Biblioteca Tia Portal Openness

Índice

Utilização da Biblioteca

Para criar uma aplicação/programa com a biblioteca, iniciar a IDE e a aplicação criada com permissões de administrador. Sem permissões não é possível executar certas funções.

Pasta de Ficheiros

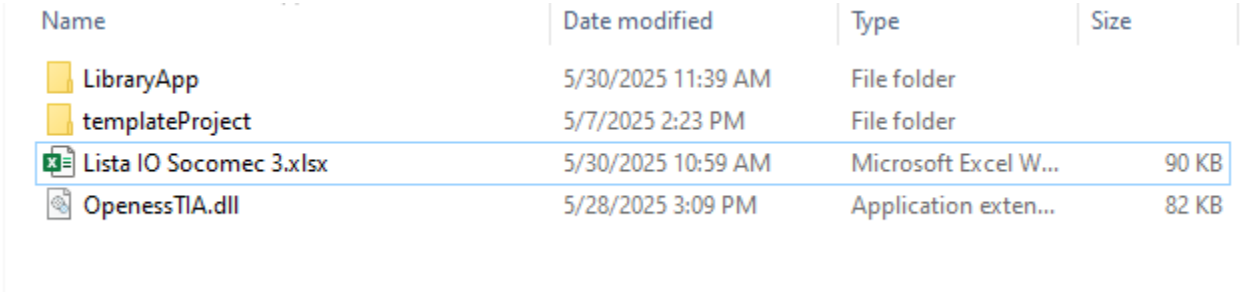
Deve ser criada uma pasta com ficheiros iniciais com os seguintes ficheiros.

“templateProject” que contém o projeto Template do TIA V18.

“LibraryApp” que contém a global Library a ser importada.

“OpenessTIA.dll” sendo esta a biblioteca compilada a utilizar

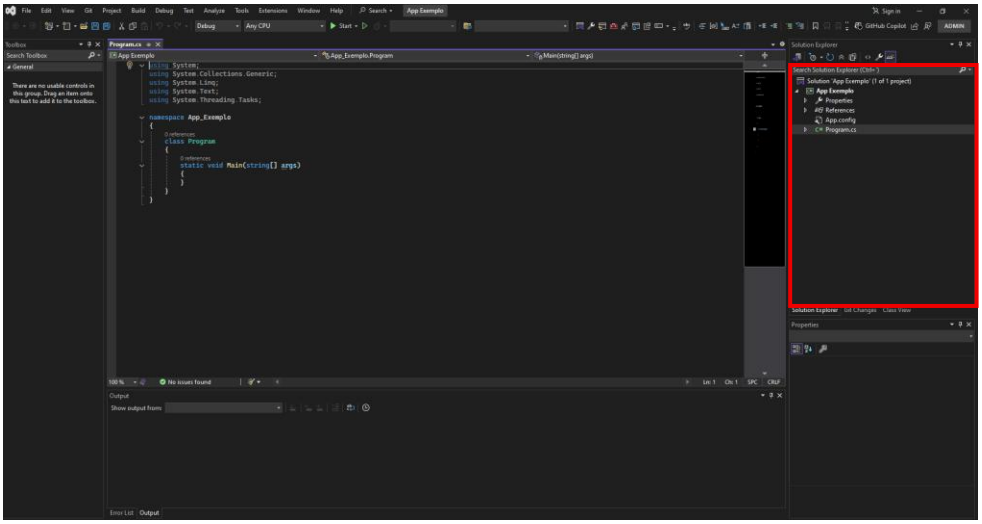
Um ficheiro em Excel para ser lido pela aplicação.

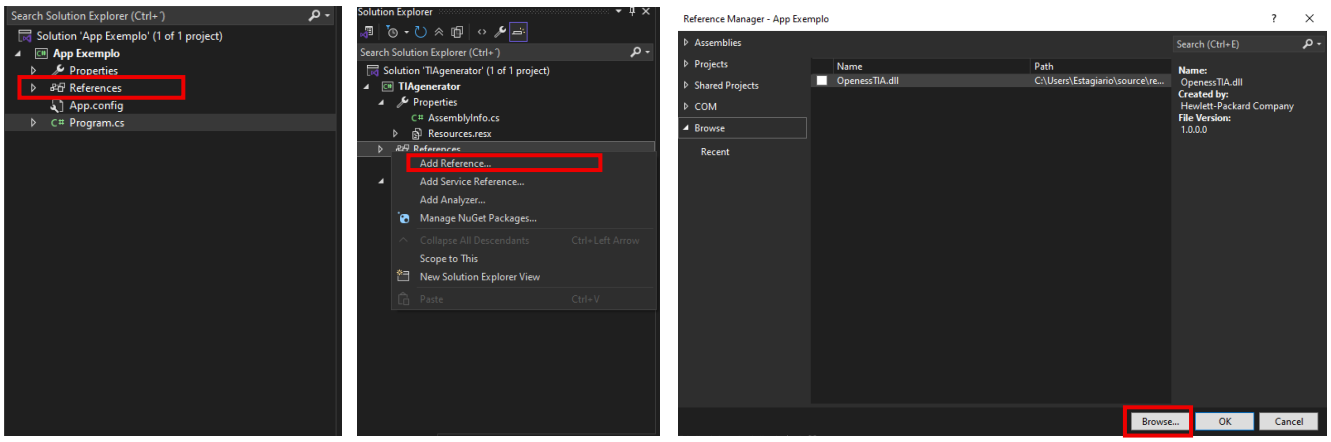


Adicionar a Biblioteca às Referências:

No “Solution Explorer” do projeto, usar o botão direito na opção “References” e adicionar uma referência com a opção “Add Reference”.

Na parte “Browse”, procurar o botão com o mesmo nome para encontrar o ficheiro “OpenessTIA.dll”.





Exemplo de Código da aplicação.

TIA_V18 é uma classe onde se encontram todos os métodos da aplicação. A classe simboliza o programa TIA Portal V18 e por isso tudo na aplicação é feito como se fosse no próprio programa.

1 - Inicio

Sendo assim, será necessário iniciar o programa inicializando a variável TIA_V18:

```
TIA_V18 newTIA = new TIA_V18();
```

Com a variável inicializada, deverão ser usadas duas funções:

```
newTIA.setFilePath(filesPath);

newTIA.createTiaInstance(true);
```

Estas funções permitem definir o caminho da pasta de ficheiros necessária e abrir o TIA Portal V18.

A variável “filesPath” deve ser uma string com a localização no Windows da pasta de ficheiros.

2 - Criar/Abrir um projeto

Após o TIA Portal V18 estar aberto é necessário criar/abrir um projeto. Para isso usar a função:

```
newTIA.createOpenTiaProject(projectPath, projectName);
```

Esta função verifica se o projeto com determinado nome existe.

Se existir abre esse projeto, se não existir, cria um novo projeto no caminho especificado com base no projeto Template presente na pasta de ficheiros inicial.

A variável projectName é uma string com o nome pretendido.

A variável projectPath é uma string da localização do Windows onde será criado o projeto, não sendo a mesma localização da pasta de ficheiros inicial.

3 – Criar PLC e HMI

Após o projeto estar aberto, é necessário criar o PLC e a HMI, caso estes não existam, que serão os devices usados no projeto. Para a criação são usadas as funções:

```
newTIA.createDevicePlc();
```

```
newTIA.createDeviceHMI(isUnified);
```

Estas funções criam por definição um PLC e uma HMI respetivamente. O PLC criado por definição é o "CPU 1512SP F-1 PN" como nome "PLC". A HMI criada por definição é a TP1200 Comfort com o nome "HMI". Também poderá se criada uma HMI unified.

O resto da Biblioteca não suporta HMI's unified

createDevicePlc()

createDeviceHmi()

Para verificar se já existem PLC e/ou HMI usar a função:

```
int info = newTIA.findDevices("PLC", "HMI");
```

Esta função verifica se existem devices com o nome indicado e retorna um valor em cada caso.

Caso não se crie um projeto do zero, é boa ideia executar esta função antes de criar o PLC e a HMI

findDevices()

Depois de serem criados/encontrados os devices é necessário obter o software dos Devices.

Para isso usar as funções:

```
newTIA.getPlcSoftware();
```

```
newTIA.getHmiTarget();
```
←Não Unified

```
newTIA.getHmiSoftware();
```
←Unified

Esta etapa é necessária para ser possível executar funções ligadas à importação e criação de objetos.

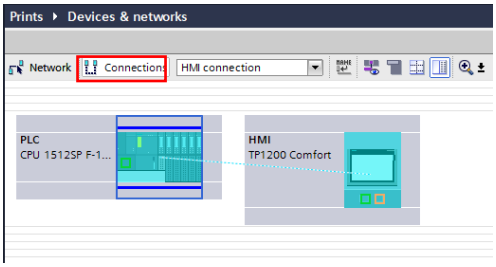
getPlcSoftware()

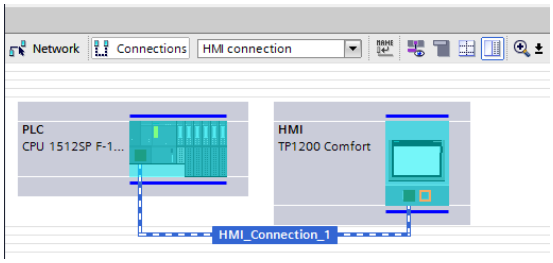
getHmiTarget() ←Não Unified

getHmiSoftware() ← Unified

Também é necessário conectar os devices manualmente. Poderá ser usada a função createConnectionPrompt() ou então implementar uma pausa até serem conectados os Devices.

Esta conexão é feita na parte de “connections” e não de “networks”





A Conexão de “Networks” é feita com a função connectDevices(), que conecta o PLC e a HMI através de uma ligação de IP.

4 – Ler Lista de I/O’s em Excel

Para a ler e importar a lista de Entradas e Saídas do PLC, usar as funções:

```
newTIA.writeXmlPlcTagTableIO(xlsxFilename);
cilindros =  newTIA.CountCylinders(xlsxFilename);
```

Assim é possível escrever a Tag Table de PLC e também obter a lista de Cilindros a criar no projeto.

writeXmlPlcTagTableIO()

CountCylinders()

Também é necessário importar as Cartas/Modules. Para isso usar a função:

```
newTIA.importPlcModules(xlsxFilename);
```

Esta função importa e adiciona as Cartas/Modules e coloca os endereços de entrada e saída corretos.

importPlcModules()

Por fim, para importar a Tag Table anteriormente escrita, usar a função:

```
newTIA.importPlcTagTable();
```

Esta função importa a Tag Table para o PLC criado/encontrado.

importPlcTagTable()

Formato do Ficheiro Excel

O ficheiro em Excel deve seguir um formato específico:

-A lista deve começar na linha 4.

