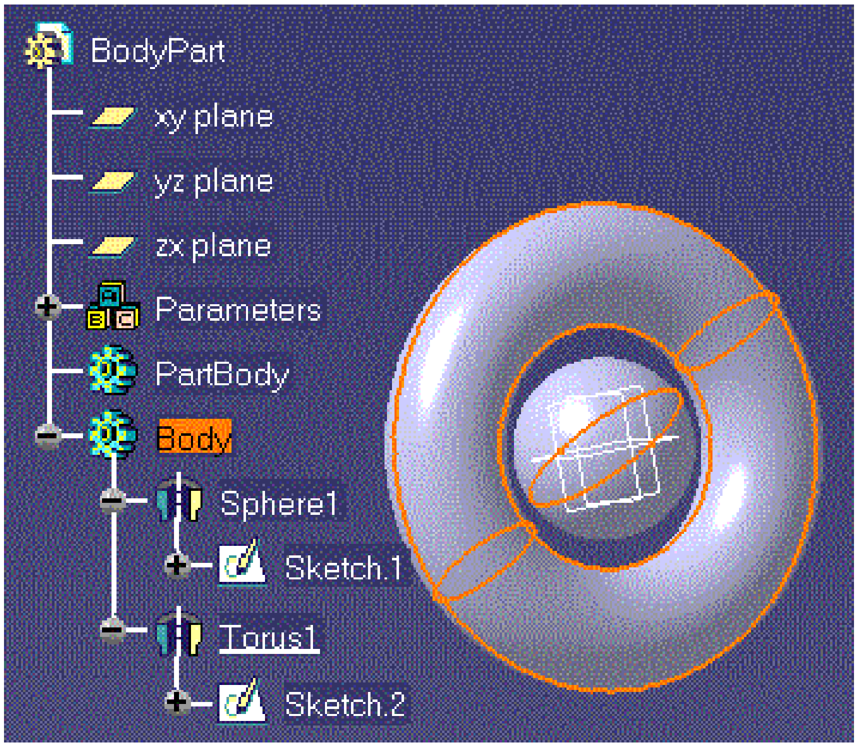
Body Object

Definition

A body is the combination of several features within a part. ***It properties are the features which are embodied in it***.

Example

```
BodyDoc isa CATPart
{
 BodyPart isa Part
  {
    Body isa Body
      {
       Sphere1 isa Sphere
        {
         Radius = 15.0mm;
        }
       Torus1 isa Torus
        {
         InnerRadius = 20.0mm ;
         SectionRadius = 10.0mm ;
        }
      }
   }
}
```



Box Object

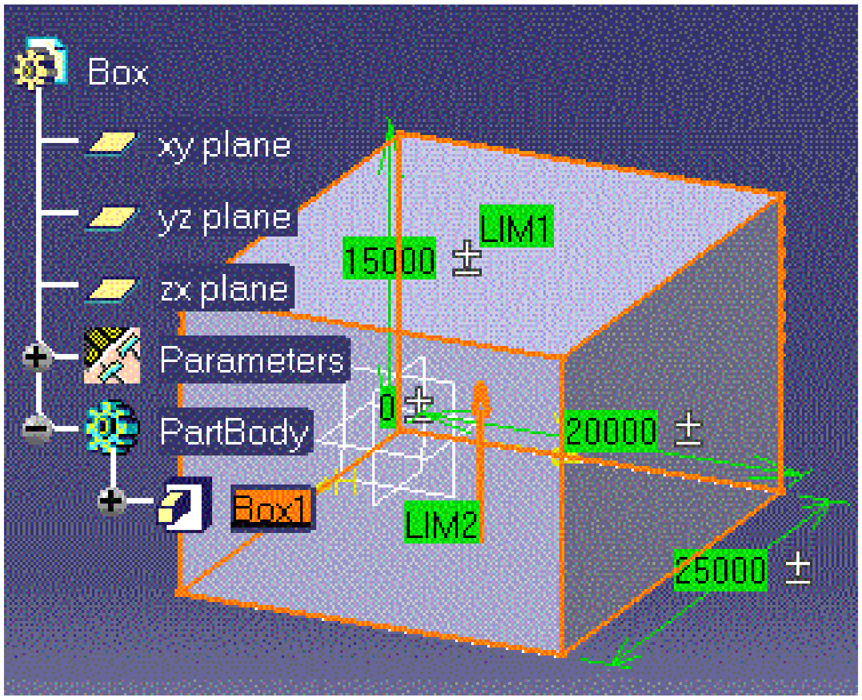
Definition

A box is a pad extruded from a rectangular sketch. It is defined by three properties:

- the ***Length*** which is the pad first limit
- the ***Width*** and the ***Height*** which define the initial sketch.

Example

```
Box isa CATPart
 {
 BoxPart isa Part
  {
   PartBody isa Body
    {
     Box1 isa Box
       {
        Width = 20.0m ;
        Height = 25.0m ;
        Length = 15.0m ;
       }
     }
   }
 }
```



Chamfer Object

Definition

A chamfer is a cut through the thickness of a part at an angle, giving a sloping edge.   It is defined by two properties:

- the ***ChamferRibbon.i\Angle***
- the ***ChamferRibbon.i\Length1***.

To specify a chamfer within your script, you must have a part open, then:

1. create a Chamfer by using the isa function
   `Chamfer1 isa Chamfer ( ) { }`
2. right-click anywhere inside the parentheses and select the 'Get Edge' or the 'Get Surface' function from the contextual menu.
3. in the geometry area, select the edge to be chamfered.

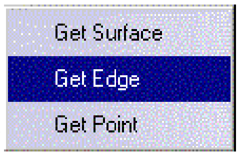
The edge definition is inserted inside the parentheses. A 45 deg chamfer is created by default. To modify the angle or Length1 values, specify the property value within the braces.

