Catia V5 knowledgeware awareness session

Francois Trudel
DS Service, Lean Engineering
KBE Development Application Consultant
Francois.trudel@3ds.com
3/3/2015
CATIA Knowledgeware Product Portfolio

Knowledge-based Engineering (KBE) tools

CATIA Knowledge Advisor (KWA)
CATIA Product Knowledge Template (PKT/KT1)
CATIA Product Engineering Optimizer (PEO)
CATIA Knowledge Expert (KWE/KE1)
CATIA Business Process Knowledge Template (BKT/BK2)

PLM Knowledge Infrastructure (COM)

Enforce Standards Usage
Optimize Product Performance
Validate Best Practices
Automate Engineering Process
Workbench important pour automatisation

- **Part Design**

- **Assembly Design**

- **Generative Shape Design**
Demo: How to design aircraft wing ribs using knowledgeware?

Key objectives

1. Eliminate repetitive task
2. Use company best practices
3. Encapsulate knowledge
4. Speed of design / modification
5. Standardize the design process
6. Lifecycle management
Manual rib design

1- Get the wing surface and define 2 planes
2- Define a plane for the rib position, extract the curve: wing curve
3- Define a Point at the CG of the wing curve
Manual rib design

4- Create a circle, use the CG as center
5- Adjust the radius to have 30mm min distance with the wing curve
6- Fill the wing curve
7- Remove the circle shape from the fill
Knowledge Advisor:

- Parametric design
- User parameters / Intrinsic parameters
- Relations (Formula, Rule, Reaction, Action, Check...)
- Create virtual variable and geometry
- Link the virtual geometry/variable to tree parameter/datum
Product knowledge template:

- User feature: Create and instantiate custom feature.
- Power copy: Create and instantiate a package of existing feature.
- Document template: Create and instantiate Part/Product template
- Knowledge pattern: Create and Instantiate datum and feature in the Catia Tree
Business Process Knowledge Template:

- Build automated process
- Build process using easy to understand Behaviors (building block)
- Reuse existing process
Product Engineering optimizer:

- Use algorithm to increase product performance
Knowledge Expert

➢ Create generic rules to embed business best practices
Catia V5 Options (1/5)
Catia V5 Options (2/5)
Catia V5 Options (3/5)
Catia V5 Options (4/5)
Catia V5 Options (5/5)
Exercise

In a new Part, Copy/Paste as Result Wing_Shape curve from Wing_R21.CATPart
Copy/Paste as Result the plane linked to the curve.

Create a Length parameter named MinBorder.
Create an Area parameter named Wing_Area.
Create a Length parameter named Wing_Shape_Length.
Create a Surface datum named Rib_Surface.

Save part
Dictionary: EKL

The dictionary regroup lots of function and keywords of the EKL (Engineering Knowledge Language). This language is used throughout the knowledgeware workbenches.

Dictionary: Parameters

Access to Intrinsic parameters and User parameters.

Filtered by type

Also possible to access the parameters by selecting elements from the Catia Tree
Parametric Sketch

The Engineering Knowledge language has limited availability is a sketch. Constraints can be assign to parameters using Relations.

The user have no power on the orientation of a distance constraint in a sketch. Use GSD as much as possible
Dictionary: Circle Constructors

Members of Circle Constructors

- circleCtrRadius (center: Point, support: Surface, radius: Length, limits: Integer, start: Angle, end: Angle): Circle
- circleCtrPt (center: Point, point: Point, support: Surface, limits: Integer, start: Angle, end: Angle): Circle
- circle2PntRadius (point1: Point, point2: Point, support: Surface, radius: Length, orientation: Boolean, limits: Integer): Circle
- circle3Pnt (point1: Point, point2: Point, point3: Point, limits: Integer): Circle
- circleBtntRadius (curve1: Curve, curve2: Curve, support: Surface, radius: Length, orientation1: Boolean, orientation2: Boolean, limits: Integer): Circle
- circleBtntRadius (curve1: Curve, point: Point, support: Surface, radius: Length, orientation1: Boolean, orientation2: Boolean, limits: Integer): Circle
- circleBtntPoint (curve1: Curve, curve2: Curve, point: Point, support: Surface, orientation1: Boolean, orientation2: Boolean, limits: Integer): Circle
- circleCrt (point: Point, curve: Curve, curve orientation: Boolean, tangent orientation: Boolean, support: Surface): Circle
- circleCrtGt (curve1: Curve, curve2: Curve, curve orientation: Boolean, curve2 tgt orientation: Boolean, support: Surface, radius: Length, curve1 orientation: Boolean, curve2 tgt orientation: Boolean): Circle
Dictionary: Circle Constructors