Entradas:

-A Coluna D nunca deve estar vazia, mesmo que os endereços não correspondam aos corretos.

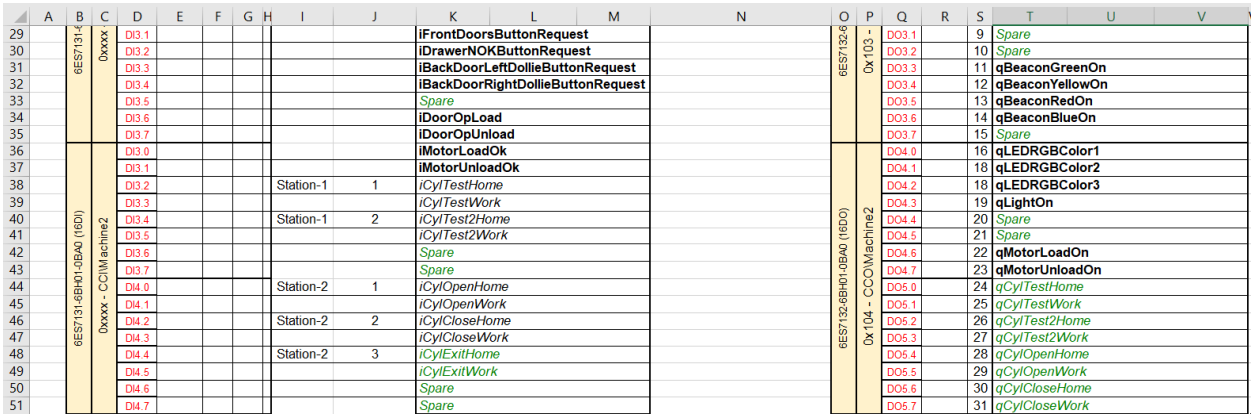
Modules: B Estações: I Postos:J Nomes: K

Saídas:

--A Coluna Q nunca deve estar vazia, mesmo que os endereços não correspondam aos corretos.

Modules: O Nomes: T

Exemplo:



29		6ES7131-4	0xxxx	DI3.1								iFrontDoorsButtonRequest			6ES7132-6	0x103 -	DO3.1		9	Spare		
30				DI3.2								iDrawerNOKButtonRequest					DO3.2		10	Spare		
31				DI3.3								iBackDoorLeftDollieButtonRequest					DO3.3		11	qBeaconGreenOn		
32				DI3.4								iBackDoorRightDollieButtonRequest					DO3.4		12	qBeaconYellowOn		
33				DI3.5								Spare					DO3.5		13	qBeaconRedOn		
34				DI3.6								iDoorOpLoad					DO3.6		14	qBeaconBlueOn		
35				DI3.7								iDoorOpUnload					DO3.7		15	Spare		
36		6ES7131-6BH01-0BA0 (16DI)	0xxxx - CCIMachine2	DI3.0								iMotorLoadOk			6ES7132-6BH01-0BA0 (16DO)	0x104 - CCOMachine2	DO4.0		16	qLEDRGBColor1		
37				DI3.1								iMotorUnloadOk					DO4.1		18	qLEDRGBColor2		
38				DI3.2					Station-1	1		iCylTestHome					DO4.2		18	qLEDRGBColor3		
39				DI3.3								iCylTestWork					DO4.3		19	qLightOn		
40				DI3.4					Station-1	2		iCylTest2Home					DO4.4		20	Spare		
41				DI3.5								iCylTest2Work					DO4.5		21	Spare		
42				DI3.6								Spare					DO4.6		22	qMotorLoadOn		
43				DI3.7								Spare					DO4.7		23	qMotorUnloadOn		
44				DI4.0					Station-2	1		iCylOpenHome					DO5.0		24	qCylTestHome		
45				DI4.1								iCylOpenWork					DO5.1		25	qCylTestWork		
46				DI4.2					Station-2	2		iCylCloseHome					DO5.2		26	qCylTest2Home		
47				DI4.3								iCylCloseWork					DO5.3		27	qCylTest2Work		
48				DI4.4					Station-2	3		iCylExitHome					DO5.4		28	qCylOpenHome		
49				DI4.5								iCylExitWork					DO5.5		29	qCylOpenWork		
50				DI4.6								Spare					DO5.6		30	qCylCloseHome		
51				DI4.7								Spare					DO5.7		31	qCylCloseWork		

5-Importar Global Library e objetos

Para importar a Global Library, usar a função:

```
newTIA.importGlobalLibrary();
```

Com esta função é possível importar a Global Library da pasta de ficheiros “LibraryApp”.

importGlobalLibrary()

Para Importar objetos como as FB’s e UDT’s, usar as funções:

```
newTIA.getUdtFromLibrary();
newTIA.getFbFromLibrary();
```

Estas funções importam os Function Blocks e as User Data Types da Global Library para serem usadas por objetos dentro do projeto.

getUdtFromLibrary()

getFbFromLibrary()

6- Criação de Pastas e Objetos no PLC e HMI

Como haverá mais do que uma estação em cada projeto, será necessário dividir as Estações em listas separadas, para cada objeto ser importado para a pasta correta. Para dividir as estações usar a função:

```
List<List<Peca>> listas = newTIA.divideLists(cilindros);
```

Esta função usa a lista de cilindros criada pela função CountCylinders() e separa em listas diferentes de acordo com as estações.

divideLists()

Para a criação dos objetos por estações, é necessário usar um ciclo, como o for() ou foreach().

As pastas no PLC e na HMI serão criadas a partir do nome das estações onde os objetos serão inseridos. Para isso usamos as seguintes funções:

```
newTIA.createPlcFolders(listas[i]);
newTIA.createScreenFolders(isUnified, listas[i]);
```

Com estas funções, serão criadas pastas correspondentes às estações que foram anteriormente lidas do ficheiro em Excel.

createPlcFolders()

createScreenFolder()

Para cada estação é necessário seguir uma ordem de criação de objetos. A ordem é a seguinte:

DataBlock Geral -> DataBlocks de cada Instância -> FC -> Main Block -> Screen

```
for(int i = 0; i < listas.Count; i++)
    {
        newTIA.createPlcFolders(listas[i]);
        newTIA.createScreenFolders(isUnified, listas[i]);

        newTIA.writeXmlfileDBCylinder( listas[i]);
        newTIA.importDB(listas[i][0].getStation());

        newTIA.getDataBlockFromLibrary(listas[i]);

        newTIA.writeXmlFileFcCylinder(listas[i]);
        newTIA.importFC(listas[i][0].getStation());

        newTIA.writeXmlFileMainBlockCylinder(listas[i]);
        newTIA.importMain(listas[i][0].getStation());

        newTIA.writeXmlFileScreenCylinder(listas[i]);
        newTIA.importScreen(listas[i][0].getStation());



    }
```

Ciclo for() com ordem de execução correta das funções

Com estas funções, nesta ordem de execução, serão criadas pastas no PLC e Hmi com DB's FC's e o Main Block.

writeXmlfileDBCylinder()

writeXmlFileFcCylinder()

writeXmlFileMainBlockCylinder()

writeXmlFileScreenCylinder()

importDB()

importFC()

importMain()

importScreen()

Após a criação das pastas e objetos é necessário importar a Tag Table da HMI, para isso usar as funções:

```
newTIA.writeXmlHmiTagTableCylinder(cilindros, "Cylinder_TagTable");
newTIA.importHmiTagTable();
```

Estas funções escrevem e importam, respetivamente a Tag Table de HMI.

writeXmlHmiTagTableCylinder()

importHmiTagTable()

Para importar as Templates das Screens, usar a função:

```
newTIA.getAllTemplatesFromLibrary(isUnified);
```

Esta função importa todas as templates da Global Library, associando automaticamente as templates às screens que as usam.

getAllTemplatesFromLibrary()

Por fim, para finalizar a aplicação poderá ser usada a função:

```
newTIA.saveProject();
```

Esta função guarda o projeto criado.

saveProject()

Biblioteca de Funções

Regiões:

1-Gets e Sets
2-Abertura do TIA Portal e do Projeto
3-Criar e Encontrar Hmi's e Plc's no Projeto
4-Obtenção do software dos devices
5-Conexão e atribuição de IP's
6-Funções base para criação de pastas, Importar Global Library e Importar Objetos da Global Library
7-Funções de pastas de screens e templates
8-Exportação/Importação de ficheiros XML
9-Funções de Auxílio à escrita de documentos em XML
10-Escrita de documento XML de uma DB de Cilindros
11-Escrita de documento XML de uma FC de Cilindros
12-Escrita de documento XML de uma Screen de Cilindros
13-Escrita do Documento XML de um Main Block de Cilindros
14-Escrita de documento XML de uma TagTable de Cilindros de HMI
15-Imports de Excel

1 – GETS E SETS

```
public int getNumeroDBsCylinder()...
```

Retorna o número de DataBlocks de Cilindro.

```
public string getGlobalLibraryPath()...
```

Retorna uma string com o caminho no Windows da Global Library a Importar.

```
public void setFilePath(string stringFilePath)...
```

Define o caminho no Windows fornecido por stringFilePath para a pasta de ficheiros que contém todos os ficheiros necessários para a aplicação e ficheiros que serão posteriormente criados durante a execução da aplicação.

Exemplo de caminho do Windows: C:\Users\Estagiario\Desktop

```
public void setsourcePath(string filePath)
```

Define o caminho no Windows fornecido por filePath para os ficheiros do projeto Template Usado para a criação de um projeto novo.

2 – ABERTURA DO TIA PORTAL E DO PROJETO

```
public void createTiaInstance(bool guiTIA)...
```

Abre o TIA Portal V18 com ou sem interface de usuário, dependendo de guiTIA.

```
public void createOpenTiaProject(string projectPath, string projectName)...
```

Abre um projeto do TIA Portal com o nome indicado em projectPath. Se não existir um projeto com esse nome, é criado um automaticamente.

A criação do projeto consiste na cópia de ficheiros de um projeto Template para a pasta indicada pelo projectPath.

