TASKS

Solidworks 参数化模型 SaaS方案

- [ ] 编写一个探测模型参数的AppBundle,供前端动态生成参数页面
  - 在一个标准的 Inventor Add-in 类库项目中（确保引用了 Autodesk.Inventor.Interop.dll 和 Newtonsoft.Json）

核心逻辑

```
using System;
using System.Collections.Generic;
using System.IO;
using Autodesk.Inventor;
using Newtonsoft.Json;

namespace ParamDiscoveryPlugin
{
public class SampleAutomation
{
private InventorServer _inventorServer;

public SampleAutomation(InventorServer inventorServer)
{
_inventorServer = inventorServer;
}

// APS 引擎实际调用的主方法
public void Run(Document doc)
{
try
{
// 1. 获取模型的所有参数
Parameters paramsCollection = null;
if (doc.DocumentType == DocumentTypeEnum.kPartDocumentObject)
paramsCollection = ((PartDocument)doc).ComponentDefinition.Parameters;
else if (doc.DocumentType == DocumentTypeEnum.kAssemblyDocumentObject)
paramsCollection = ((AssemblyDocument)doc).ComponentDefinition.Parameters;

if (paramsCollection == null) return;

// 2. 构造元数据列表
var exportList = new List<object>();

// 我们只导出用户定义的参数（UserParameters），避开系统生成的 d1, d2...
foreach (UserParameter p in paramsCollection.UserParameters)
{
// 高级过滤：可以只导出在 Inventor 中勾选了 "Key" 的参数
// if (!p.IsKey) continue;

exportList.Add(new
{
Name = p.Name,
Value = p.Expression, // 带单位的原始值，如 "50 in"
PrecisionValue = p.Value, // 内部标准单位值 (通常是 cm)
Units = p.Units,
Comment = p.Comment
});
}

// 3. 序列化为 JSON 并写入文件
// 注意：由于是 Discovery 任务，我们固定保存为 parameterMetadata.json
string jsonOutput = JsonConvert.SerializeObject(exportList, Formatting.Indented);
File.WriteAllText("parameterMetadata.json", jsonOutput);

Console.WriteLine("Successfully exported " + exportList.Count + " parameters.");
}
catch (Exception ex)
{
```

```
Console.WriteLine("Error: " + ex.Message);
}
}
}
}
```

- [ ] 探测参数Bundle对应的Activity设置注意

对应的 Activity 配置建议
为了让这个 Bundle 正常工作，您在定义 Activity 时需要这样设置参数：

inputPart: (Verb: GET) 输入的 .ipt 模型。
outputPart: (Verb: PUT, Local Name: parameterMetadata.json)
这就是您的探测结果。Web 前端之后会去下载这个文件来生成 UI

- [ ] 利用 APS 的 Webhooks 功能：模型上传后自动探测参数

- [ ] 参数探测json文件与模型位置规划

一个成熟的自动化系统通常会结合这两种路径：

Bucket 划分：

SourceBucket: 存放原始模型（按 URN 分类）。
MetadataBucket: 存放
metadata.json
（按 URN 分类，长期保存）。
WorkingBucket: 专门用于 WorkItem 的输入输出（按 JobID 分类，设置生命周期规则，比如 7 天后自动删除）。
后端逻辑：

当 Web 前端需要某个模型的 UI 时：
查询本地数据库或 OSS：MetadataBucket/model_A/metadata.json。
当用户点击“执行修改”时：
后端动态生成一个临时文件名：WorkingBucket/job_XYZ/outputFile.ipt。
把这个带签名的路径喂给 WorkItem。

- [ ] 使用APS web化驱动Solidworks的模型参数修改

- SW设计 -> 导入Inventor -> 定义驱动参数(Move等) -> 上传APS -> Web动态调参
- Inventor的 Direct Edit 功能完成参数适配工作（SolidWorks 的内部参数无法直接在云端被“驱动”）
- 当在 Inventor 中通过 AnyCAD 打开一个 SW 文件时，它看到的是一个“参考几何体”(Reference Body)。SW 内部的 D1@Sketch1 这类参数是存储在 SW 专有的特征树里的，Inventor 能读出它的样子，但拿不到它的“控制器”。
- APS 的局限：APS 的计算引擎（Design Automation for Inventor）只能识别 Inventor 自身的 Paramters 表
- 这个流程在工业软件界被称为“Multi-CAD 混合工作流”，是目前处理跨厂商自动化最成熟的方案之一