Name of the function

Input #1, a Point
For the center of the circle

Input #2, a Surface
For the circle support

Input #3, a length
For the circle radius

Input #4, an Integer
For the limits

Input #5, an Angle
For the end angle

Output of the function:
Object of type Circle

Same as the GSD Circle definition
Dictionary: Mesure

Members of Measures

distance (Body, Body): Length
minimumCurvatureRadius (Curve): Length
nbDomains (Body): Integer
length (Curve, ...): Length
length (Curve, Point, Boolean): Length
length (Curve, Point, Point): Length
area (Surface, ...): Area
area (Curve, ...): Area
perimeter (Surface, ...): Length
Point->coord (x: out Length, y: out Length, z: out Length): VoidType
Point->coord (rank: Integer): Length
Body->centerOfGravity (x: out Length, y: out Length, z: out Length): VoidType
volume (closed_surface: Surface, ...): Volume
volume (Volume geometry, ...): Volume
angle (center: Point, pt1: Point, pt2: Point): Angle
angle (Direction, Direction): Angle
angle (Line, Line): Angle
angle (Plane, Plane): Angle
angleOriented (Direction, Direction, Direction): Angle
angleOriented (Line, Line, Direction): Angle
angleOriented (Plane, Plane, Direction): Angle
curvature (curv: Curve, pt: Point): Real
Dictionary: Point Constructors

Members of Point Constructors

<table>
<thead>
<tr>
<th>Method Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>point(x: Length, y: Length, z: Length): Point</td>
</tr>
<tr>
<td>Point Between</td>
<td>pointbetween(pt1: Point, pt2: Point, ratio: Real, orientation: Boolean): Point</td>
</tr>
<tr>
<td>Point on Curve</td>
<td>pointoncurve(crv: Curve, pt: Point, distance: Length, orientation: Boolean): Point</td>
</tr>
<tr>
<td>Point on Curve Ratio</td>
<td>pointoncurveRatio(crv: Curve, pt: Point, ratio: Real, orientation: Boolean): Point</td>
</tr>
<tr>
<td>Point on Plane</td>
<td>pointonplane(plt: Plane, pt: Point, dx: Length, dy: Length): Point</td>
</tr>
<tr>
<td>Point on Surface</td>
<td>pointonsurface(sur: Surface, pt: Point, dir: Direction, dist: Length): Point</td>
</tr>
<tr>
<td>Center (Circle)</td>
<td>center(Circle): Point</td>
</tr>
<tr>
<td>Point on Tangent</td>
<td>pointtangent(crv: Curve, Direction): Point</td>
</tr>
<tr>
<td>Extremum</td>
<td>extremum(crv: Curve, Direction, Boolean, Direction, Boolean): Point</td>
</tr>
<tr>
<td>Extremum (Surface)</td>
<td>extremum(sur: Surface, Direction, Boolean, Direction, Boolean): Point</td>
</tr>
<tr>
<td>Extremum (Solid)</td>
<td>extremum(solid: Solid, Direction, Boolean, Direction, Boolean): Point</td>
</tr>
<tr>
<td>Center of Gravity</td>
<td>centerofgravity(body, ...): Point</td>
</tr>
<tr>
<td>Curvature Center</td>
<td>curvaturecenter(crv: Curve, pt: Point): Point</td>
</tr>
</tbody>
</table>

---

The 3DEXPERIENCE Company
Exercise

Create a formula on parameter Wing_Shape_Length to find the length of Wing_Shape curve.

Create a Point datum. Create a formula to set this point as Center of gravity of Wing_Shape curve. Rename CG.