É necessária a execução prévia da função setFilePath().

```
public void openProjectView()...
```

Abre a janela “Project View” do TIA Portal.

```
static void SetWhitelist(string ApplicationName, string ApplicationStartupPath)...
```

Cria uma whitelist na Registry do Windows para a aplicação poder executar ações dentro do TIA Portal.

```
public void saveProject()...
```

Guarda o projeto atual.

3- CRIAR E ENCONTRAR HMI'S E PLC'S NO PROJETO

```
public void createDevicePlc(string plcName = "PLC", string plcVersion = "V3.0", string plcArticle = "6ES7 512-1SM03-0AB0")
```

Cria um Controlador (PLC) com o nome indicado em plcName.

O modelo do Controlador (PLC) é especificado em plcVersion e plcArticle.

```
public void createDeviceHMI(bool unified = false , string hmiName = "HMI", string hmiVersion = "17.0.0.0", string hmiArticle = "6AV2 124-0MC01-0AX0")...
```

Cria uma HMI com o nome indicado em hmiName.

O modelo da HMI é indicado por hmiVersion e hmiArticle.

O argumento unified especifica se é uma HMI do tipo unified. O restante da biblioteca não utiliza HMI's unified.

```
public int findDevices(string plcName, string hmiName)...
```

Procura Devices com os nomes especificados por plcName e hmiName.

Retorna um número inteiro com código de resultado.

0 – Existe PLC e HMI com o nome fornecido

1- Existe HMI, mas não existe PLC

2- Existe PLC, mas não existe HMI

3- Não existe nem PLC nem HMI

Após a execução a função não executa as funções de criação de PLC e HMI

4- OBTENÇÃO DO SOFTWARE DOS DEVICES

```
public void getPlcSoftware()...
```

Obtém o software do PLC, necessário para a configuração e programação do mesmo.

Deverá ser usada a função findDevices() caso o PLC exista ou a função createDevicePLC() caso o PLC não exista.

```
public void getHmiTarget()...
```

Obtém o Software de uma HMI não Unified, necessário para a configuração e programação do mesmo.

Deverá ser usada a função findDevices() caso o PLC exista ou a função createDeviceHMI() caso o PLC não exista.

```
public void getHmiSoftware()...
```

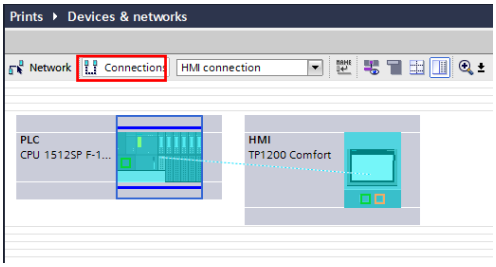
Obtém o Software de uma HMI Unified, necessário para a configuração e programação do mesmo.

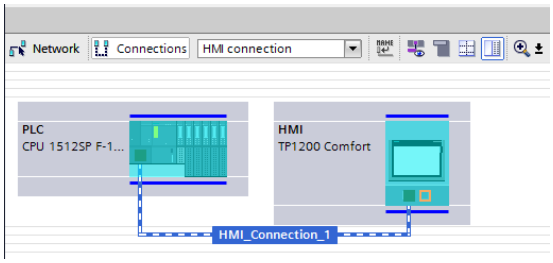
Deverá ser usada a função findDevices() caso o PLC exista ou a função createDeviceHMI() caso o PLC não exista.

5- CONEXÃO E ATRIBUIÇÃO DE IP'S

```
public void createConnectionPrompt()...
```

Prompt na consola pedindo ao usuário para conectar os Devices






```
public void givePlcIPAddress(string ipAddress, string subnetName = "PN/IE_1")...
```

Atribui o endereço de IP fornecido por ipAddress e cria uma subnet ligada ao PLC com o nome fornecido por subnetName.

Deverá ser executada a função getPlcSoftware() previamente

```
public void giveHmiIPAddress(string ipAddress)...
```

Atribui o endereço de IP fornecido por ipAddress e conecta a HMI à subente do projeto.

Deverão ser executadas as funções getHmiTarget() e givePlcIPAddress() previamente.

```
public void connectDevices(string plcIp = "192.168.192.1", string hmiIp = "192.168.192.2", string subnetName = "PN/IE_1")...
```

Executa da forma correta as funções givePlcIPAddress() e giveHmiIPAddress().

Deverão ser executadas as funções getPlcSoftware() e getHmiTarget() previamente.

6- FUNÇÕES BASE PARA CRIAÇÃO DE PASTAS, IMPORTAR GLOBAL LIBRARIES E IMPORTAR OBJETOS DA GLOBAL LIBRARY

```
public void countDataBlocks()...
```

Conta os DataBlocks totais e os Datablocks de Cilindros no TIA Portal

Deve ser executada a função getPlcSoftware()previamente.

```
public void countFCs()...
```

Conta o número de Function totais no TIA Portal

Deve ser executada a função getPlcSoftware()previamente.

```
public void importGlobalLibrary()...
```

Abre a Biblioteca “libraryApp” presente na pasta de ficheiros

```
public void createPlcFolders()...
```

Cria as pastas correspondentes às estações do projeto.

Deve ser executada a função getPlcSoftware()previamente.

```
public void getUdtFromLibrary()...
```

Importa todas as UDTs da Global Library Importada.

Devem ser executadas as funções getPlcSoftware()e importGlobalLibrary()previamente.

```
public void getFbFromLibrary()...
```

Importa todas as FB’s da Global Library Importada.

Devem ser executadas as funções getPlcSoftware()e importGlobalLibrary()previamente.

```
public int getDataBlockFromLibrary(List<Cilindro> listaCilindros, string dbName = "Cylinder")...
```

Importa os DataBlocks a partir de uma lista de Cilindros.

Devem ser executadas as funções getPlcSoftware() e importGlobalLibrary() previamente.

```
public List<List<Cilindro>> divideLists(List<Cilindro> listaCilindros)...
```

Divide a lista de Cilindros fornecida por listaCilindros e retorna uma Lista de Listas de Cilindro.

```
public List<string> getStations(List<Cilindro> listaCilindros)...
```

Cria uma lista de strings com todas as estações diferentes na lista de Cilindros fornecida por listaCilindros

É utilizada pela função divideLists() e é especifica para a utilização desta.

```
public void changeDataBlock(DataBlock db, string name, List<Cilindro> listaCilindros)...
```

Altera o nome de um DataBlock fornecido por db. Altera também o número de acordo com o número de DataBlocks já existentes no projeto.

É utilizada pela função getDataBlockFromLibrary() e é especifica para a utilização desta.

Devem ser executadas as funções getPlcSoftware() e importGlobalLibrary() previamente e deve existir pelo menos um DataBlock.

```
public void importSingleFb()...
```

Importa apenas uma FB.

Devem ser executadas as funções getPlcSoftware() e importGlobalLibrary()previamente.

```
public void copyToProjectLibraryPrompt()...
```

Pede ao Usuário para copiar os elementos da Global Library para a Project Library. Útil para o caso de não ser possível criar um projeto a partir do Template.

7- FUNÇÕES DE PASTAS DE SCREENS E TEMPLATES

```
public void createScreenFolder(bool isUnified,List<Cilindro> listaCilindros)...
```

Cria pastas de screen de acordo com as Stations da listaCilindros.

Deve ser executada a função getHmiTarget() previamente.

```
public void getAllTemplatesFromLibrary(bool isUnified)...
```

Importa todas as Screen Templates da Global Library.

Devem ser executadas as funções getHmiTarget() e importGlobalLibrary() previamente.

```
public void getTemplateFomLibrary(string templateName)...
```

Importa uma Screen Template da Global Library.

É usada pela função getAllTemplatesFromLibary().

Devem ser executadas as funções getHmiTarget()e importGlobalLibrary() previamente.

8 – EXPORTAÇÃO/IMPORTAÇÃO DE FICHEIROS XML

Todos os ficheiros em XML devem ter o formato correto para cada tipo. Se o ficheiro não estiver no formato correto o objeto não é importado e o programa encerra.

```
public void importFB(string stationName)...
```

Importa uma FB da pasta de ficheiros a partir do ficheiro FB_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Deve ser executada a função getPlcSoftware() previamente e deve existir um ficheiro "FB_write.xml" com o formato correto.

```
public void importFC(string stationName)...
```

Importa uma FC da pasta de ficheiros a partir do ficheiro FC_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Para a FC ser importada corretamente, deve existir previamente no projeto a DB da estação, todas as DB's de Instâncias e todos os FB's dos objetos das Networks.

Deve ser executada a função getPlcSoftware()previamente e deve existir um ficheiro "FC_write.xml" com o formato correto.

```
public void importDB(string stationName)...
```

Importa uma DB da pasta de ficheiros a partir do ficheiro DB_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Para a DB ser corretamente importada, devem existir todas as UDT's utilizadas.

Deve ser executada a função getPlcSoftware() previamente e deve existir um ficheiro "DB_write.xml" com o formato correto.

```
public void importMain(string stationName)...
```

Importa uma Main da pasta de ficheiros a partir do ficheiro Main_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Para O Main Block ser importado corretamente, deve existir previamente a FC a ser inserida.

Deve ser executada a função getPlcSoftware() previamente e deve existir um ficheiro "Main_write.xml" com o formato correto.

```
public void importScreen(string stationName)...
```

Importa uma Screen da pasta Inicial a partir do ficheiro Screen_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Para importar a Screen corretamente, devem existir as Faceplates na Project Library.

Deve ser executada a função getHmiTaget() previamente e deve existir um ficheiro "Screen_write.xml" com o formato correto.

```
public void importHmiTagTable()...
```

Importa uma Tag Table de HMI da pasta de ficheiros a partir do ficheiro HmiTagTable_write.xml.

Para a TagTable da Hmi ser importada corretamente devem existir os Datablocks de todas as estações, a ligação entre PLC e HMI (HMI_Connection_1) e todas as UDT's utilizadas.

Deve ser executada a função getHmiTaget()previamente e deve existir um ficheiro "HmiTagTable_write.xml" com o formato correto.

```
public void importPlcTagTable()...
```

Importa uma Tag Table de PLC da pasta de ficheiros a partir do ficheiro PlcTagTable_write.xml.