Direct Edit (直接编辑) 的底层原理是什么

如果说传统建模是“看菜谱做菜”(改变配方，重新下锅)，那么 Direct Edit 就是“直接捏橡皮泥”。

底层科学：B-Rep 操纵
CAD 模型在底层是用 B-Rep (Boundary Representation，边界表示法) 描述的，即由一系列相互连接的面 (Face)、边 (Edge) 和 点 (Vertex) 组成的闭合体。

传统方式 (Feature-based)：你要改高度，系统会去修改拉伸草图的数值，然后重新计算整个特征树。如果中间断了，模型就报错。
Direct Edit 方式：
识别目标：当你选中一个面准备移动时，算法会识别出这个面的几何定义（比如是一个平面或圆柱面)。
邻域重算：当你拖动面时，算法会实时计算与该面相连的其他面。它会自动尝试保持切向 (Tangent)、垂直 (Perpendicular) 或平行 (Parallel) 等几何约束。
拓扑修补：面移动后，边缘会变长或缩短，算法会自动“缝合”这些空隙，确保模型依然是封死的实心体

Inventor AnyCAD打开SW模型验证，防止Web驱动产生废品

. 如何确保不出“废品”？（安全核对表）
为了保证 Web 驱动生成的模型可以直接送上生产线，建议采取以下三道防线：

第一道：导入诊断 (Import Diagnostics)
在 Inventor 中打开 SW 文件后，第一时间运行插件的“输入诊断”。它会扫描是否有无效面。只有诊断为“Green（绿色)”无误的模型，才适合作为 APS 的模板。
第二道：数值驱动 (Numerical Control)
在您的 Move 操作中，千万不要手动拖拽，而是建立参数（比如命名为 side_offset)。
在 APS 的 Web 端，用户输入 25.5，后端就传入 25.5。这样能确保几何层面的移动步长与工程需求完全一致。
第三道：结果闭环验证 (Result Validation)
体积/质量对比：您可以写一小段代码，让 APS 在完成修改后，实时返回模型的重量（Mass)。如果在 Web 端改了一个大参数，重量却没变，说明模型报错了。
关键尺寸检测：在导出生产文件（如 STEP/DXF）前，可以在云端用代码测量一下关键点之间的距离，作为最后一道质检网。

万能参数修改生产级别保证

针对“工业生产”的避坑建议：
如果您要将其用于正式生产，我建议在目前这个“万能包”的基础上做两点增强：

A. 预定义约束（在模型端做)： 在上传 .ipt 前，利用 Inventor 的 参数限制 功能。例如，限制 width 只能在 10
到 50 之间。这样即使用户在 Web 端乱填，模型也不会崩溃。
B. 结果校验（在 AppBundle 端做)： 如果您有开发能力，可以在 SampleAutomation.cs 中加入一段代码，在保存前检查 doc.RebuildStatus。如果模型重算状态是“失败 (kDirty)”，直接报错不生成 output，防止将“废品”发给客户。

APS云点费用

跑一次 WorkItem 要花多少钱?
以您刚才成功的任务为例，我们来算一笔账：

Inventor 引擎费率：6 个云点 / 每处理小时。
您的任务耗时：从日志看，处理阶段大约耗时 6 秒（6/3600 小时）。
计算公式：6 秒 / 3600 秒 * 6 点 * 22 元 ≈ 0.22 元。
结论：您跑一次这种参数化修改任务，成本大约只需要 2 角 2 分钱。这比雇一个工程师手动打开软件去改参数要便宜得多，而且速度是秒级的

在 APS 的计费体系中，确实是遵循**“重计算、轻存储”**的原则。

针对您列出的整套流程，费用的分布如下：

1. 免费（或暂不计费）的部分：
上传 .ipt 模型 (OSS)：目前 APS 并不根据存储文件的容量（GB）或上传流量收费。您可以自由上传、更新模型。
创建 AppBundle：这是免费的。您可以上传多个版本的代码包。
创建 Activity：这是免费的。这只是定义一个任务模板。
获取签名链接 (Signed URL)：这也是免费的 API 调用。

2. 主要收费的部分：
提交 WorkItem (核心收费点)：
这是您唯一需要重点关注的支出。
它是按实际运行时间计费的（按秒计费，如 Inventor 引擎 6 个云点/小时）。