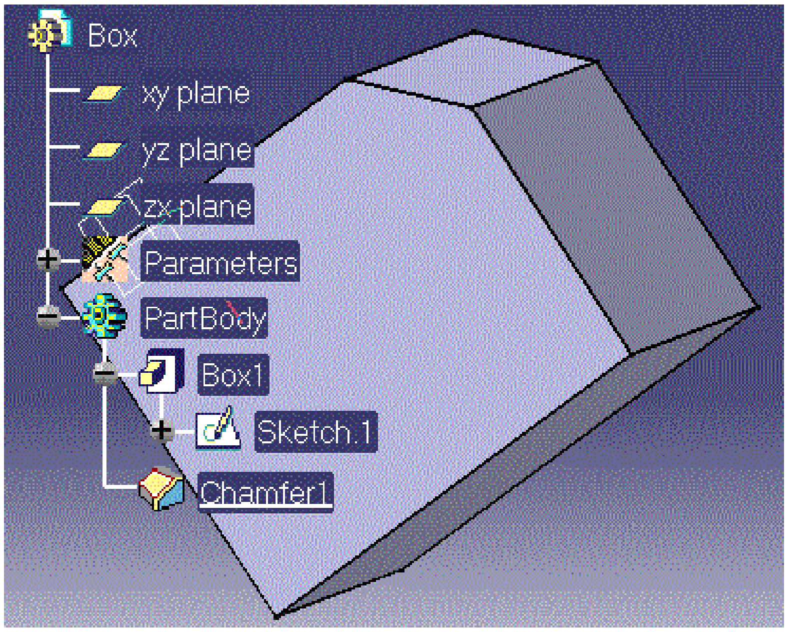
Example

```
Box isa CATPart
  {
    BoxPart isa Part
      {
        PartBody is a Body
          {
            Box1 isa Box
               {
                 Width = 20.0m ;
                 Height = 25.0m ;
                 Length = 15.0m ;
                }



            Chamfer isa Chamfer ( )
                {
                }
         }
      }
  }
```





Cone Object

Definition

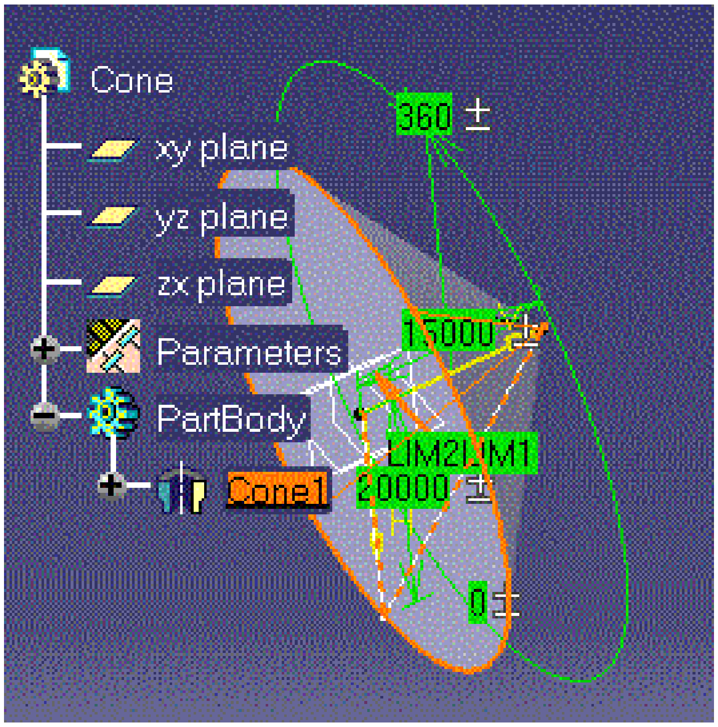
A cone is a shaft created by rotating a triangular sketch. It is defined by two properties:

- the ***Length***
- the ***Radius***.

Unless otherwise specified, the length unit is the meter.

Example

```
Cone  isa CATPart
{
 ConePart isa Part
  {
   PartBody isa Body
    {
     Cone1 isa Cone
       {
        Radius = 20.0 ;
        Length = 15.0 ;
       }
    }
  }
}
```



Cylinder Object

Definition

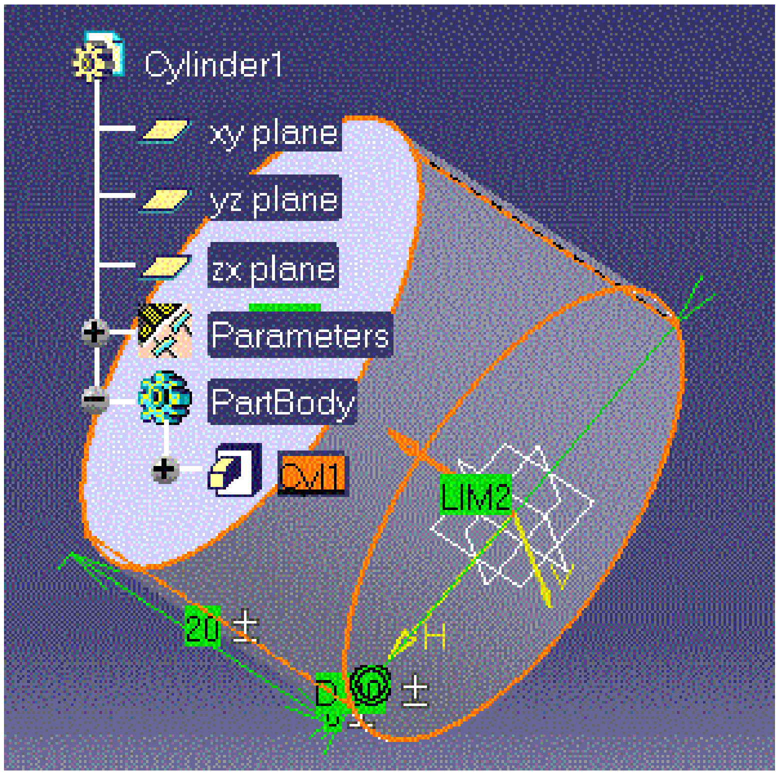
A cylinder is a pad created by extruding a circular sketch. It is defined by two properties:

- the ***Length***
- the ***Radius***.

Unless otherwise specified, the length unit is the meter.

Example

```
Cylinder1 isa CATPart
  {
   Part isa Part
    {
      PartBody isa Body
        {
          Cyl1 isa Cylinder
             {
                Radius=15.0mm;
                Length=30.0mm;
             }
        }
     }
  }
```



Fillet Object

Definition

A fillet is a curved surface of a constant or variable radius that is tangent to, and that joins two surfaces. Together, these three surfaces form either an inside corner or an outside corner. It is defined by two properties:

- the ***CstEdgeRibbon.i\Radius***
- the ***activity***.

To specify a chamfer within your script, you must have a part open, then:

1. create a Fillet by using the isa function
   `Fillet1 isa Fillet ( ) { }`
2. right-click anywhere inside the parentheses and select the 'Get Edge' or the 'Get Surface' function from the contextual menu.
3. in the geometry area, select the edge or the face to be filleted.

The edge or face definition is inserted inside the parentheses. A 5mm radius fillet is created by default. If need be, specify a new radius value within the braces right after the fillet creation statement. Retrieve the parameter name from the ***f(x)*** parameter list.