Deve ser executada a função getHmiTaget()previamente e deve existir um ficheiro "PlcTagTable_write.xml" com o formato correto.

```
public void exportScreen()...
```

Exporta uma Screen para a pasta de ficheiros com o nome Screen.xml

Deve ser executada a função getHmiTaget()previamente.

```
public void exportHmiTagTable()...
```

Exporta uma Tag Table de Hmi para a pasta de ficheiros com o nome TagTable.xml

Deve ser executada a função getHmiTaget()previamente.

```
public void exportPlcTagTable()...
```

Exporta uma Tag Table de PLC para a pasta de ficheiros com o nome TagTable.xml

Deve ser executada a função getPlcSoftware() previamente.

9 – FUNÇÕES DE AUXILIO À ESCRITA DE DOCUMENTOS EM XML

```
public void writeXmlDocumentInfo(XmlWriter writer)...
```

Escreve o Document Info de um objeto do TIA Portal em XML no ficheiro fornecido por writer.

Esta função escreve este código.

Usada por outras funções de escrita de XML.

```
public void writeXmlDocumentInfoTagTable(XmlWriter writer)...
```

Escreve o Document Info de uma Tag Table do TIA Portal em XML no ficheiro fornecido por writer.

Esta função escreve este código.

Usada por outras funções de escrita de XML.

```
public void writeXmlElementWithattribute(string elementString, string elementValue, string attributeString, string attributeValue, XmlWriter writer)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

<elementString attributeString="attributeValue">elementValue</CodeModifiedDate>

Usada por outras funções de escrita de XML.

```
public void writeXmlElementWithTwoattributes(string elementString, string elementValue, string attributeString,
    string attributeValue, string secondAttributeString, string secondAttributeValue, XmlWriter writer)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

<elementString attributeString="attributeValue" secondAttributeString=”secondAttributeValue”>elementValue</CodeModifiedDate>

Usada por outras funções de escrita de XML.

```
public void writeXmlElementWithThreeattributes(string elementString, string elementValue, string attributeString, string attributeValue,
    string secondAttributeString, string secondAttributeValue, string thirdAttributeString, string thirdAttributeValue, XmlWriter writer)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

<elementString attributeString="attributeValue" secondAttributeString=”secondAttributeValue” thirdAttributeString= “thirdAttributeValue”>elementValue</CodeModifiedDate>

Usada por outras funções de escrita de XML.

```
intToHex(int idCounter)
```

Retorna uma string com o número inteiro decimal transformado em hexadecimal.

Usada por outras funções de escrita de XML.

10 – ESCRITA DE DOCUMENTO XML DE UMA DB DE CILINDROS

```
public void writeXmlMemberElementCylinder(string name, string dataType, XmlWriter writer)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

<Member Name="name" Datatype="dataType"/>

Usada exclusivamente na função writeXmlInterfaceDbCylinder().

```
public void writeXmlInterfaceDbCylinder(XmlWriter writer, List<Cilindro> listaCilindros)...
```

Escreve a interface de uma DB de Cilindros no ficheiro fornecido por writer.

Usada exclusivamente na função writeXmlInterfaceDbCylinder().

```
public void writeXmlfileDBCylinder( List<Cilindro> listaCilindros)...
```

Escreve o ficheiro em XML de um DataBlock com os cilindros fornecidos pela listaCilindros para depois ser importado na pasta de ficheiros com o nome DB_write.xml.

11 - ESCRITA DE DOCUMENTO XML DE UMA FC DE CILINDROS

```
public void writeXmlSingleWire(XmlWriter writer, int Uid_1, int Uid_2, int Uid_3, string name)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

```
<Wire UId="Uid_1">
        <OpenCon UId="Uid_2"/>
        <NameCon UId="Uid_3" Name="name "/>
</Wire>
```

Usada exclusivamente na função writeXmlWires().

```
public void writeXmlWiresCylinder(XmlWriter writer, int callUid)...
```

Escreve as Wires da FC no ficheiro fornecido por writer.

Usada exclusivamente na função writeXmlNetorksFcCylinder().

```
public void writeXmlPartParameterCylinder(string name, string section, string type, XmlWriter writer)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

```
<Parameter Name="name" Section="section" Type="type ">
        <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
</Parameter>
```

Usada exclusivamente na função writeXmlNetorksFcCylinder().

```
public int writeXmlNetorksFcCylinder(int idCounter, XmlWriter writer, List<Cilindro> listaCilindros)...
```

Escreve as Networks da FC de acordo com a listaCilindros no ficheiro fornecido por writer.

Usada exclusivamente na função writeXmlFileFcCylinder().

```
public void writeXmlInterfaceFcCylinder(XmlWriter writer)...
```

Escreve a Interface da FC no ficheiro fornecido por writer.

Usada exclusivamente na função writeXmlFileFcCylinder().

```
public void writeXmlFileFcCylinder(List<Cilindro> listaCilindros)...
```

Escreve o ficheiro em XML de uma FC com os cilindros fornecidos pela listaCilindros para depois ser importado na pasta de ficheiros com o nome FC_write.xml.

12 - ESCRITA DE DOCUMENTO XML DE UMA SCREEN COM CILINDROS

```
public int writeFaceplateInstancesCylinder(XmlWriter writer, int idCounter, List<Cilindro> listaCilindros )...
```

Escreve a instâncias de Cilindros de acordo com a listaCilindros no ficheiro fornecido por writer.

Usada exclusivamente pela função writeXmlFileScreenCylinder().

```
public void writeXmlFileScreenCylinder(List<Cilindro> listaCilindros, string templateName = "Template")...
```

Escreve o ficheiro em XML de uma Screen com os cilindros fornecidos pela listaCilindros para depois ser importado na pasta de ficheiros com o nome Screen_write.xml.

13 - ESCRITA DO DOCUMENTO XML DE UM MAIN BLOCK DE CILINDROS

```
public void writeXmlFileMainBlockCylinder(List<Cilindro> listaCilindros)...
```

Escreve o ficheiro em XML de um Main Block com a FC já criada anteriormente na pasta de ficheiros com o nome Main_write.xml.

14 - ESCRITA DO DOCUMENTO XML DE UMA TAG TABLE DE HMI

```
public int writeSingleIdTagTableObjectCylinderStructure(XmlWriter writer, int idCounter, string name)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

```
<Hmi.Tag.TagStructureMember ID="idCounter" CompositionName="Members">
        <AttributeList>
                <AcquisitionTriggerMode>Visible</AcquisitionTriggerMode>
                <LinearScaling>false</LinearScaling>
                <LogicalAddress/>
                <Name>name</Name>
                <ScalingHmiHigh>100</ScalingHmiHigh>
                <ScalingHmiLow>0</ScalingHmiLow>
                <ScalingPlcHigh>10</ScalingPlcHigh>
                <ScalingPlcLow>0</ScalingPlcLow>
                <StartValue/>
                <SubstituteValue/>
```

```
                <SubstituteValueUsage>None</SubstituteValueUsage>
        </AttributeList>
        <ObjectList>
                <MultilingualText ID=" idCounter " CompositionName="Comment">
                        <ObjectList>
                                <MultilingualTextItem ID=" idCounter " CompositionName="Items">
                                        <AttributeList>
                                                <Culture>en-US</Culture>
                                                <Text/>
                                        </AttributeList>
                                </MultilingualTextItem>
                        </ObjectList>
                </MultilingualText>
        </ObjectList>
</Hmi.Tag.TagStructureMember>
```

A Variável idCounter é usada pela função intToHex() para escrever o ID no formato hexadecimal. Usada por outras funções de escrita de XML de Tag Table de Hmi.

```
public int writeDoubleIdTagTableObjectCylinder(XmlWriter writer, int idCounter)...
```

Escreve a seguinte estrutura no ficheiro fornecido por writer:

```
<MultilingualText ID="idCounter" CompositionName="Comment">
        <ObjectList>
                <MultilingualTextItem ID="idCounter" CompositionName="Items">
                        <AttributeList>
                                <Culture>en-US</Culture>
                                <Text/>
                        </AttributeList>
                </MultilingualTextItem>
        </ObjectList>
</MultilingualText>
```

A Variável idCounter é usada pela função intToHex() para escrever o ID no formato hexadecimal Usada por outras funções de escrita de XML de Tag Table de HMI.

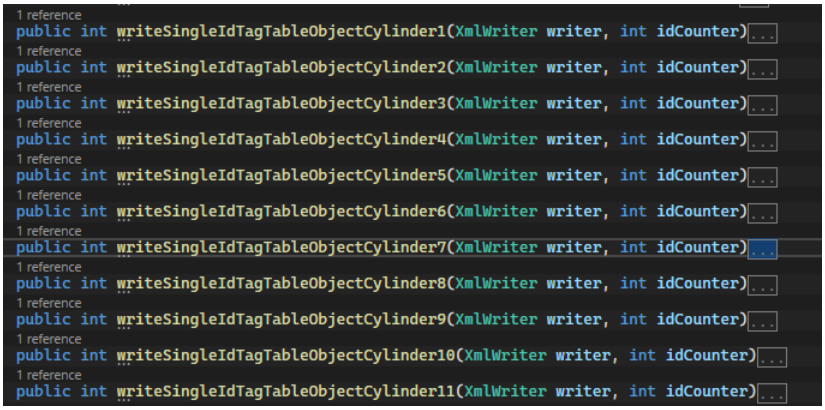
```
public int writeTagTableMembersCylinder(XmlWriter writer, int idCounter, List<Cilindro> listaCilindros)...
```

Escreve os membros da TagTable da HMI de acordo com a listaCilindros no ficheiro fornecido por writer.

Usada exclusivamente pela função writeXmlHmiTagTableCylinder().

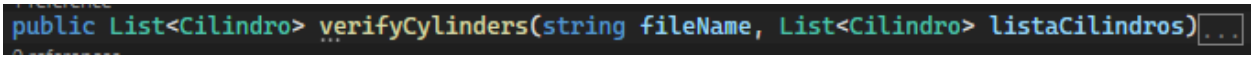
```
public void writeXmlHmiTagTableCylinder(List<Cilindro> listaCilindros, string name)...
```

Escreve o ficheiro em XML de uma Tag Table com os cilindros fornecidos pela listaCilindros para depois ser importado na pasta de ficheiros com o nome HmiTagTable_write.xml.



Funções que escrevem estruturas específicas da Tag Table de HMI de Cilindros no ficheiro fornecido por writer.

15 – IMPORTS DO EXCEL



Verifica na lista do Excel se existem todas as entradas e saídas de um dado Cilindro:

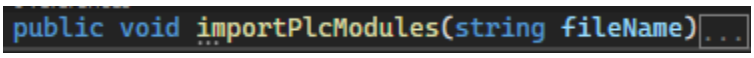
iCyl…Home iCyl…Work qCyl…Home qCyl…Work.

Usada exclusivamente pela função CountCylinders().

Lê um ficheiro de Excel na pasta de ficheiros com o nome especificado por fileName e cria a lista de Cilindros a partir da mesma.

A Atribuição de estação, posto e tipo de peça ocorre na linha de iCyl…Home.

Escreve o ficheiro em XML de uma Tag Table de PLC para ser importado na pasta de ficheiros com o nome FC_write.xml. A função escreve a o ficheiro em XML a partir do ficheiro em Excel na pasta de ficheiros com o nome especificado por fileName.



Lê o ficheiro em Excel na pasta de ficheiros com o nome especificado por fileName e importa as cartas/modules.

A partir do código especificado em code adiciona uma carta/module ao PLC e retorna o número de linhas a avançar no ficheiro em Excel.

Usada exclusivamente na função importPlcModules().

Para facilitar o trabalho com listas, foi criada a classe “Peca” com os parâmetros “name”, “station”, “nest” e “type”.