3. 一个容易忽略的额外收费点：
Model Derivative (模型转译)：
如果您希望在网页前端（Viewer）直接看到修改后的 3D 模型，就需要对生成出的新模型调用“转译 (Translate)”接口。
费用标准：
复杂模型 (如 Revit/Navisworks)：通常是 1.5 个点/个模型。
普通模型 (您的 Inventor .ipt)：通常是 0.2 个点/个模型（折合人民币约 4-5 元）。

注意：如果您只是下载模型回本地 SolidWorks，不在线查看，那么这笔费用也是 0

- [ ] 万能型 iLogic 规则示例：UniversalParamUpdater（APS 工业级应用中最流行的核心架构：AppBundle (壳) + iLogic (瓤) + JSON (驱动) ）

```
' iLogic Rule: UniversalParamUpdater
' 功能：自动读取目录下的 params.json 并更新模型同名参数
' 适用场景：APS Design Automation 或本地自动化测试

AddReference "Newtonsoft.Json" ' Inventor 默认包含此库
Imports Newtonsoft.Json.Linq
Imports System.IO

Sub Main()
' 1. 定位 params.json (在 APS 环境中，它与模型在同一个根目录)
Dim jsonPath As String = "params.json"

If Not File.Exists(jsonPath) Then
Logger.Info("未找到 params.json，跳过参数更新。当前目录：" & Directory.GetCurrentDirectory())
Return
End If

' 2. 读取并解析 JSON
Dim jsonContent As String = File.ReadAllText(jsonPath)
Dim data As JObject = Nothing

Try
data = JObject.Parse(jsonContent)
Catch ex As Exception
Logger.Error("JSON 格式错误: " & ex.Message)
Return
End Try

' 3. 遍历 JSON 中的所有键值对并映射到参数
For Each prop In data.Properties()
Dim pName As String = prop.Name
Dim pValue As String = prop.Value.ToString()

Try
' 尝试寻找同名参数并赋值
' iLogic 的 Parameter() 函数支持自动单位类型转换
Parameter(pName) = pValue
Logger.Info("成功更新参数: " & pName & " = " & pValue)
Catch ex As Exception
' 如果模型里没定义这个参数，或者数值不合法，会跳过并记录警告
Logger.Warn("参数匹配失败或数值无效: " & pName & " (错误详情: " & ex.Message & ")")
End Try
Next

' 4. 强制执行模型重算
RuleParametersOutput() ' 确保参数推送到模型
iLogicVb.UpdateWhenDone = True
InventorVb.DocumentUpdate()

Logger.Info("万能参数更新任务完成！")
End Sub
```

- [ ] DriveWorks

  如何实现：
  设计端：设计员在 SolidWorks 中使用 DriveWorks 插件定义参数规则（类似 iLogic）。
  服务器端：使用 DriveWorks Pro 搭建一台服务器（可以部署在 Azure/AWS 虚拟机上），服务器上安装有 SolidWorks 实例。
  网页端：利用 DriveWorks Live 自动生成 Web 界面，或通过其 API 将表单嵌入到你自己的官网。
  流程：用户在网页提交参数 -> DriveWorks 调度服务器上的 SolidWorks 自动打开、改数、另存、导出 -> 网页提供下载链接。
  优点：不需要编写复杂的 CAD API 代码，规则引擎极其强大，支持自动生成报价单和 BOM。
  缺点：需要购买 DriveWorks 授权，且需要自己维护（或在云端托管）一个安装了 SolidWorks 的 Windows 环境

- [ ] 3DEXPERIENCE Platform (官方云平台)

  使用 3DX 平台上的 xDesign 或 xFrame 等原生云建模应用。
  通过 Enterprise Knowledge Language (EKL) 编写自动化规则。
  利用其 Web Services API 进行数据集成。
  痛点：它与传统的 SolidWorks 桌面版（SLDPRT/SLDASM）是有代差的。如果你想让云端引擎像 APS 操纵 Inventor 那样操纵原生的 SolidWorks 文件，目前的开放程度不如 Autodesk

- [ ] Tacton Design Automation

  专注于复杂的规则配置，在大型跨国企业的定制化流程中非常常见，同样支持从 Web 界面驱动 SolidWorks 建模

- [ ] Open Cascade

  其实 APS 的那种“无头计算”思想是非常先进的。如果在 SolidWorks 侧找不到如此纯粹的 API 体验，也有很多企业会选择将 SolidWorks 模型转成通用格式后，在后端利用其他的几何引擎（甚至是开源引擎如 Open Cascade）来做此类简单的参数化变形