Example

```
 Box isa CATPart
  {
   BoxPart isa Part
     {
      PartBody isa Body
        {
         Box1 isa Box
           {
            Width = 20.0 ;
            Height = 25.0 ;
            Length = 15.0 ;
           }
         // Use the contextual menu to retrieve
         // the surface definition
         Fillet1 isa Fillet ("Face:(Brp:(Box1;2);None:())" )
           {
            CstEdgeRibbon.1\ Radius =1.0;
            }
        }
     }
  }
```

Pad Object

Definition

A pad is a feature created by extruding a sketch. It can be defined by three properties:

- the ***sketch***
- the ***FirstLimit\Length*** (StartLimit\Length)
- the ***SecondLimit\Length*** (EndLimit\Length).

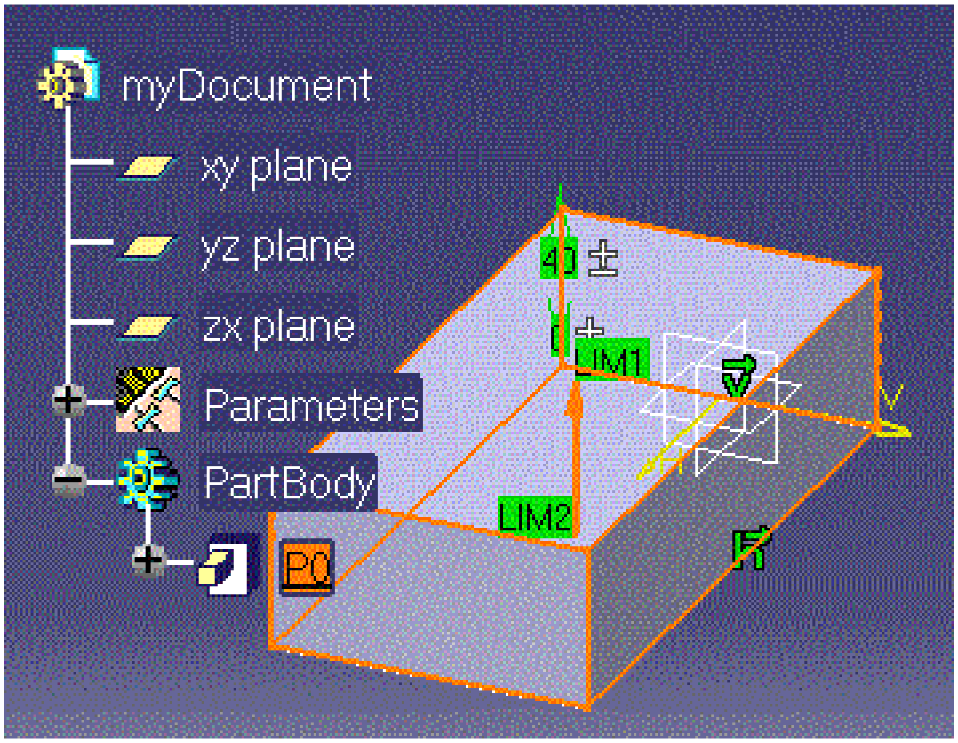
If no sketch is specified, a rectangular sketch is created by default. A limit which is not specified is set by default to zero.

Unless otherwise specified, the length unit is the meter.

Example

```
// Imports the KweRectangle document
import "~Samples\GenerativeKnowledge\KweRectangle.CATPart";
myDocument isa CATPart
{
 myPart isa Part
  {
   PartBody isa Body
    {
      Sketch isa Sketch.1
        {}
      P0  isa Pad("Sketch")
        {
         EndLimit\Length=40.0mm;
        }
    }
  }
}
```

Note: In the script above, the P0 pad is created from the Sketch.1 sketch which is imported from the KweRectangle.CATPart file.



Hole Object

Definition

A hole is an opening through a feature. It is defined by three properties:

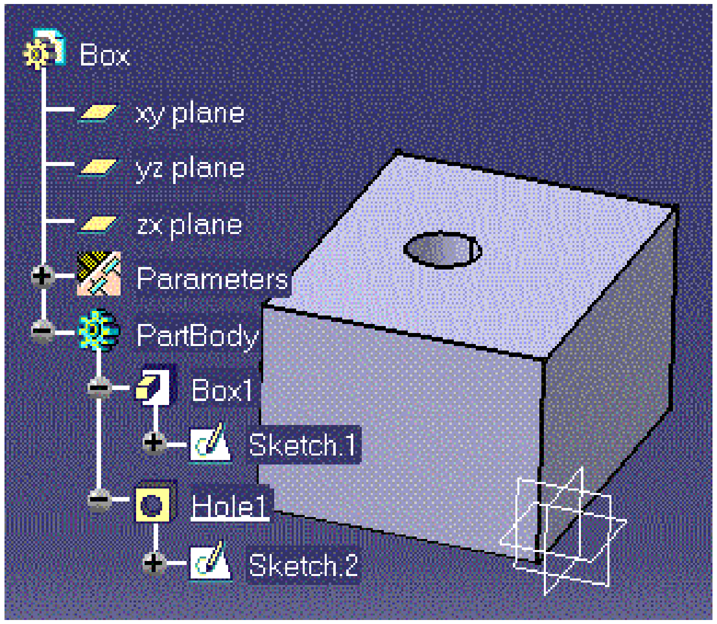
- the ***Diameter***
- the ***HoleLimit.1\Length***
- the ***Activity*** (1 deactivates the Hole, 0 activates the Hole).

To specify a hole within your script, you must have a part open, then:

1. create a Hole by using the isa function
   `Hole1 isa Hole ( ) { }`
2. right-click anywhere inside the parentheses and select the 'Get Point' function from the contextual menu.
3. in the geometry area, indicate a point on a face to define the hole axis.
   The hole definition is inserted inside the parentheses. If need be, modify any property by specifying a new value within the braces right after the hole creation statement.

Example

```
Box isa CATPart
 {
 BoxPart isa Part
  {
   PartBody isa Body
    {
     Box1 isa Box
        {
         Width = 20.0 ;
         Height = 25.0 ;
         Length = 15.0 ;
        }
     Hole1 isa Hole("Face:(Brp:(Box1;2);None:())",-7.0,-12.02,14.9)
      {
          Diameter=5.0;
          Activity=0;
      }
     }
   }
 }
```



Pattern Object

Definition

A pattern is a set of similar features repeated in the same part. Two types of patterns can be created with CATIA: the rectangular patterns and the circular patterns. At present, only rectangular patterns can be generated from a script. A rectangular pattern is defined by the following properties:

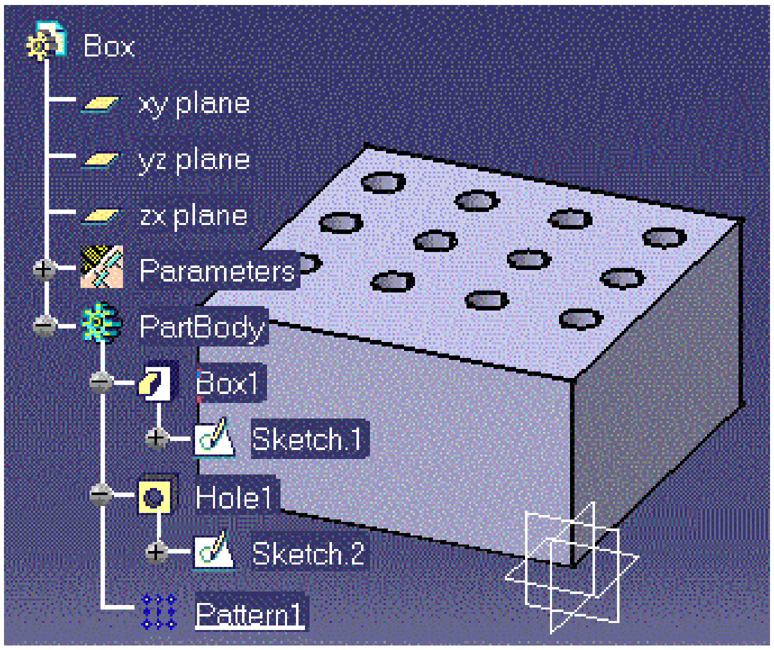
- ***Nb1***, the number of elements to be replicated along the first direction
- ***Nb2***, the number of elements to be replicated along the second direction
- ***Step1***, the element spacing along the first direction
- ***Step2***, the element spacing along the second direction
- ***activity***.

Syntax

`pattern1 isa pattern [NumberInDir1,NumberInDir2] of feature_to_be_repeated`

Example

```
  Box isa CATPart
  {
  BoxPart isa Part
   {
    PartBody isa Body
     {
      Box1 isa Box
        {
         Width = 20mm ;
         Height = 20mm ;
         Length = 10mm ;
        }
      Hole1 isa Hole (...)
         {
           Diameter = 2.0mm;
         }
       Pattern1 isa Pattern[3,4] of Hole1
         {
           Step1 = 5.0mm;
           Step2 = 5.0mm;
         }
 }
```



Shaft Object

Definition

A shaft is a feature created by rotating a sketch around and axis. A shaft has two attributes: the ***StartAngle*** and the ***EndAngle***. The sketch to be rotated must be imported from an external .CATPart document. This external document must also include a rotation axis.

Example

```
import "~\credocr3\Samples\KweEllipsis.CATPart";
myDocument isa CATPart
{
myPart isa Part
 {
   PartBody isa Body
      {
         Sketch isa Sketch.1 {}
         S0 isa Shaft("Sketch")
          {
            StartAngle = 20 deg ;
            EndAngle = 300 deg ;
          }
      }
  }
}
```

Shell Object

Definition

A shell is a hollowed out feature. It is defined by three properties:

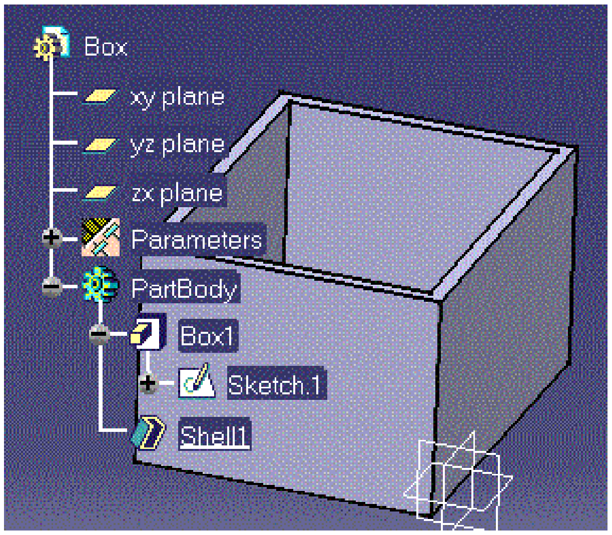
- the ***InnerThickness***
- the ***OutsideThickness***
- the ***Activity***.

To specify a shell within your script, you must have a part open, then:

1. create a Shell by using the isa function
   `Shell1 isa Shell ( ) { }`
2. right-click anywhere inside the parentheses and select the 'Get Surface' function from the contextual menu.
3. in the geometry area, select the face to be hollowed out.
   The face definition is inserted inside the parentheses. A 1mm thick shell is created by default. If need be, modify any property by specifying a new value within the braces right after the shell creation statement.

Example

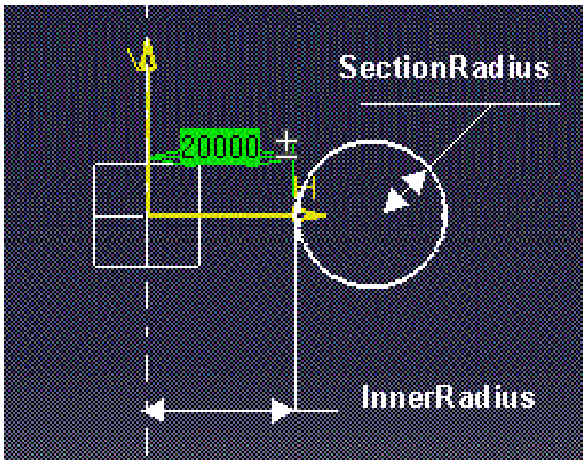
```
Box isa CATPart
  {
   BoxPart isa Part
     {
      PartBody isa Body
        {
          Box1 isa Box
            {
              Width = 20.0 m;
              Height = 25.0 m;
              Length = 15.0 m;
            }
          Shell1 isa Shell ( "Face:(Brp:(Box1;2);None:())" )
            {
            }
        }
    }
}
```



Torus Object

Definition

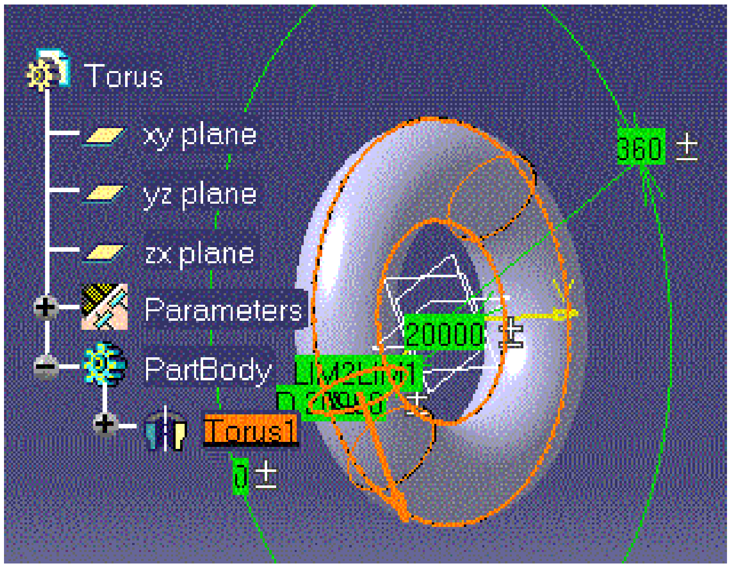
A torus is a shaft created by rotating a circular sketch aroung an axis. It is defined by two properties: the ***InnerRadius*** and the ***SectionRadius***.



