Biblioteca Tia Portal Openness

Índice

[Utilização da Biblioteca 3](#_Toc200359923)

[Pasta de Ficheiros 3](#_Toc200359924)

[Adicionar a Biblioteca às Referências: 3](#_Toc200359925)

[Exemplo de Código da aplicação. 4](#_Toc200359926)

[1 - Inicio 4](#_Toc200359927)

[2 - Criar/Abrir um projeto 4](#_Toc200359928)

[3 – Criar PLC e HMI 5](#_Toc200359929)

[4 – Ler Lista de I/O’s em Excel 6](#_Toc200359930)

[5-Importar Global Library e objetos 8](#_Toc200359931)

[6- Criação de Pastas e Objetos no PLC e HMI 8](#_Toc200359932)

[Biblioteca de Funções 10](#_Toc200359933)

[1 – GETS E SETS 11](#_Toc200359934)

[2 – ABERTURA DO TIA PORTAL E DO PROJETO 11](#_Toc200359935)

[3- CRIAR E ENCONTRAR HMI’S E PLC’S NO PROJETO 12](#_Toc200359936)

[4- OBTENÇÃO DO SOFTWARE DOS DEVICES 12](#_Toc200359937)

[5- CONEXÃO E ATRIBUIÇÃO DE IP’S 13](#_Toc200359938)

[6- FUNÇÕES BASE PARA CRIAÇÃO DE PASTAS, IMPORTAR GLOBAL LIBRARIES E IMPORTAR OBJETOS DA GLOBAL LIBRARY 14](#_Toc200359939)

[7- FUNÇÕES DE PASTAS DE SCREENS E TEMPLATES 15](#_Toc200359940)

[8 – EXPORTAÇÃO/IMPORTAÇÃO DE FICHEIROS XML 16](#_Toc200359941)

[9 – FUNÇÕES DE AUXILIO À ESCRITA DE DOCUMENTOS EM XML 18](#_Toc200359942)

[10 – ESCRITA DE DOCUMENTO XML DE UMA DB DE CILINDROS 19](#_Toc200359943)

[11 - ESCRITA DE DOCUMENTO XML DE UMA FC DE CILINDROS 20](#_Toc200359944)

[12 - ESCRITA DE DOCUMENTO XML DE UMA SCREEN COM CILINDROS 21](#_Toc200359945)

[13 - ESCRITA DO DOCUMENTO XML DE UM MAIN BLOCK DE CILINDROS 21](#_Toc200359946)

[14 - ESCRITA DO DOCUMENTO XML DE UMA TAG TABLE DE HMI 21](#_Toc200359947)

[15 – IMPORTS DO EXCEL 23](#_Toc200359948)

[Expansão da Biblioteca 24](#_Toc200359949)

[1 - Funções a Alterar ou Criar: 24](#_Toc200359950)

[2 - DataBlocks: 24](#_Toc200359951)

[3 - FC’s 26](#_Toc200359952)

[4 - Screens 28](#_Toc200359953)

[5 - Tag Table da HMI 29](#_Toc200359954)

[6 - DataBlocks de instâncias. 30](#_Toc200359955)

[Excertos de Código em XML 31](#_Toc200359956)

[1 – Código em XML escrito pela função writeXmlDocumentInfo() 31](#_Toc200359957)

[2 – Código em XML escrito peça função writeXmlDocumentInfoTagTable() 31](#_Toc200359958)

[3 - Código em XML de apenas um membro da função 31](#_Toc200359959)

[4 – Código XML de uma Network de Cilindro escrito pela função 33](#_Toc200359960)

[5 – Código XML de uma Faceplate de Cilindro escrito pela função 34](#_Toc200359961)

[Softwares adicionais 34](#_Toc200359962)

[Obtenção de código de XML de Objetos 34](#_Toc200359963)

[Openess Explorer 36](#_Toc200359964)