```
public class Peca
{
    public string name;
    public string station;
    public string nest;
    public string type;

    //Construtor
    0 references
    public Peca(string nameString = "", string stationString = "", string nestString = "", string typeString = "")
    {
        name = nameString;
        station = stationString;
        nest = nestString;
        type = typeString;
    }

    Gets e Sets

}
```

Expansão da Biblioteca

A expansão da Biblioteca será inteiramente ligada ao acrescentar novas peças para além dos cilindros.

1 - Funções a Alterar ou Criar:

Criar ou alterar funções e regiões com o mesmo propósito dos seguintes métodos:

writeXmlfileDBCylinder()

writeXmlFileFcCylinder()

writeXmlFileScreenCylinder()

writeXmlFileHmiTagTableCylinder()

Seria necessário alterar estas funções e as suas regiões, criando funções com a mesma operação de funções já existentes, mas para novas peças.

Como todas as peças são diferentes, todas as peças terão de ser visualizadas caso a caso e escritas individualmente, não sendo viável a escrita de uma função que possa escrever o código XML de qualquer peça.

2 - DataBlocks:

Criar uma função semelhante a writeXmlInterfaceDbCylinder() mas para outra peça e inicializa-la em writeXmlfileDBCylinder();

Um exemplo para ilustrar como funcionaria com uma função de escrita de Conveyor Belts

É necessário criar uma nova função pois as Function Blocks e variáveis de outras peças seriam completamente diferentes das FB's e variáveis de Cilindros.

É também importante manter o mesmo formato de nome da DB, para assim ser mais fácil a sua associação com as FC's e Tag Tables da HMI. O nome tem uma estrutura simples sendo ela:

DB (Nome da estação)

```
dbName = "DB " + listaCilindros[i].getStation();
```

Exemplo de uso de uma nova função:

```
writer.WriteStartElement("Interface");
    writer.WriteStartElement("Sections", "http://www.siemens.com/automation/Openness/SW/Interface/v5");
    writer.WriteAttributeString("xmlns", "http://www.siemens.com/automation/Openness/SW/Interface/v5");
        writer.WriteStartElement("Section");
        writer.WriteAttributeString("Name", "Static");
writeXmlInterfaceDbCylinder(writer, listaCilindros);
//writeXmlInterfaceDbConveyor(writer, listaConveyors);
        writer.WriteEndElement();
    writer.WriteEndElement();
writer.WriteEndElement();
```

Neste exemplo, seria necessário um sistema de separação entre Cilindros, Conveyors e outras peças que poderá ser feito através do parâmetro "type" da classe "Peca".

Exemplo de Código XML de membros de um DataBlock

```
<SW.Blocks.GlobalDB ID="0">
  <AttributeList>
    <AutoNumber>true</AutoNumber>
    <CodeModifiedDate ReadOnly="true">2025-03-19T15:17:54.4622546Z</CodeModifiedDate>
    <CompileDate ReadOnly="true">2025-03-19T15:18:53.1916012Z</CompileDate>
    <CreationDate ReadOnly="true">2025-03-19T15:17:54.4622546Z</CreationDate>
    <DBAccessibleFromOPCUA>true</DBAccessibleFromOPCUA>
    <DBAccessibleFromWebserver>true</DBAccessibleFromWebserver>
    <DownloadWithoutReinit ReadOnly="true">false</DownloadWithoutReinit>
    <HeaderAuthor/>
    <HeaderFamily/>
    <HeaderName/>
    <HeaderVersion>0.1</HeaderVersion>
    <Interface>
      <Sections xmlns="http://www.siemens.com/automation/Openness/SW/Interface/v5">
        <Section Name="Static">
          <Member Name="Open" Datatype=""CTRL_Cylinder"" Remanence="NonRetain" Accessibility="Public">
            ...
          </Member>
          <Member Name="Close" Datatype=""CTRL_Cylinder"" Remanence="NonRetain" Accessibility="Public">
            ...
          </Member>
          <Member Name="Exit" Datatype=""CTRL_Cylinder"" Remanence="NonRetain" Accessibility="Public">
            ...
          </Member>
        </Section>
      </Sections>
    </Interface>
    <InterfaceModifiedDate ReadOnly="true">2025-03-19T15:17:54.4622546Z</InterfaceModifiedDate>
    <IsConsistent ReadOnly="true">true</IsConsistent>
    <IsKnowHowProtected ReadOnly="true">false</IsKnowHowProtected>
    <IsOnlyStoredInLoadMemory>false</IsOnlyStoredInLoadMemory>
    <IsPLCDB ReadOnly="true">false</IsPLCDB>
    <IsRetainMemResEnabled>false</IsRetainMemResEnabled>
    <IsWriteProtectedInAS>false</IsWriteProtectedInAS>
    <MemoryLayout>Optimized</MemoryLayout>
    <MemoryReserve>100</MemoryReserve>
    <ModifiedDate ReadOnly="true">2025-03-19T15:17:54.4622546Z</ModifiedDate>
    <Name>DB Station-2</Name>
    <Namespace/>
    <Number>8</Number>
    <ParameterModified ReadOnly="true">2025-03-19T15:17:54.4622546Z</ParameterModified>
    <ProgrammingLanguage>DB</ProgrammingLanguage>
    <StructureModified ReadOnly="true">2025-03-19T15:17:54.4622546Z</StructureModified>
  </AttributeList>
  <ObjectList>
    ...
  </ObjectList>
</SW.Blocks.GlobalDB>
```

DB Station-2

	Name	Data type	Start value	Retain	Accessible f...	Write...	Visible in ...	Setpoint
1	Static							
2	Open	"CTRL_Cylinder"						
3	Close	"CTRL_Cylinder"						
4	Exit	"CTRL_Cylinder"						

O código XML diferente seria o código dentro do membro e o seu Datatype. O código de outra peça será inevitavelmente diferente e terá de ser escrito individualmente. Cada peça terá de ser vista caso a caso. Por isso seria uma boa ideia criar uma função como "writeXmlInterfaceDbCylinder", mas para outras peças.

Código em XML de um membro do tipo Cyllinder dentro da DB:

3 - FC's

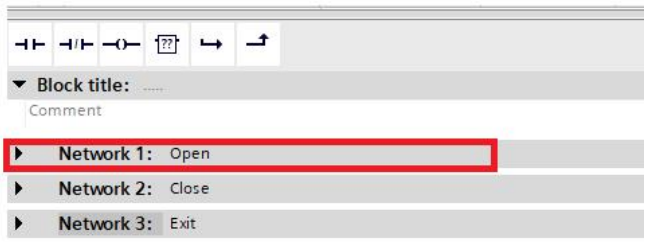
No caso das Fc's seria o mesmo. Seria Necessário alterar as funções writeXmlWiresCylinder() e writeXmlNetorksFcCylinder() ou criar funções novas.

```
2816                    //Start Object List
2817                    writer.WriteStartElement("ObjectList");
2818                        writer.WriteStartElement("MultilingualText");
2819                        writer.WriteAttributeString("ID", "1");
2820                        writer.WriteAttributeString("CompositionName", "Comment");
2821                            writer.WriteStartElement("ObjectList");
2822                                writer.WriteStartElement("MultilingualTextItem");
2823                                writer.WriteAttributeString("ID", "2");
2824                                writer.WriteAttributeString("CompositionName", "Items");
2825                                                writer.WriteStartElement("AttributeList");
2826                                                    writer.WriteElementString("Culture", "en-US");
2827                                                    writer.WriteElementString("Text", "");
2828                                                writer.WriteEndElement();
2829                                            writer.WriteEndElement();
2830                                        writer.WriteEndElement();
2831                                    writer.WriteEndElement();
2832                    idCounter = 3;
2833
2834                     //Start Networks
2835
2836                     idCounter = writeXmlNetworksFcCylinder(idCounter, writer, listaCilindros);
2837                    //idCounter = writeXmlNetworksFcConveyor(idCounter, writer, listaConveyors);
2838
2839                    //End Networks
```

Este exemplo também usaria um sistema de separação de peças por tipo.

Código XML das Networks de uma FC

```
▼<ObjectList>
  ▼<MultilingualText ID="1" CompositionName="Comment">
    ▼<ObjectList>
      ▼<MultilingualTextItem ID="2" CompositionName="Items">
        ▼<AttributeList>
            <Culture>en-US</Culture>
            <Text/>
          </AttributeList>
        </MultilingualTextItem>
    </ObjectList>
  </MultilingualText>
  ▶<SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
    ...
    </SW.Blocks.CompileUnit>
  ▶<SW.Blocks.CompileUnit ID="8" CompositionName="CompileUnits">
    ...
    </SW.Blocks.CompileUnit>
  ▶<SW.Blocks.CompileUnit ID="d" CompositionName="CompileUnits">
    ...
    </SW.Blocks.CompileUnit>
  ▼<MultilingualText ID="12" CompositionName="Title">
    ▼<ObjectList>
      ▼<MultilingualTextItem ID="13" CompositionName="Items">
        ▼<AttributeList>
            <Culture>en-US</Culture>
            <Text/>
          </AttributeList>
        </MultilingualTextItem>
    </ObjectList>
```



Os ID's são todos sequenciais mas têm de estar na forma Hexadecimal. Por isso deve ser usada a função intToHex().

Exemplo de utilização de ID's:

```
writer.WriteAttributeString("ID", intToHex(idCounter));
idCounter++;
```

Código XML de uma Network de Cilindro:

Ainda dentro das Networks existem os Wires. Estas são escritas pela função writeXmlWiresCilindro(). Esta função é executada dentro da função writeXmlNetorksFcCylinder().

Tal como as próprias networks, os Wires também terão de ser diferentes para outras peças tendo de ser vistos e escritos caso a caso.