Save part
Dictionary: Line Constructors

Members of Line Constructors

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inertiaAxis</td>
<td>(rank: Integer, Body, ...) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>line</td>
<td>(Point, Point) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>line</td>
<td>(pt: Point, dir: Direction, start: Length, end: Length, orientation: Boolean) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>linetangent</td>
<td>(crv: Curve, pt: Point, start: Length, end: Length, orientation: Boolean) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>linenormal</td>
<td>(sur: Surface, pt: Point, start: Length, end: Length, orientation: Boolean) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>mainnormal</td>
<td>(crv: Curve, pt: Point) -&gt; Line</td>
<td></td>
</tr>
<tr>
<td>binormal</td>
<td>(crv: Curve, pt: Point) -&gt; Line</td>
<td></td>
</tr>
</tbody>
</table>
Dictionary: Operation constructors

Members of Operations Constructors

assemble (Curve, ...): Curve
assemble (Surface, ...): Surface
split (tosplit: Curve, splitting: Wireframe, orientation: Boolean): Curve
split (tosplit: Surface, splitting: Surface, orientation: Boolean): Surface
split (tosplit: Surface, splitting: Curve, orientation: Boolean): Surface
trim (crv1: Curve, orientationCrv1: Boolean, crv2: Curve, orientationCrv2: Boolean): Curve
near (crv: Curve, near: Wireframe): Curve
near (sur: Surface, near: Wireframe): Surface
near (pt: Point, near: Wireframe): Point
boundary (element: Surface): Curve
translate (ttransform: Wireframe, dir: Direction, distance: Length): Wireframe
symmetry (ttransform: Wireframe, invariant: Wireframe): Wireframe
invert (CurveOrSurfaceToInvert: Wireframe): Wireframe
**Dictionary: Plane Constructors**

<table>
<thead>
<tr>
<th>Members of Plane Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>plane (Point, Point, Point): Plane</td>
</tr>
<tr>
<td>plane (a: Real, b: Real, c: Real, d: Length): Plane</td>
</tr>
<tr>
<td>plane (Line, Line): Plane</td>
</tr>
<tr>
<td>plane (Point, Line): Plane</td>
</tr>
<tr>
<td>plane (Curve): Plane</td>
</tr>
<tr>
<td>planetangent (Surface, Point): Plane</td>
</tr>
<tr>
<td>planenormal (Curve, Point): Plane</td>
</tr>
<tr>
<td>planeoffset (Plane, offset: Length, orientation: Boolean): Plane</td>
</tr>
<tr>
<td>planeoffset (Plane, Point): Plane</td>
</tr>
<tr>
<td>planemean (Point, ...): Plane</td>
</tr>
</tbody>
</table>

![Plane Definition Window](image1.png)
## Dictionary: Wireframe Constructors

### Members of Wireframe Constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spline (Point, ...): Curve</td>
<td></td>
</tr>
<tr>
<td>intersect (Curve, Curve): Point</td>
<td></td>
</tr>
<tr>
<td>intersect (Curve, Surface): Point</td>
<td></td>
</tr>
<tr>
<td>intersect (Surface, Surface): Curve</td>
<td></td>
</tr>
<tr>
<td>curveparallel (crv: Curve, sur: Surface, offset: Length, OptionalRoundMode: Boolean): Curve</td>
<td></td>
</tr>
<tr>
<td>project (toproject: Point, CurveOrSurfaceSupport: Wireframe, OptionalDir: Direction): Point</td>
<td></td>
</tr>
<tr>
<td>project (toproject: Curve, support: Surface, OptionalDir: Direction): Curve</td>
<td></td>
</tr>
<tr>
<td>extrude (Point, Direction, length1: Length, length2: Length, orientation: Boolean): Line</td>
<td></td>
</tr>
<tr>
<td>revolve (Point, axis: Line, angle1: Angle, angle2: Angle): Circle</td>
<td></td>
</tr>
</tbody>
</table>
Dictionary & Language browser limitation

No Feature Constructor? No need when you can instantiate UDF and Power Copy
Dictionary: Keywords -> Let & Set