Unless otherwise specified, the length unit is the meter.

Example

```
Torus isa CATPart
{
 TorusPart isa Part
  {
    PartBody isa Body
    {
      Torus1 isa Torus
      {
       InnerRadius = 20.0 m ;
       SectionRadius = 10.0 m;
      }
    }
  }
}
```



ExpertRule & ExpertCheck Objects

Definition

Expert Rules and Expert Checks are features generated with the CATIA Expert Knowledge product. Rules and Checks are regrouped into rule sets. Rule sets belong to a rule base. When writing a script with rules and checks you must comply with the RuleBase/RuleSet hierarchy. Refer to the CATIA Expert Knowledge product for more information on the concepts behind the expert rules and checks.

Expert rules as well as expert checks have two attributes:

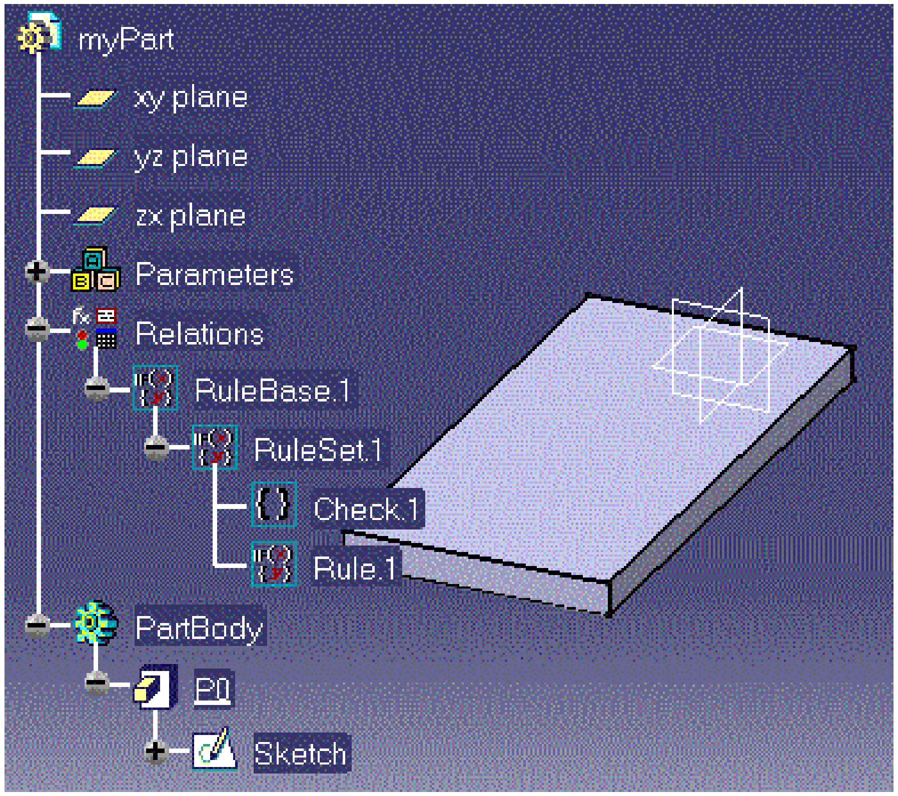
- the ***Variables*** attribute which must be defined with the appropriate syntax and corresponds to the
- the ***RuleBody*** (or ***CheckBody***) attribute which defines the rule (or check) statements.

Note: When specifying an attribute within a rule or a check, you can use either syntax:

`P.`*`attributename`* or `P\`*`attributename`* where P denotes the features the rule or check applies to (for example all the document pads if the ***Variables*** attribute has been set to "P:Pad").

Example

```
import "e:\users\cre\adele\credocr3\Samples\GenerativeKnowledge\KweRectangle.CATPart";
ExpertRule isa CATPart
{
  myPart isa Part
    {
      RBase isa RuleBase
        {
          RSet isa RuleSet
            {
              R isa ExpertRule
                {
                  Variables="P:Pad";
                  RuleBody="if P\EndLimit\Length==10.0 P\EndLimit\Length=20.0";
                }
                C isa ExpertCheck
                {
                  Variables="P:Pad";
                  CheckBody="P\EndLimit\Length>10.0";
                }
            }
          }
         PartBody isa Body
             {
               Sketch isa Sketch.1 {}
               P0 isa Pad("Sketch")
                 {
                   EndLimit\Length=10.0mm;
                 }
             }
       }
   }
```



Formulas

Definition

A feature attribute can be defined by a formula. There are two means to specify a formula:

- either by using the ? operator. The parameter definitions are scanned from the statement block where the formula is defined to the outermost block of the script.
- or by specifying a relative path (**..\\..**\\). The parameter definition is searched for in the location defined by the **..**\\ operators.

Both methods are examplified below.

Example 1

```
// A 20.0 mm length cylinder is generated
Cylinder1 isa CATPart
  {
   Part isa Part
    {
      L = 21.0mm;
      PartBody isa Body
        {
          L = 12.0 mm ;
          Cyl1 isa Cylinder
             {
               L = 10.0mm;
                Radius=15.0mm;
                EndLimit\Length = 2*?L;
             }
        }
     }
  }
```

Example 2

```
// A 24.0 mm length cylinder is generated
Cylinder1 isa CATPart
  {
   Part isa Part
    {
      L = 21.0mm;
      PartBody isa Body
        {
          L = 12.0 mm ;
          Cyl1 isa Cylinder
             {
                Radius=15.0mm;
                EndLimit\Length = 2*?L;
             }
        }
     }
  }
```

Example 3

```
Cylinder1 isa CATPart
  {
   Part isa Part
    {
      L = 21.0mm;
      PartBody isa Body
        {
          L = 12.0 mm ;
          Cyl1 isa Cylinder
             {
                Radius=15.0mm;
                EndLimit\Length = 2*?L;
             }
          // Cyl2 is generated with a 48.0 mm length
          Cyl2 isa Cylinder
             {
                Radius=  5.0 mm ;
                EndLimit\Length = 2* ..\..\Cyl1\EndLimit\Length;
             }
        }
     }
  }
```

import Function

Definition

Specifies a document file (.CATPart or .CATProduct) containing definitions to be reused or redefined in the document to be generated. All the features and feature values in the imported file become available to the document to be generated.