```
//Start Wires
writer.WriteStartElement("Wires");
writeXmlWiresCylinder(writer, callUid);
//writeXmlWiresConveyor(writer, callUid);
writer.WriteEndElement();
//writeXmlWire();
```

Na escrita das FC é necessário associá-las à DB da estação e à FB a ser inserida.

Isso pode ser feito desta forma:

1 - Ligação com FB

```
▼<CallInfo Name="FB_Cylinder" BlockType="FB">
   <IntegerAttribute Name="BlockNumber" Informative="true">5</IntegerAttribute>
   <DateAttribute Name="ParameterModifiedTS" Informative="true">2024-07-16T16:22:51</DateAttribute>
  ▼<Instance Scope="GlobalVariable" UId="35">
     <Component Name="DB Open - 1"/>
     <Address Area="DB" Type="FB_Cylinder" BlockNumber="10" BitOffset="0" Informative="true"/>
   </Instance>
```

```
writer.WriteAttributeString("UId", callUid.ToString());
    writer.WriteStartElement("CallInfo");
    writer.WriteAttributeString("Name", "FB_Cylinder");
    writer.WriteAttributeString("BlockType", "FB");
        string fbBlockNumber = plcSoftware.BlockGroup.Groups.Find("FBs").Blocks.Find("FB_Cylinder").Number.ToString();
        writeXmlElementWithTwoattributes("IntegerAttribute", fbBlockNumber, "Name", "BlockNumber",  "Informative", "true", writer);
        writeXmlElementWithTwoattributes("DateAttribute", "2024-07-16T16:22:51", "Name", "ParameterModifiedTS",  "Informative", "true", writer);
            writer.WriteStartElement("Instance");
            writer.WriteAttributeString("Scope", "GlobalVariable");
            writer.WriteAttributeString("UId", (callUid+1).ToString());
                numCilindroString = "DB " + listaCilindros[i].getName() + " - " + listaCilindros[i].getNest() ;
                writer.WriteStartElement("Component");
                writer.WriteAttributeString("Name", numCilindroString);
                writer.WriteEndElement();
```

2 - Ligação com DB

```
<Access Scope="GlobalVariable" UId="21">
  <Symbol>
    <Component Name="DB Station-2"/>
    <Component Name="Open"/>
    <Component Name="Enable"/>
    <Component Name="home"/>
    <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="464" Informative="true"/>
  </Symbol>
</Access>
```

```
writer.WriteStartElement("Access");
writer.WriteAttributeString("Scope", "GlobalVariable");
writer.WriteAttributeString("UId", "25");
writer.WriteStartElement("Symbol");
writer.WriteStartElement("Component");
nameAttribute = "DB " + listaCilindros[i].getStation();
writer.WriteAttributeString("Name", nameAttribute);
writer.WriteEndElement();
writer.WriteStartElement("Component");
writer.WriteAttributeString("Name", listaCilindros[i].getName());
writer.WriteEndElement();
writer.WriteStartElement("Component");
writer.WriteAttributeString("Name", "Order");
writer.WriteEndElement();
writer.WriteStartElement("Component");
writer.WriteAttributeString("Name", "home");
writer.WriteEndElement();


writer.WriteStartElement("Address");
writer.WriteAttributeString("Area", "None");
writer.WriteAttributeString("Type", "Bool");
dbName = "DB " + listaCilindros[i].getStation();
blockNumber = plcSoftware.BlockGroup.Groups.Find(listaCilindros[i].getStation()).Blocks.Find(dbName).Number.ToString();
writer.WriteAttributeString("BlockNumber", blockNumber);

writer.WriteAttributeString("BitOffset", (480  + (i * 568)).ToString());
writer.WriteAttributeString("Informative", "true");

writer.WriteEndElement();
writer.WriteEndElement();
writer.WriteEndElement();
```

O nome das DB's terão sempre a mesma estrutura, o que facilita a sua associação com a FC e posteriormente com a TagTable da HMI.

Para a posterior importação funcionar estas DB's e FB's devem já existir no projeto.

4 - Screens

Criar funções semelhantes a writeFaceplateInstancesCylinder() mas com outras peças.

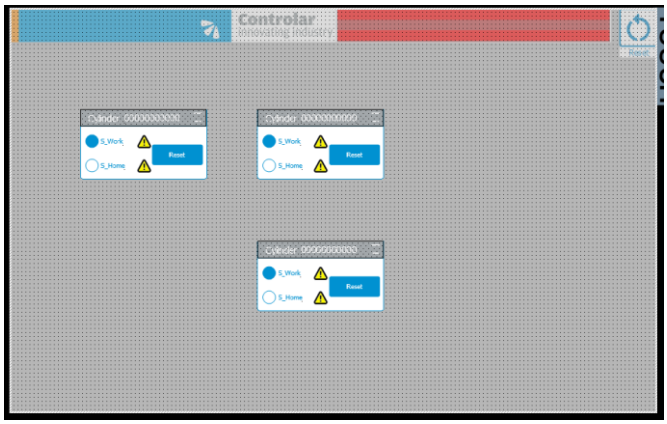
```
writer.WriteStartElement("ObjectList");            //Start Object List de Faceplates

writeFaceplateInstancesCylinder(writer, idCounter, listaCilindros);
//writeFaceplateInstancesConveyor(writer, idCounter, listaConveyors);

writer.WriteEndElement();                          //End Object List de Faceplates
```

Também usaria um sistema de separação de peças por tipo.

```
<ObjectList>
  <Hmi.Screen.FaceplateInstance ID="4" CompositionName="ScreenItems">
    ...
  </Hmi.Screen.FaceplateInstance>
  <Hmi.Screen.FaceplateInstance ID="8" CompositionName="ScreenItems">
    ...
  </Hmi.Screen.FaceplateInstance>
  <Hmi.Screen.FaceplateInstance ID="c" CompositionName="ScreenItems">
    ...
  </Hmi.Screen.FaceplateInstance>
</ObjectList>
```



Na escrita das Screens também é preciso ter em conta os ID's e escrevê-los em Hexadecimal usando a função intToHex().

```
writer.WriteAttributeString("ID", intToHex(idCounter));
idCounter++;
```

Código em XML de Faceplates de Cilindro:

Para a posterior importação funcionar, a Faceplate deve existir na Project Library (Não pode ser a Global Library). O Projeto Template já possui uma Project Library com diversas Faceplates, mas no caso de ser necessário adicionar uma nova deve ser adicionado ao Projeto Template e guardado.

5 - Tag Table da HMI

Sem dúvida a parte mais difícil.

Será necessário criar funções semelhantes a writeTagTableMembersCylinder()

```
//Start Object List (Tag Table Members)
    writer.WriteStartElement("ObjectList");
        idCounter = writeTagTableMembersCylinder(writer, idCounter, listaCilindros);
        //idCounter = writeTagTableMembersConveyor(writer, idCounter, listaConveyors);
    writer.WriteEndElement();
//End Object List (Tag Table Members)
```

Também usaria um sistema de separação de peças por tipo.

```
<ObjectList>
  <Hmi.Tag.Tag ID="1" CompositionName="Tags">
    ...
  </Hmi.Tag.Tag>
  <Hmi.Tag.Tag ID="96" CompositionName="Tags">
    ...
  </Hmi.Tag.Tag>
  <Hmi.Tag.Tag ID="128" CompositionName="Tags">
    ...
  </Hmi.Tag.Tag>
  <Hmi.Tag.Tag ID="1ba" CompositionName="Tags">
    ...
  </Hmi.Tag.Tag>
  <Hmi.Tag.Tag ID="24c" CompositionName="Tags">
    ...
  </Hmi.Tag.Tag>
</ObjectList>
```

Cylinder_TagTable

Name	Data type	Connection	PLC name	PLC tag	Address
Close	CTRL_Cylinder	HMI_Conne...	PLC	"DB Station-2".Close	
Exit	CTRL_Cylinder	HMI_Connectio...	PLC	"DB Station-2".Exit	
Open	CTRL_Cylinder	HMI_Connectio...	PLC	"DB Station-2".Open	
Test	CTRL_Cylinder	HMI_Connectio...	PLC	"DB Station-1".Test	
Test2	CTRL_Cylinder	HMI_Connectio...	PLC	"DB Station-1".Test2	

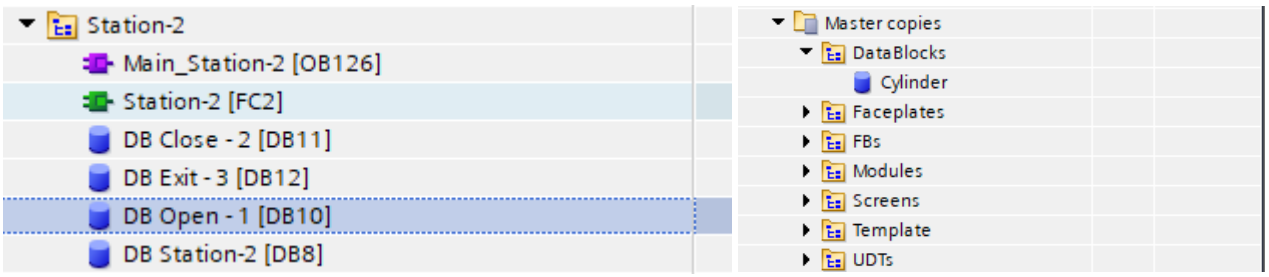
Esta é a parte mais difícil pois os membros da TagTable de HMI são muito extensos e pela necessidade de ver cada peça caso a caso pode ser um trabalho demorado.

Foram criadas diversas funções auxiliares que infelizmente não servirão para a escrita do código XML de novas peças. Estas funções não são necessárias mas ajudam a não deixar muito código em apenas uma única função.

6 - DataBlocks de instâncias.

Existe também a necessidade de importar os dataBlocks de cada instância para a correta Importação das FC's. A função getDataBlockFromLibrary() já está preparada para isso.

A função importa a DB de instância do tipo pretendido caso a DB exista como uma MasterCopy da Global Library na pasta "DataBlocks". A DB na Global Library deve ter o mesmo nome do tipo da peça.



Excertos de Código em XML

1 – Código em XML escrito pela função writeXmlDocumentInfo()