<table>
<thead>
<tr>
<th>Members of Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>else if</td>
</tr>
<tr>
<td>let</td>
</tr>
<tr>
<td>set</td>
</tr>
<tr>
<td>for</td>
</tr>
<tr>
<td>inside</td>
</tr>
<tr>
<td>while</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

Let

Example: Let aCurve (Curve)
Create a memory allocation for a virtual variable or object
Assign a type to the variable

Set

Example: Set aCurve = PartBody\Curve.1
Example: Set aPoint = Point(0mm,0mm,0mm)
Assign an existing geometry to a variable of the same type
Or create a virtual geometry of the same type
Dictionary: Keywords -> IF

Ask a question and do something based on the answer

Statement

If <Variable or Object> <operator> <variable or object>

If statement = true then
<do this>

If <Variable or Object> <operator> <variable or object>

{  
If statement = true then
<do this>
<and this>
}

Single line, no need {}

More than one line, need {}
Dictionary: Keywords -> IF

If <Variable or Object> <operator> <variable or object>
{
    <do this>
    <and this>
}

If statement = false than Skip the IF

<NEXT LINE OF CODE>
Dictionary: Keywords -> IF

Examples:

if aLength == 10mm
    aPoint.Show = true

if aLength <> 100mm
{
    bLength = 10mm
    Message("aLength : ", aLength)
}

Members of Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Is equal to</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Is Not equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is more than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is more or equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is less or equal to</td>
</tr>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>-</td>
<td>Minus</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>(</td>
<td>Open parenthesis</td>
</tr>
<tr>
<td>)</td>
<td>Close parenthesis</td>
</tr>
<tr>
<td>and</td>
<td>Combine statement, inclusive</td>
</tr>
<tr>
<td>or</td>
<td>Combine statement, exclusive</td>
</tr>
</tbody>
</table>
Dictionary: Keywords -> ELSE

Statement

If <Variable or Object> <operator> <variable or object>
{
    <do this>
    <and this>
}
Else
{
    <do this>
    <and this>
}
Dictionary: Keywords -> ELSE IF

Statement #1

If <Variable or Object> <operator> <variable or object>
{
    <do this>
}

Statement #2

Else if <Variable or Object> <operator> <variable or object>
{
    <do this>
}
Exercise

Continue with the Rib construction using Rule and virtual geometries

Output the end result using the datum Rib_Surface

Save part
Exercise solution 1/2

In the rule:

Let oWingShape (Curve)
Let oCircle (Circle)
Let oLineUp (Line)
Let oLineDown (Line)
Let oSurfaceOutput (Surface)
Let oSurfaceOutputTrue (Surface)
Let oSurfaceOutputFalse (Surface)
Let oPointUp (Point)
Let oPointDown (Point)
Let oPointCG (Point)
Let lUp (Length)
Let lDown (Length)

Set oWingShape = Input\ .Wing_ Shape
Set oPointCG = centerofgravity(oWingShape )

Set oLineUp = line(oPointCG , direction(0in,0in,1in),0in,100in,true)
Set oLineDown = line(oPointCG , direction(0in,0in,-1in),0in,100in,true)

Set oPointUp= intersect(oLineUp , oWingShape )
Set oPointDown= intersect(oLineDown , oWingShape )
Exercise solution 2/2

```plaintext
Set IUp = distance(oPointUp, oPointCG)
Set IDown = distance(oPointDown, oPointCG)

if IUp >= IDown
{
    Set oCircle = circleCtrRadius(oPointCG, 'yz plane', IDown - MinBorder, 1, 0, 2*PI)
}
else
{
    Set oCircle = circleCtrRadius(oPointCG, 'yz plane', IUp - MinBorder, 1, 0, 2*PI)
}

Set oSurfaceOutput = fill(oWingShape)
Set oSurfaceOutputFalse = split(oSurfaceOutput, oCircle, false)
Set oSurfaceOutputTrue = split(oSurfaceOutput, oCircle, true)

if area(oSurfaceOutputFalse) >= area(oSurfaceOutputTrue)
{
    Output\Surface.1 = oSurfaceOutputFalse
}
else
{
    Output\Surface.1 = oSurfaceOutputTrue
}
```

Compute distance, store data in variables

Basic logics

Language browser

To access functions

Non ambiguous design enabled