Importing a document is:

- of interest whenever you want to retrieve a consistent set of definitions from an already existing document
- required whenever you need to create a feature by extruding a sketch (the script language does not allow you to program the creation of a sketch).

Syntax

`import(`*`FileName`*`) ;`

FileName is the name of the file which contains the document to be imported.
You must enclose the document name within quotation marks and end the import statement with a semicolon (;).
You can also use the ~ symbol to specify a relative path.

Example

```
import "e:\users\credocr3\Samples\GenerativeKnowledge\KweRectangle.CATPart";
```

isa Function

Definition

Creates a typed object or instantiates an object.

Syntax

`*ObjectName* **isa** *ObjectType*` or `*ObjectName* **isa** *InstanceName*`

where:

- *ObjectName* is *t*he name of the object to be created.
- *ObjectType* is the type of the object to be created.
- *InstanceName* is the name of the object to be instantiated.

Example

```
myDocument isa CATPart
  {
   myPart isa Part
      {
        PartBody isa Body
           {
            // Sketch.1 is the name of the object to be instantiated
            Sketch isa Sketch.1
```

? (Question Mark in Formulas)

Definition

In a formula, specifies that the parameter value to be applied is the first parameter value read when scanning the script from the statement block where the formula is defined to the outmost statement block.

Example

```
// A 24.0 mm length cylinder is generated
Cylinder1 isa CATPart
  {
   Part isa Part
    {
      L = 21.0mm;
      PartBody isa Body
        {
          L = 12.0 mm ;
          Cyl1 isa Cylinder
             {
                Radius=15.0mm;
                EndLimit\Length = 2*?L;
             }
        }
     }
  }
```

..\.. (Relative Path in Formulas)

Definition

Defines where the value of a parameter used as an argument in a formula is to be read. A single **..** exits the statement block where the formula is defined. The parameter value applied in the formula is then the one defined in the parent feature scope. Any additional **..** exits one additional statement block.

Example

```
 Cylinder1 isa CATPart
  {
   Part isa Part
    {
      L = 21.0mm;
      PartBody isa Body
        {
          L = 12.0 mm ;
          Cyl1 isa Cylinder
             {
                Radius=15.0mm;
                EndLimit\Length = 2*?L;
             }
          // Cyl2 is generated with a 48.0 mm length
          Cyl2 isa Cylinder
             {
                Radius=  5.0 mm ;
                EndLimit\Length = 2* ..\..\Cyl1\EndLimit\Length;
             }
        }
     }
  }
```

Coaxiality Transformation

Definition

Makes coaxial two cylinders. To specify a coaxiality within your script, you must have a part open, then:

1. create a coaxiality constraint
   ```
   constraints:
   coaxiality( ,  );
   ```
2. right-click inside the parentheses before the comma and select the 'Get Surface' function from the contextual menu.
3. in the geometry area, select one of the surfaces to be constrained
4. right-click inside the parentheses after the comma and select the 'Get Surface' function from the contextual menu.
5. in the geometry area, select the other face to be constrained.

Example

```
Cylinder1 isa CATPart
  {
    Part isa Part
       {
         PartBody isa Body
           {
             Cyl1 isa Cylinder
               {
                 Radius=15.0mm;
                 Length=30.0mm;
               }
             Cyl2 isa Cylinder
               {
                 Radius=15.0mm;
                 Length=30.0mm;
                 constraints:
                 coaxiality(...,...)
               }

           }
       }
}
```

Distance Constraint

Definition

Specifies a distance between two faces of two different features. To specify a distance within your script, you must have a part open, then:

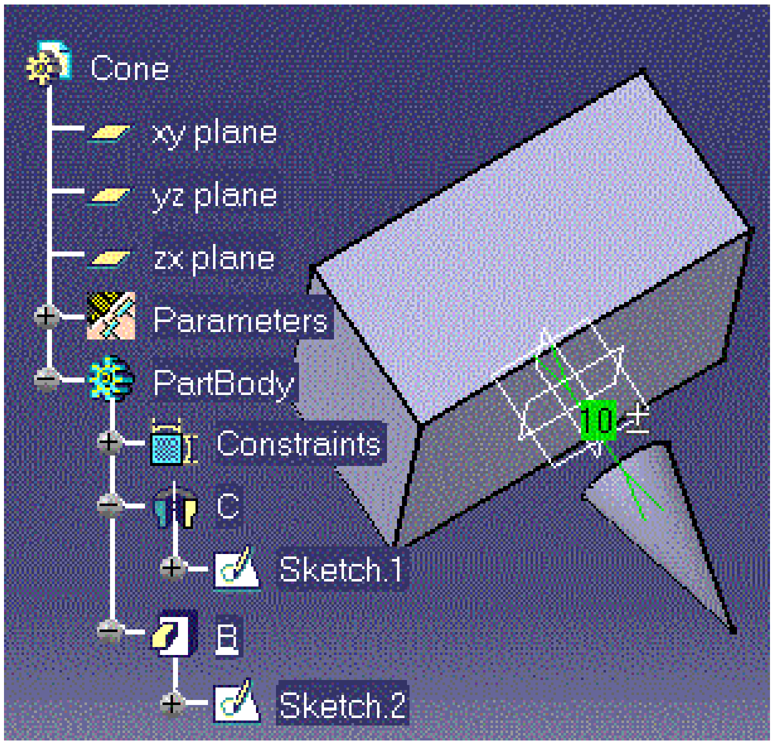
1. create a distance constraint
   ```
   constraints:
   distance(  ,  , );
   ```
2. right-click inside the parentheses before the comma and select the 'Get Surface' function from the contextual menu.
3. in the geometry area, select one of the faces to be constrained
4. right-click inside the parentheses after the first comma and select the 'Get Surface' function from the contextual menu.
5. in the geometry area, select the other face to be constrained
6. specify the distance value in the third parameter.

Syntax

```
constraints:
distance("Face:(Brp:(P;2);None:())","Face:(Brp:(Q;1);None:())",20.0);
```

Example

```
 Cone isa CATPart
   {
     Part isa Part
      {
        PartBody isa Body
        {
          C isa Cone
            {
            }
          B isa Box
            {
             constraints:
             distance(...,...,10.0);
            }
        }
     }
 }
```


Cone
xy plane
yz plane
zx plane
Parameters
PartBody
Constraints
C
Sketch.1
B
Sketch.2
10

Position Constraint

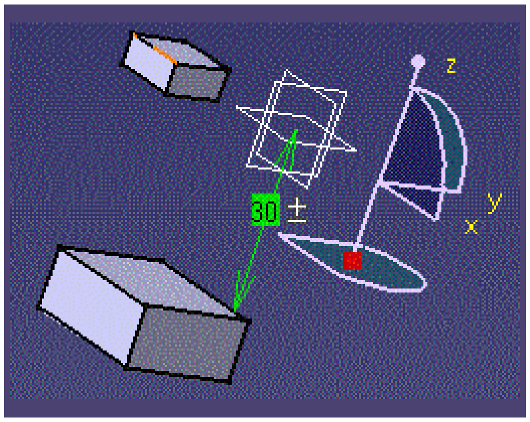
Definition

Places a feature with respect to another.