# 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.

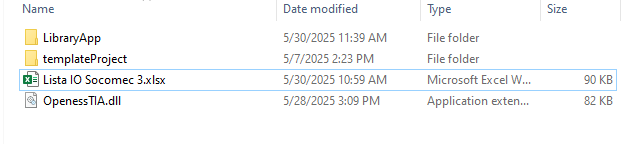
## Pasta de Ficheiros

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

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

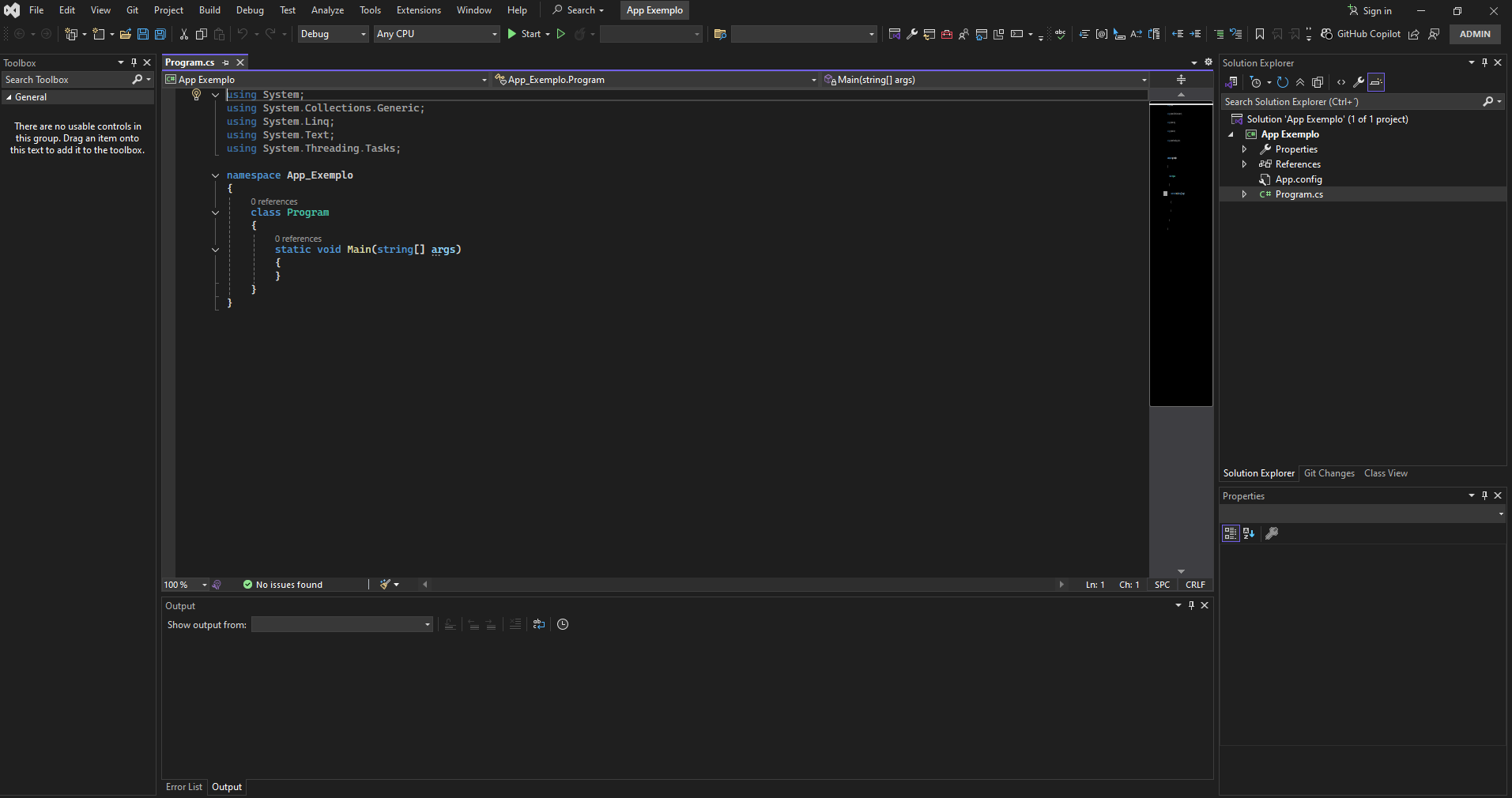
“LibraryApp” que contém a global Library a ser importada.