```
<DocumentInfo>
  <Created>2025-03-19T15:18:57.2065646Z</Created>
  <ExportSetting>WithDefaults, WithReadOnly</ExportSetting>
  <InstalledProducts>
    <Product>
      <DisplayName>Totally Integrated Automation Portal</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
    <OptionPackage>
      <DisplayName>TIA Portal Openness</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <OptionPackage>
      <DisplayName>TIA Portal Version Control Interface</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <Product>
      <DisplayName>STEP 7 Professional</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
    <OptionPackage>
      <DisplayName>STEP 7 Safety</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <Product>
      <DisplayName>WinCC Advanced / Unified PC</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
  </InstalledProducts>
</DocumentInfo>
```

2 – Código em XML escrito peça função writeXmlDocumentInfoTagTable()

```
<DocumentInfo>
  <Created>2025-05-06T10:13:32.3319099Z</Created>
  <ExportSetting>WithDefaults</ExportSetting>
  <InstalledProducts>
    <Product>
      <DisplayName>Totally Integrated Automation Portal</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
    <OptionPackage>
      <DisplayName>TIA Portal Openness</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <OptionPackage>
      <DisplayName>TIA Portal Version Control Interface</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <Product>
      <DisplayName>STEP 7 Professional</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
    <OptionPackage>
      <DisplayName>STEP 7 Safety</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </OptionPackage>
    <Product>
      <DisplayName>WinCC Advanced / Unified PC</DisplayName>
      <DisplayVersion>V18</DisplayVersion>
    </Product>
  </InstalledProducts>
</DocumentInfo>
```

3 - Código em XML de apenas um membro da função writeXmlInterfaceDbCylinder()

```
<Member Name="Open" Datatype=""CTRL_Cylinder"" Remanence="NonRetain" Accessibility="Public">
  <AttributeList>
    <BooleanAttribute Name="ExternalAccessible" SystemDefined="true">true</BooleanAttribute>
    <BooleanAttribute Name="ExternalVisible" SystemDefined="true">true</BooleanAttribute>
    <BooleanAttribute Name="ExternalWritable" SystemDefined="true">true</BooleanAttribute>
    <BooleanAttribute Name="UserVisible" Informative="true" SystemDefined="true">true</BooleanAttribute>
    <BooleanAttribute Name="UserReadOnly" Informative="true" SystemDefined="true">false</BooleanAttribute>
    <BooleanAttribute Name="UserDeletable" Informative="true" SystemDefined="true">true</BooleanAttribute>
    <BooleanAttribute Name="SetPoint" SystemDefined="true">false</BooleanAttribute>
  </AttributeList>
  <Sections>
    <Section Name="None">
      <Member Name="name" Datatype="String[20]"/>
      <Member Name="Status" Datatype=""CTRL_DeviceStatus"">
        <Sections>
          <Section Name="None">
            <Member Name="ready" Datatype="Bool"/>
            <Member Name="done" Datatype="Bool"/>
            <Member Name="busy" Datatype="Bool"/>
            <Member Name="idle" Datatype="Bool"/>
            <Member Name="nextDeviceReady" Datatype="Bool"/>
            <Member Name="error" Datatype="Bool"/>
            <Member Name="reset" Datatype="Bool"/>
            <Member Name="step" Datatype="Int"/>
            <Member Name="homeStep" Datatype="Int"/>
            <Member Name="manualMode" Datatype="Bool"/>
            <Member Name="homingOrder" Datatype="Bool"/>
            <Member Name="homed" Datatype="Bool"/>
            <Member Name="clock" Datatype="Bool"/>
            <Member Name="maximized" Datatype="Bool"/>
          </Section>
        </Sections>
      </Member>
      <Member Name="Enable" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
      </Member>
      <Member Name="Order" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
        <Member Name="hmiHome" Datatype="Bool"/>
        <Member Name="hmiWork" Datatype="Bool"/>
      </Member>
      <Member Name="Time" Datatype="Struct">
        <Member Name="filterHome" Datatype="Time"/>
        <Member Name="filterWork" Datatype="Time"/>
        <Member Name="timeoutHome" Datatype="Time"/>
        <Member Name="timeoutWork" Datatype="Time"/>
      </Member>
      <Member Name="Sensor" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
      </Member>
      <Member Name="Output" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
      </Member>
      <Member Name="Position" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
      </Member>
      <Member Name="Error" Datatype="Struct">
        <Member Name="home" Datatype="Bool"/>
        <Member Name="work" Datatype="Bool"/>
      </Member>
      <Member Name="hmiMaximized" Datatype="Struct">
        <Member Name="errorHome" Datatype="Bool"/>
        <Member Name="errorWork" Datatype="Bool"/>
        <Member Name="error" Datatype="Bool"/>
      </Member>
      <Member Name="doesNotRetainOutput" Datatype="Bool"/>
    </Section>
  </Sections>
</Member>
```

4 – Código XML de uma Network de Cilindro escrito pela função writeXmlNetworksFcCylinder()

Início

```
<SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
  <AttributeList>
    <NetworkSource>
      <FlgNet xmlns="http://www.siemens.com/automation/Openness/SW/NetworkSource/FlgNet/v4">
        <Parts>
          <Access Scope="GlobalVariable" UId="21">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Enable"/>
              <Component Name="home"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="464" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="22">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Enable"/>
              <Component Name="work"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="472" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="23">
            <Symbol>
              <Component Name="iCylOpenHome"/>
              <Address Area="Input" Type="Bool" BitOffset="130" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="24">
            <Symbol>
              <Component Name="iCylOpenWork"/>
              <Address Area="Input" Type="Bool" BitOffset="131" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="25">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Order"/>
              <Component Name="home"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="480" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="26">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Order"/>
              <Component Name="work"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="488" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="27">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Address Area="None" Type="CTRL_Cylinder" BlockNumber="8" BitOffset="32" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="28">
            <Symbol>
              <Component Name="qCylOpenHome"/>
              <Address Area="Output" Type="Bool" BitOffset="376" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="29">
            <Symbol>
              <Component Name="qCylOpenWork"/>
              <Address Area="Output" Type="Bool" BitOffset="377" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="30">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Enable"/>
              <Component Name="home"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="464" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="31">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Enable"/>
              <Component Name="work"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="472" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="32">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Order"/>
              <Component Name="home"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="480" Informative="true"/>
            </Symbol>
          </Access>
          <Access Scope="GlobalVariable" UId="33">
            <Symbol>
              <Component Name="DB Station-2"/>
              <Component Name="Open"/>
              <Component Name="Order"/>
              <Component Name="work"/>
              <Address Area="None" Type="Bool" BlockNumber="8" BitOffset="488" Informative="true"/>
            </Symbol>
          </Access>
          <Call UId="34">
            <CallInfo Name="FB_Cylinder" BlockType="FB">
              <IntegerAttribute Name="BlockNumber" Informative="true">5</IntegerAttribute>
              <DateAttribute Name="ParameterModifiedTS" Informative="true">2024-07-16T16:22:51</DateAttribute>
              <Instance Scope="GlobalVariable" UId="35">
                <Component Name="DB Open - 1"/>
                <Address Area="DB" Type="FB_Cylinder" BlockNumber="10" BitOffset="0" Informative="true"/>
              </Instance>
              <Parameter Name="name" Section="Input" Type="String[20]">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="enableHome" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="enableWork" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="doorOpen" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="manualMode" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="reset" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="iHome" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="iWork" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="orderHome" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="orderWork" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="doesNotRetainOutput" Section="Input" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="timeFilterHome" Section="Input" Type="Time">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="timeFilterWork" Section="Input" Type="Time">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="timeTimeout" Section="Input" Type="Time">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="outputHome" Section="Output" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="outputWork" Section="Output" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="errorTimeoutWork" Section="Output" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="errorTimeoutHome" Section="Output" Type="Bool">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
              <Parameter Name="cylinder" Section="InOut" Type=""CTRL_Cylinder"">
                <StringAttribute Name="InterfaceFlags" Informative="true">S7_Visible</StringAttribute>
              </Parameter>
            </CallInfo>
          </Call>
          <Part Name="Coil" UId="36"/>
          <Part Name="Coil" UId="37"/>
          <Part Name="Coil" UId="38"/>
          <Part Name="Coil" UId="39"/>
        </Parts>
        <Wires>
          <Wire UId="50">
            <Powerrail/>
            <NameCon UId="34" Name="en"/>
            <NameCon UId="36" Name="in"/>
            <NameCon UId="37" Name="in"/>
            <NameCon UId="38" Name="in"/>
            <NameCon UId="39" Name="in"/>
          </Wire>
          <Wire UId="51">
            <OpenCon UId="40"/>
            <NameCon UId="34" Name="name"/>
          </Wire>
          <Wire UId="52">
            <IdentCon UId="21"/>
            <NameCon UId="34" Name="enableHome"/>
          </Wire>
          <Wire UId="53">
            <IdentCon UId="22"/>
            <NameCon UId="34" Name="enableWork"/>
          </Wire>
          <Wire UId="54">
            <OpenCon UId="41"/>
            <NameCon UId="34" Name="doorOpen"/>
          </Wire>
          <Wire UId="55">
            <OpenCon UId="42"/>
            <NameCon UId="34" Name="manualMode"/>
          </Wire>
          <Wire UId="56">
            <OpenCon UId="43"/>
            <NameCon UId="34" Name="reset"/>
          </Wire>
          <Wire UId="57">
            <IdentCon UId="23"/>
            <NameCon UId="34" Name="iHome"/>
          </Wire>
          <Wire UId="58">
            <IdentCon UId="24"/>
            <NameCon UId="34" Name="iWork"/>
          </Wire>
          <Wire UId="59">
            <IdentCon UId="25"/>
            <NameCon UId="34" Name="orderHome"/>
          </Wire>
          <Wire UId="60">
            <IdentCon UId="26"/>
            <NameCon UId="34" Name="orderWork"/>
          </Wire>
          <Wire UId="61">
            <OpenCon UId="44"/>
            <NameCon UId="34" Name="doesNotRetainOutput"/>
          </Wire>
          <Wire UId="62">
            <OpenCon UId="45"/>
            <NameCon UId="34" Name="timeFilterHome"/>
          </Wire>
          <Wire UId="63">
            <OpenCon UId="46"/>
            <NameCon UId="34" Name="timeFilterWork"/>
          </Wire>
          <Wire UId="64">
            <OpenCon UId="47"/>
            <NameCon UId="34" Name="timeTimeout"/>
          </Wire>
          <Wire UId="65">
            <IdentCon UId="27"/>
            <NameCon UId="34" Name="Cylinder"/>
          </Wire>
          <Wire UId="66">
            <NameCon UId="34" Name="outputHome"/>
            <IdentCon UId="28"/>
          </Wire>
          <Wire UId="67">
            <NameCon UId="34" Name="outputWork"/>
            <IdentCon UId="29"/>
          </Wire>
          <Wire UId="68">
            <NameCon UId="34" Name="errorTimeoutWork"/>
            <OpenCon UId="48"/>
          </Wire>
          <Wire UId="69">
            <NameCon UId="34" Name="errorTimeoutHome"/>
            <OpenCon UId="49"/>
          </Wire>
          <Wire UId="70">
            <IdentCon UId="30"/>
            <NameCon UId="36" Name="operand"/>
          </Wire>
          <Wire UId="71">
            <IdentCon UId="31"/>
            <NameCon UId="37" Name="operand"/>
          </Wire>
          <Wire UId="72">
            <IdentCon UId="32"/>
            <NameCon UId="38" Name="operand"/>
          </Wire>
          <Wire UId="73">
            <IdentCon UId="33"/>
            <NameCon UId="39" Name="operand"/>
          </Wire>
        </Wires>
      </FlgNet>
    </NetworkSource>
    <ProgrammingLanguage>LAD</ProgrammingLanguage>
  </AttributeList>
  <ObjectList>
    <MultilingualText ID="4" CompositionName="Comment">
      <ObjectList>
        <MultilingualTextItem ID="5" CompositionName="Items">
          <AttributeList>
            <Culture>en-US</Culture>
            <Text/>
          </AttributeList>
        </MultilingualTextItem>
      </ObjectList>
    </MultilingualText>
    <MultilingualText ID="6" CompositionName="Title">
      <ObjectList>
        <MultilingualTextItem ID="7" CompositionName="Items">
          <AttributeList>
            <Culture>en-US</Culture>
            <Text>Open</Text>
          </AttributeList>
        </MultilingualTextItem>
      </ObjectList>
    </MultilingualText>
  </ObjectList>
</SW.Blocks.CompileUnit>
```


Fim

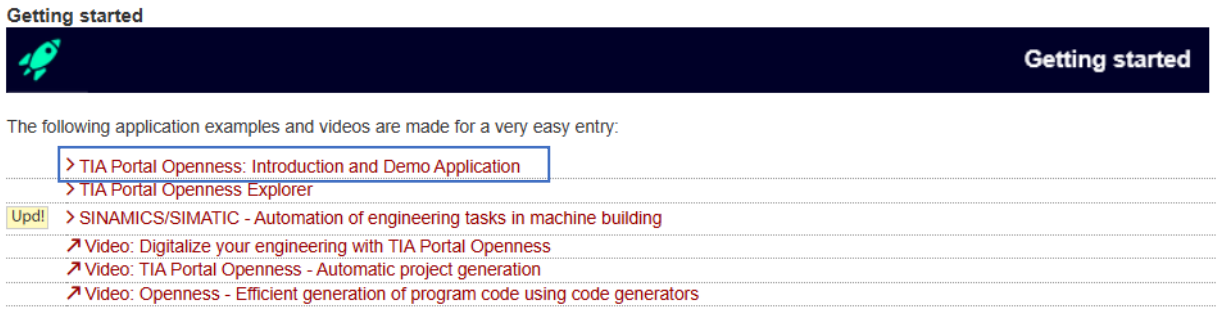
5 – Código XML de uma Faceplate de Cilindro escrito pela função [writeFaceplateInstancesCylinder()]