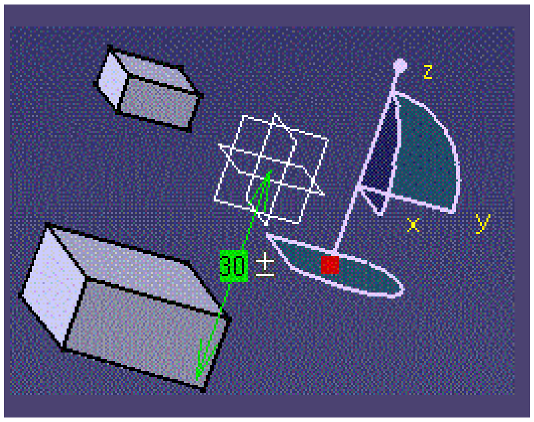
Syntax

```
constraints:
position infront | front  | rear  | inrear  |
         onleft  | left   | right | onright |
         below   | bottom | top   | above
         at distance.
```

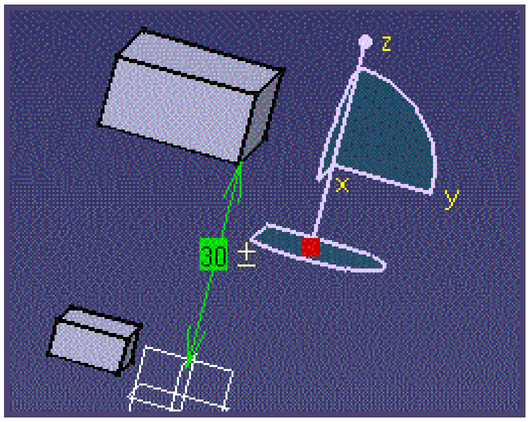
above



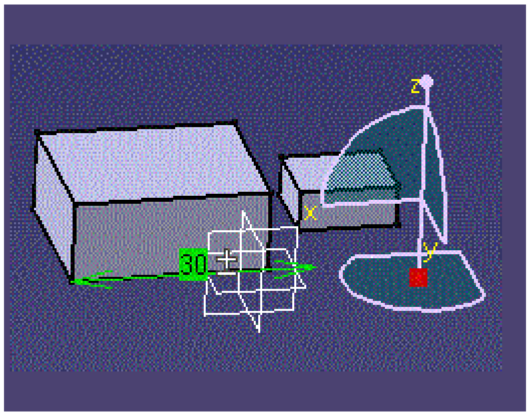
bottom



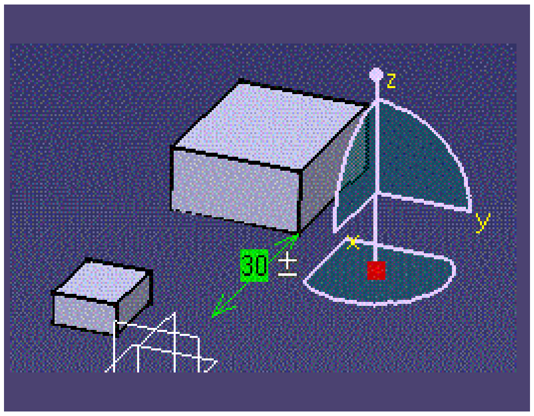
below



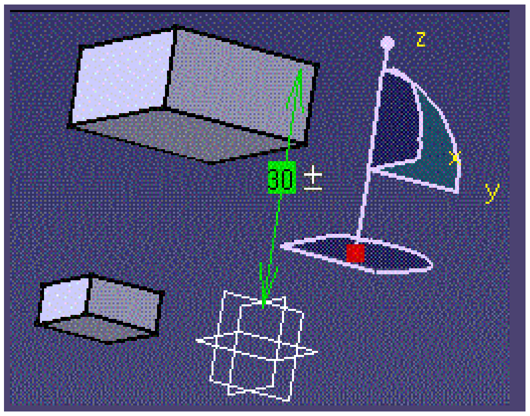
right



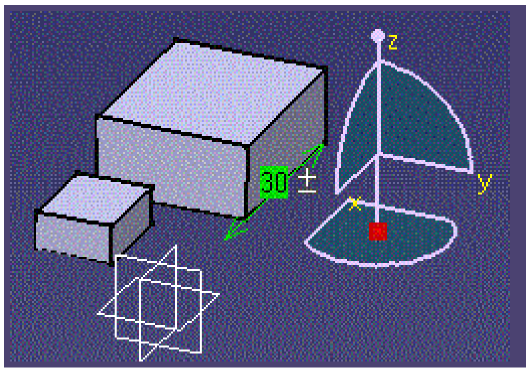
onright



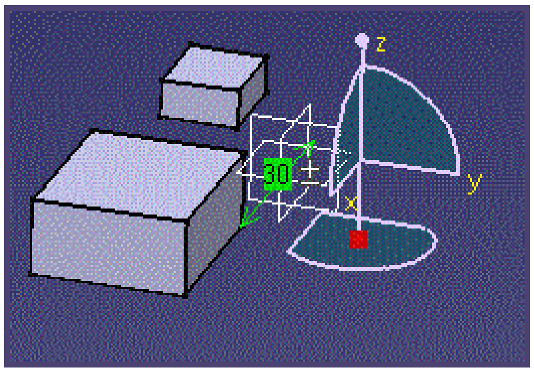
top



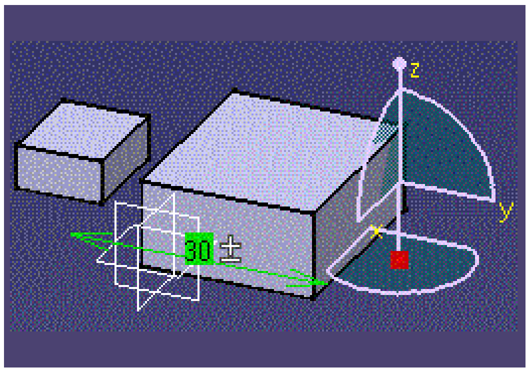
left



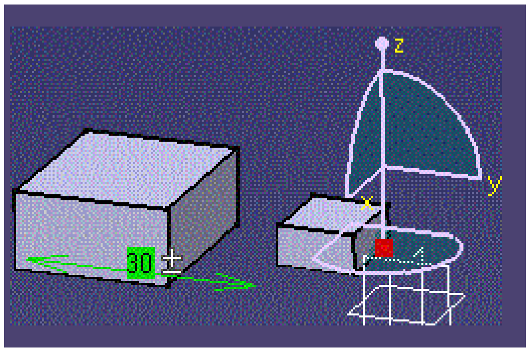
onleft

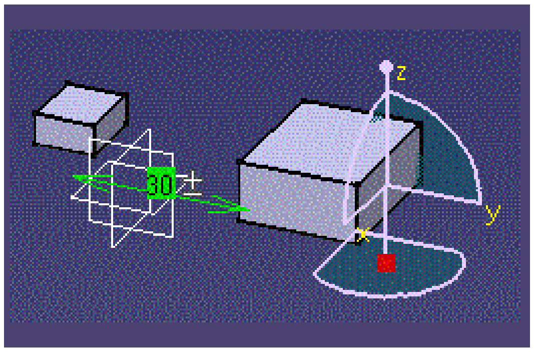


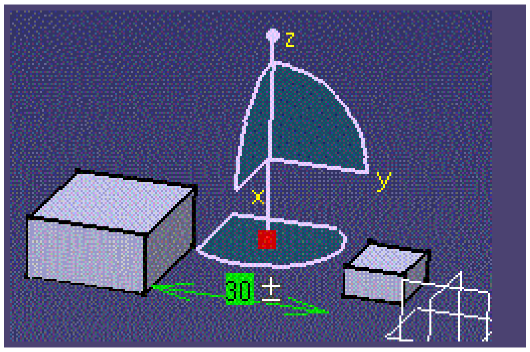
rear



front ***infront*** ***inrear***

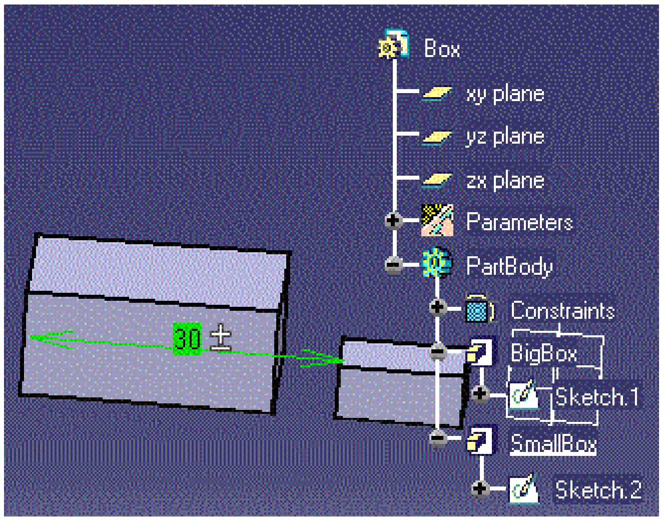






Example

```
Box isa CATPart
 {
 BoxPart isa Part
  {
   PartBody isa Body
    {
     BigBox isa Box
       {
        Width = 20.0 ;
        Height = 24.0 ;
        Length = 10.0 ;
       }
     SmallBox isa Box
  {
        Width = 10.0 ;
        Height = 12.0 ;
        Length = 5.0 ;
        constraints:
        position left ?BigBox at 30.0mm;
       }
     }
   }
 }
```



Rotate Constraint

Definition

Rotates a feature by a given angle around the x, y and z axes.