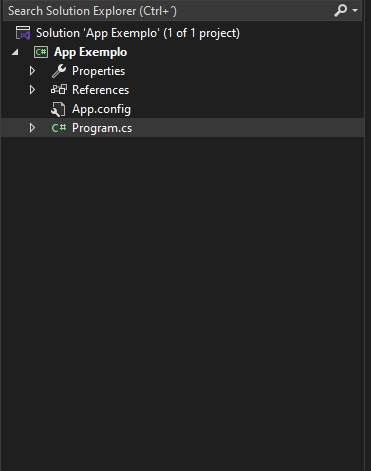
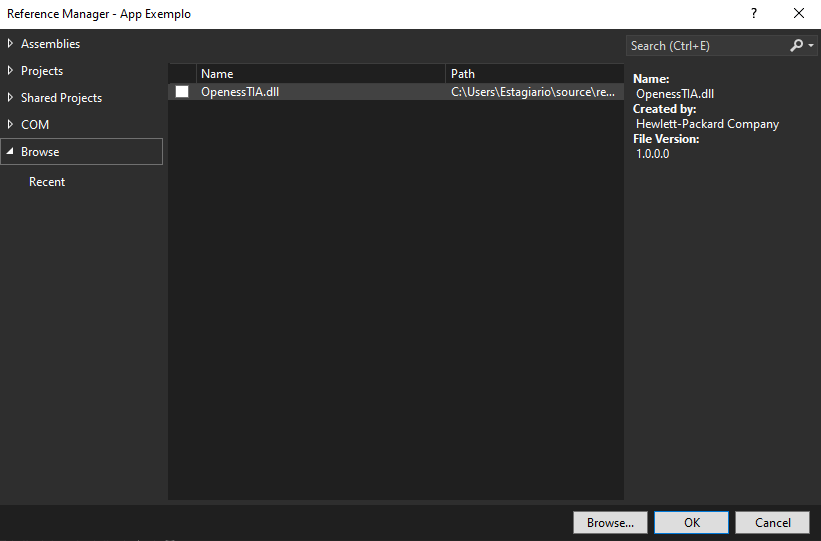
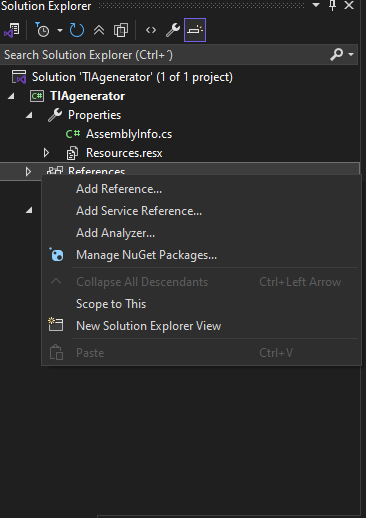
“OpenessTIA.dll” sendo esta a biblioteca compilada a utilizar

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:



Com a variável inicializada, deverão ser usadas duas funções:

Estas funções permitem definir o caminho da [pasta de ficheiros](#AAPasta) necessária e abrir o TIA Portal V18.

A variável “filesPath” deve ser uma string com a localização no Windows da [pasta de ficheiros](#AAPasta).