```
<Hmi.Screen.FaceplateInstance ID="4" CompositionName="ScreenItems">
  <AttributeList>
    <FaceplateTypeName>HMI@$@Cylinder V 0.1.36</FaceplateTypeName>
    <Height>137</Height>
    <Left>490</Left>
    <ObjectName>Open - 1</ObjectName>
    <Resizing>FixedSize</Resizing>
    <TabIndex>1</TabIndex>
    <Top>200</Top>
    <Width>249</Width>
  </AttributeList>
  <ObjectList>
    <Hmi.Screen.InterfacePropertySimple ID="5" CompositionName="Interface">
      <AttributeList>
        <Name>Cylinder</Name>
      </AttributeList>
      <ObjectList>
        <Hmi.Dynamic.TagConnectionDynamic ID="6" CompositionName="Dynamic">
          <AttributeList>
            <Indirect>false</Indirect>
          </AttributeList>
          <LinkList>
            <Tag TargetID="@OpenLink">
              <Name>Open</Name>
            </Tag>
          </LinkList>
        </Hmi.Dynamic.TagConnectionDynamic>
      </ObjectList>
    </Hmi.Screen.InterfacePropertySimple>
  </ObjectList>
</Hmi.Screen.FaceplateInstance>
```

Softwares adicionais

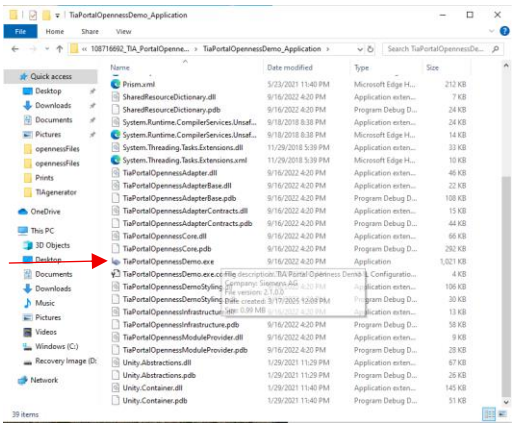
Obtenção de código de XML de Objetos

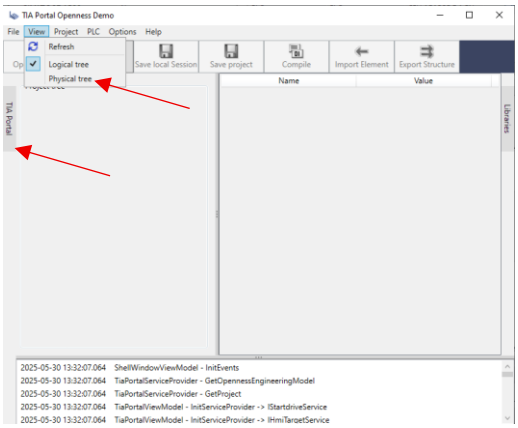
Uma forma fácil de obter o ficheiro em XML de um objeto é usando a Demo disponibilizada no site da Siemens na página do TIA Portal Openness.

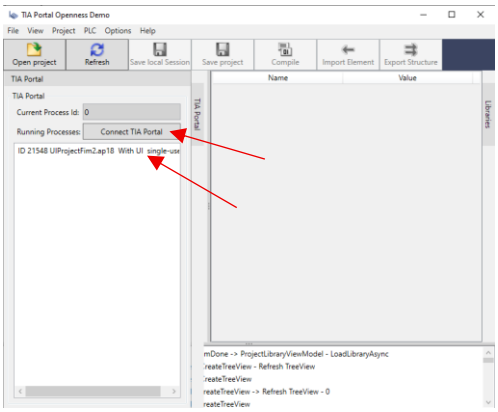


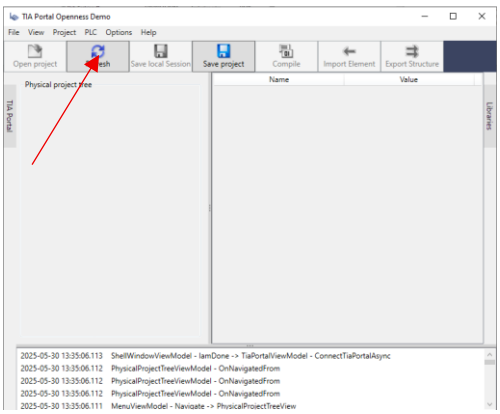
Esta aplicação permite aceder diretamente a objetos no TIA Portal e exportá-los em formato XML

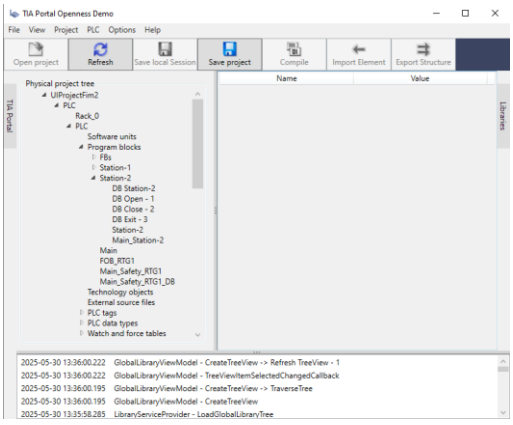
Inicialização:







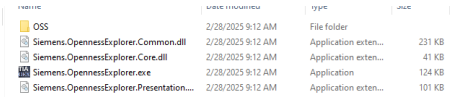


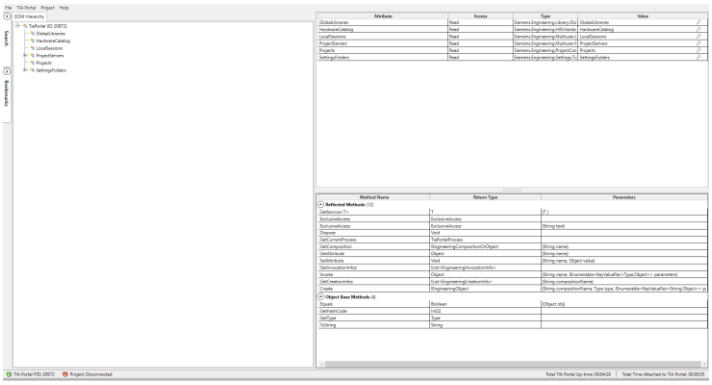


Para exportar, selecionar o objeto, compila-lo e exportá-lo. Alguns objetos, para serem compilados, precisam que outros objetos estejam compilados.

Por exemplo, para compilar uma FC, é preciso que os DB’s que ela usa também estejam compilados.

Openess Explorer





O Openness Explorer é uma boa ferramenta para entender as hierarquias dos objetos, Devices e Device Items. Torna mais legível a forma como são organizados os objetos a um nível mais superficial