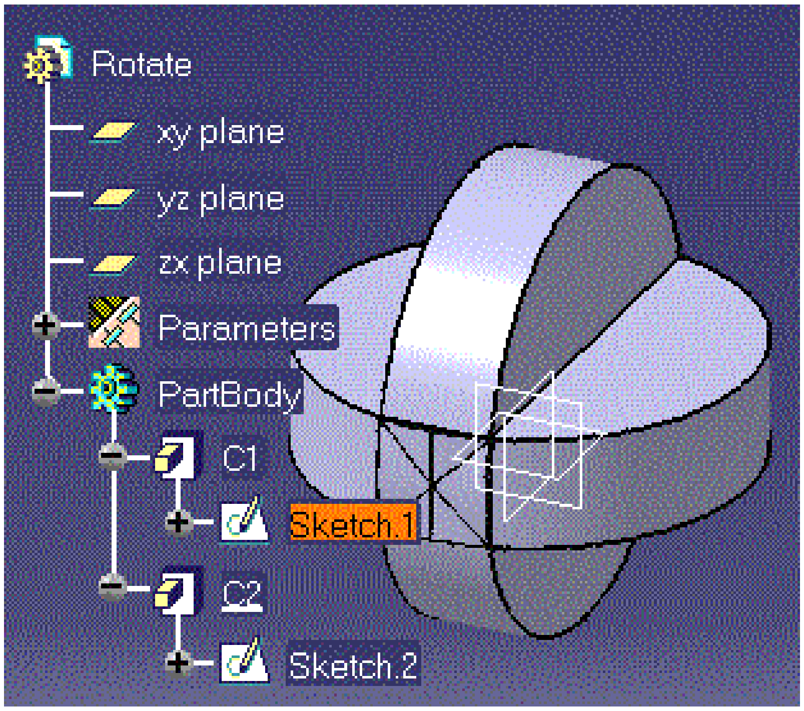
```
constraints:
rotate angle1, angle2, angle3  ;
```

where:

angle1, *angle2* and *angle3* represent the angles of rotation (expressed in degrees) around the x, y and z axes. The angles can be defined by a formula.

Example

```
Rotate isa CATPart
{
  Part isa Part
   {
     PartBody isa Body
       {
         C1 isa Cylinder
          {
            Radius=40.0mm;
            Length=100.0mm;
          }
         C2  isa Cylinder
          {
            Radius=40.0mm;
            Length=100.0mm;
            constraints:
            rotate 0.0,90.0,90.0;
          }
       }
    }
}
```



Translate Constraint

Definition

Moves a feature by a given amount along the x, y and z axes.

```
    constraints:
    translate x1, y1, z1  ;
```

where the amounts specified are given in meters.

Example

```
 Translate isa CATPart
 {
     Part isa Part
     {
        PartBody isa Body
        {
             C1 isa Cylinder
             {
                 Radius=40.0mm;
                 Length=100.0mm;
             }
           C2  isa Cylinder
             {
                 Radius=40.0mm;
                 Length=100.0mm;
                 constraints:
                 translate 100.0,0.0,0.0;
             }
        }
     }
 }
```