### 2 - Criar/Abrir um projeto

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



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](#AAPasta) 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](#AAPasta) 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:

[](#createDevicePlc)

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()](#createDevicePlc)

[createDeviceHmi()](#createDeviceHmi)

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



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()](#findDevices)

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

Para isso usar as funções:



🡨Não Unified

🡨Unified

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

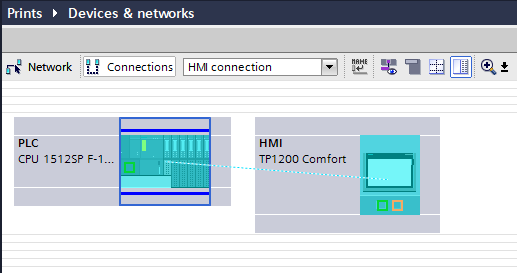
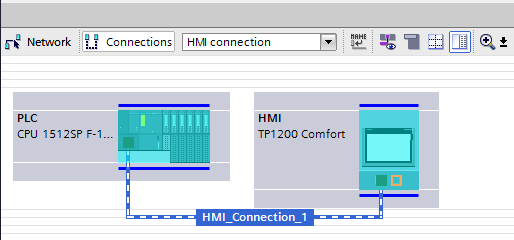
[getPlcSoftware()](#getPlcSoftware)

[getHmiTarget()](#getHmiTarget) 🡨Não Unified

[getHmiSoftware()](#getHmiSoftware) 🡨 Unified

Também é necessário conectar os devices manualmente. Poderá ser usada a função [createConnectionPrompt()](#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()](#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:

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

[writeXmlPlcTagTableIO()](#writeXmlPlcTagTableIO)

[CountCylinders()](#CountCylinders)

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



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

[importPlcModules()](#importPlcModules)

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



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

[importPlcTagTable()](#importPlcTagTable)

#### Formato do Ficheiro Excel

O ficheiro em Excel deve seguir um formato específico:

-A lista deve começar na linha 4.

Entradas:

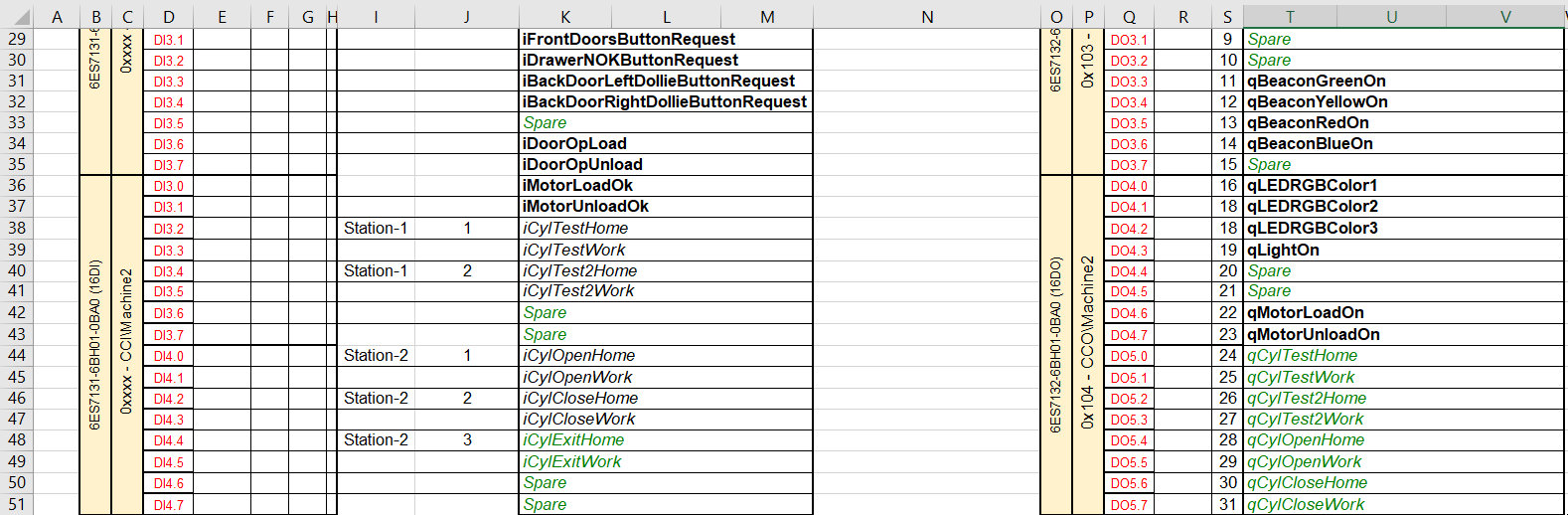
-A Coluna D nunca deve estar vazia, mesmo que os endereços não correspondam aos corretos.

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:

### 5-Importar Global Library e objetos

Para importar a Global Library, usar a função:



Com esta função é possível importar a Global Library da [pasta de ficheiros](#AAPasta) “LibraryApp”.

[importGlobalLibrary()](#importGlobalLibrary)

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



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

[getUdtFromLibrary()](#getUdtFromLibrary)

[getFbFromLibrary()](#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:



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

[divideLists()](#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:



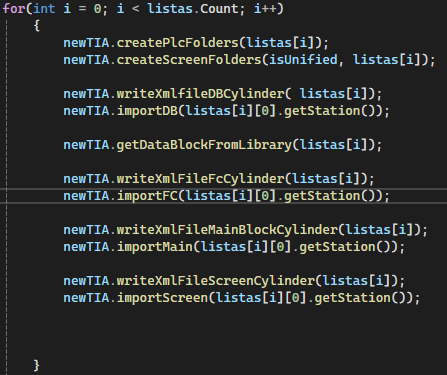
Com estas funções, serão criadas pastas correspondentes às estações que foram anteriormente lidas do ficheiro em Excel.

[createPlcFolders()](#createPlcFolders)

[createScreenFolder()](#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



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()](#writeXmlfileDBCylinder)

[writeXmlFileFcCylinder()](#writeXmlFileFcCylinder)

[writeXmlFileMainBlockCylinder()](#writeXmlFileMainBlockCylinder)

[writeXmlFileScreenCylinder()](#writeXmlFileScreenCylinder)

[importDB()](#importDB)

[importFC()](#importFC)

[importMain()](#importMain)

[importScreen()](#importScreen)

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



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

[writeXmlHmiTagTableCylinder()](#writeXmlHmiTagTableCylinder)

[importHmiTagTable()](#importHmiTagTable)

Para importar as Templates das Screens, usar a função:



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

[getAllTemplatesFromLibrary()](#getAllTemplatesFromLibrary)

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



Esta função guarda o projeto criado.

[saveProject()](#saveProject)

## Biblioteca de Funções

Regiões:

1. [Gets e Sets](#_1_–_GETS)
2. [Abertura do TIA Portal e do Projeto](#_2_–_ABERTURA)
3. [Criar e Encontrar Hmi’s e Plc’s no Projeto](#_3-_CRIAR_E)
4. [Obtenção do software dos devices](#_4-_OBTENÇÃO_DO)
5. [Conexão e atribuição de IP’s](#_4-_OBTENÇÃO_DO)
6. [Funções base para criação de pastas, Importar Global Library e Importar Objetos da Global Library](#_6-_FUNÇÕES_BASE)
7. [Funções de pastas de screens e templates](#_7-_FUNÇÕES_DE)
8. [Exportação/Importação de ficheiros XML](#_8_–_EXPORTAÇÃO/IMPORTAÇÃO)
9. [Funções de Auxílio à escrita de documentos em XML](#_9_–_FUNÇÕES)
10. [Escrita de documento XML de uma DB de Cilindros](#_10_–_ESCRITA)
11. [Escrita de documento XML de uma FC de Cilindros](#_11_-_ESCRITA)
12. [Escrita de documento XML de uma Screen de Cilindros](#_12_-_ESCRITA)
13. [Escrita do Documento XML de um Main Block de Cilindros](#_13_-_ESCRITA)
14. [Escrita de documento XML de uma TagTable de Cilindros de HMI](#_14_-_ESCRITA)
15. [Imports de Excel](#_15_–_IMPORTS)

### 1 – GETS E SETS



Retorna o número de DataBlocks de Cilindro.



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



Define o caminho no Windows fornecido por stringFilePath para a [pasta de ficheiros](#AAPasta) 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

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

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



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](#setFilePath)().



Abre a janela “Project View” do TIA Portal.



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



Guarda o projeto atual.

### 3- CRIAR E ENCONTRAR HMI’S E PLC’S NO PROJETO



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

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



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.



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

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

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



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

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



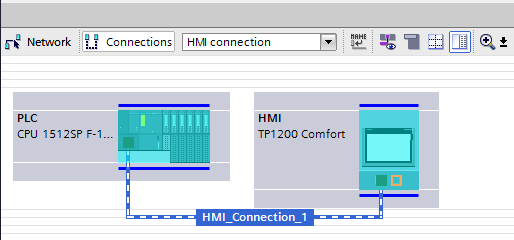
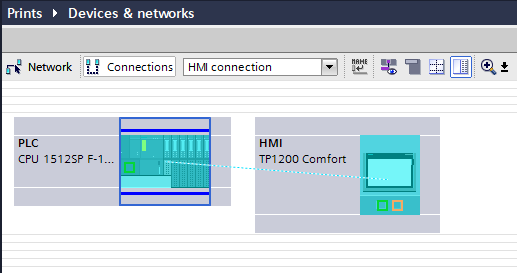
Obtém o Software de uma HMI Unified, necessário para a configuração e programação do mesmo.

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

### 5- CONEXÃO E ATRIBUIÇÃO DE IP’S



Prompt na consola pedindo ao usuário para conectar os Devices





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()](#getPlcSoftware) previamente



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

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



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

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

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



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

Deve ser executada a função [getPlcSoftware()](#getPlcSoftware)previamente.



Conta o número de Function totais no TIA Portal

Deve ser executada a função [getPlcSoftware()](#getPlcSoftware)previamente.

Abre a Biblioteca “libraryApp” presente na [pasta de ficheiros](#AAPasta)



Cria as pastas correspondentes às estações do projeto.

Deve ser executada a função [getPlcSoftware()](#getPlcSoftware)previamente.



Importa todas as UDTs da Global Library Importada.

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

****

Importa todas as FB’s da Global Library Importada.

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



Importa os DataBlocks a partir de uma lista de Cilindros.

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

****

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

****

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

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



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()](#getDataBlockFromLibrary) e é especifica para a utilização desta.

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

****

Importa apenas uma FB.

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

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



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

Deve ser executada a função [getHmiTarget()](#getHmiTarget) previamente.



Importa todas as Screen Templates da Global Library.

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



Importa uma Screen Template da Global Library.

É usada pela função [getAllTemplatesFromLibary().](#getAllTemplatesFromLibrary)

Devem ser executadas as funções [getHmiTarget()](#getHmiTarget)e [importGlobalLibrary()](#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.



Importa uma FB da [pasta de ficheiros](#AAPasta) a partir do ficheiro FB\_write.xml e insere na estação (pasta) com o nome fornecido por stationName.

Deve ser executada a função [getPlcSoftware()](#getPlcSoftware) previamente e deve existir um ficheiro “FB\_write.xml” com o formato correto.



Importa uma FC da [pasta de ficheiros](#AAPasta) 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()](#getPlcSoftware)previamente e deve existir um ficheiro “FC\_write.xml” com o formato correto.



Importa uma DB da [pasta de ficheiros](#AAPasta) 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()](#getPlcSoftware) previamente e deve existir um ficheiro “DB\_write.xml” com o formato correto.



Importa uma Main da [pasta de ficheiros](#AAPasta) 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()](#getPlcSoftware) previamente e deve existir um ficheiro “Main\_write.xml” com o formato correto.



Importa uma Screen da [pasta Inicial](#AAPasta) 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()](#getHmiTarget) previamente e deve existir um ficheiro “Screen\_write.xml” com o formato correto.



Importa uma Tag Table de HMI da [pasta de ficheiros](#AAPasta) 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()](#getHmiTarget)previamente e deve existir um ficheiro “HmiTagTable\_write.xml” com o formato correto.



Importa uma Tag Table de PLC da [pasta de ficheiros](#AAPasta) a partir do ficheiro PlcTagTable\_write.xml.

Deve ser executada a função [getHmiTaget()](#getHmiTarget)previamente e deve existir um ficheiro “PlcTagTable\_write.xml” com o formato correto.



Exporta uma Screen para a [pasta de ficheiros](#AAPasta) com o nome Screen.xml

Deve ser executada a função [getHmiTaget()](#getHmiTarget)previamente.



Exporta uma Tag Table de Hmi para a [pasta de ficheiros](#AAPasta) com o nome TagTable.xml

Deve ser executada a função [getHmiTaget()](#getHmiTarget)previamente.



Exporta uma Tag Table de PLC para a [pasta de ficheiros](#AAPasta) com o nome TagTable.xml

Deve ser executada a função [getPlcSoftware()](#getPlcSoftware) previamente.

### 9 – FUNÇÕES DE AUXILIO À ESCRITA DE DOCUMENTOS EM XML



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

Esta função escreve [este código](#DocumentInfoXML).

Usada por outras funções de escrita de XML.



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

Esta função escreve [este código](#DocumentInfoXMLTagTable).

Usada por outras funções de escrita de XML.



Escreve a seguinte estrutura no ficheiro fornecido por writer:

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

Usada por outras funções de escrita de XML.



Escreve a seguinte estrutura no ficheiro fornecido por writer:

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

Usada por outras funções de escrita de XML.



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.



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



Escreve a seguinte estrutura no ficheiro fornecido por writer:

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

Usada exclusivamente na função [writeXmlInterfaceDbCylinder().](#writeXmlInterfaceDbCylinder)



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

Usada exclusivamente na função [writeXmlInterfaceDbCylinder().](#writeXmlfileDBCylinder)



Escreve o ficheiro em XML de um DataBlock com os cilindros fornecidos pela listaCilindros para depois ser importado na [pasta de ficheiros](#AAPasta) com o nome DB\_write.xml.

### 11 - ESCRITA DE DOCUMENTO XML DE UMA FC DE CILINDROS



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().](#writeXmlWires)



Escreve as Wires da FC no ficheiro fornecido por writer.

Usada exclusivamente na função [writeXmlNetorksFcCylinder().](#writeXmlNetworksFcCylinder)



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().](#writeXmlNetworksFcCylinder)



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

Usada exclusivamente na função [writeXmlFileFcCylinder().](#writeXmlFileFcCylinder)



Escreve a Interface da FC no ficheiro fornecido por writer.

Usada exclusivamente na função [writeXmlFileFcCylinder().](#writeXmlFileFcCylinder)



Escreve o ficheiro em XML de uma FC com os cilindros fornecidos pela listaCilindros para depois ser importado na [pasta de ficheiros](#AAPasta) com o nome FC\_write.xml.

### 12 - ESCRITA DE DOCUMENTO XML DE UMA SCREEN COM CILINDROS



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

Usada exclusivamente pela função [writeXmlFileScreenCylinder().](#writeXmlFileScreenCylinder)



Escreve o ficheiro em XML de uma Screen com os cilindros fornecidos pela listaCilindros para depois ser importado na [pasta de ficheiros](#AAPasta) com o nome Screen\_write.xml.

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



Escreve o ficheiro em XML de um Main Block com a FC já criada anteriormente na [pasta de ficheiros](#AAPasta) com o nome Main\_write.xml.

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



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()](#intToHex) para escrever o ID no formato hexadecimal.

Usada por outras funções de escrita de XML de Tag Table de Hmi.



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()](#intToHex) para escrever o ID no formato hexadecimal

Usada por outras funções de escrita de XML de Tag Table de HMI.

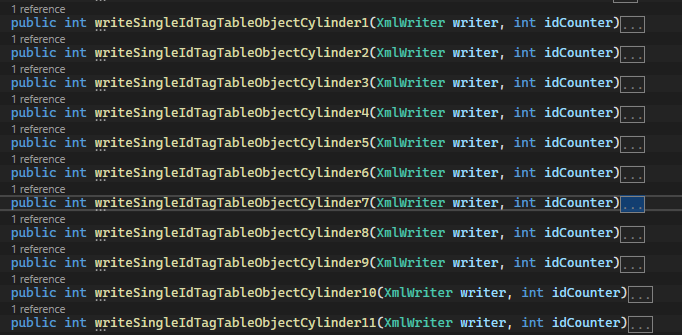


Escreve os membros da TagTable da HMI de acordo com a listaCilindros no ficheiro fornecido por writer.

Usada exclusivamente pela função [writeXmlHmiTagTableCylinder().](#writeXmlHmiTagTableCylinder)



Escreve o ficheiro em XML de uma Tag Table com os cilindros fornecidos pela listaCilindros para depois ser importado na [pasta de ficheiros](#AAPasta) 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().](#CountCylinders)



Lê um ficheiro de Excel na [pasta de ficheiros](#AAPasta) 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](#AAPasta) com o nome FC\_write.xml. A função escreve a o ficheiro em XML a partir do ficheiro em Excel na [pasta de ficheiros](#AAPasta) com o nome especificado por fileName.

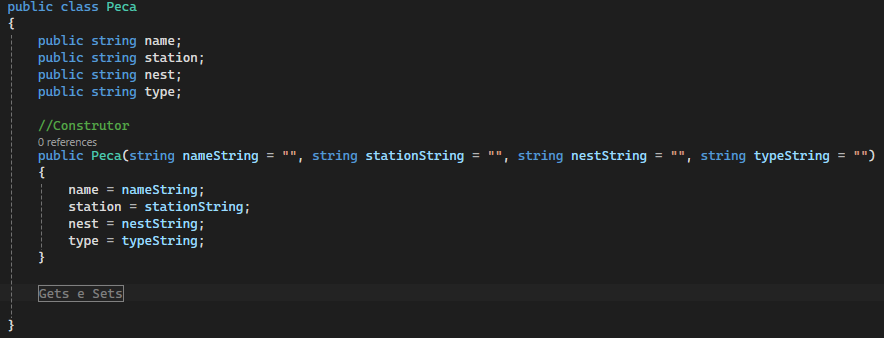


Lê o ficheiro em Excel na [pasta de ficheiros](#AAPasta) com o nome especificado por fileName e importa as cartas/modules.



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().](#importPlcModules)

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

## 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()](#writeXmlfileDBCylinder)

[writeXmlFileFcCylinder()](#writeXmlFileFcCylinder)

[writeXmlFileScreenCylinder()](#writeXmlFileScreenCylinder)

[writeXmlFileHmiTagTableCylinder()](#writeXmlHmiTagTableCylinder)

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()](#writeXmlInterfaceDbCylinder) mas para outra peça e inicializa-la em [writeXmlfileDBCylinder()](#writeXmlfileDBCylinder);

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

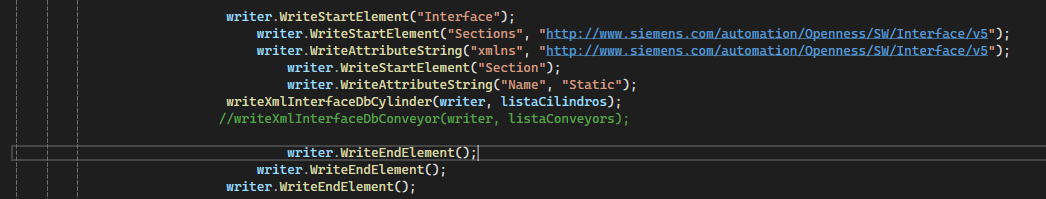
É necessário criar uma nova função pois as Function Blocks e variáveis de outras peças seriam completamente diferentes das FB’s e variáveis de Cilindros.

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:

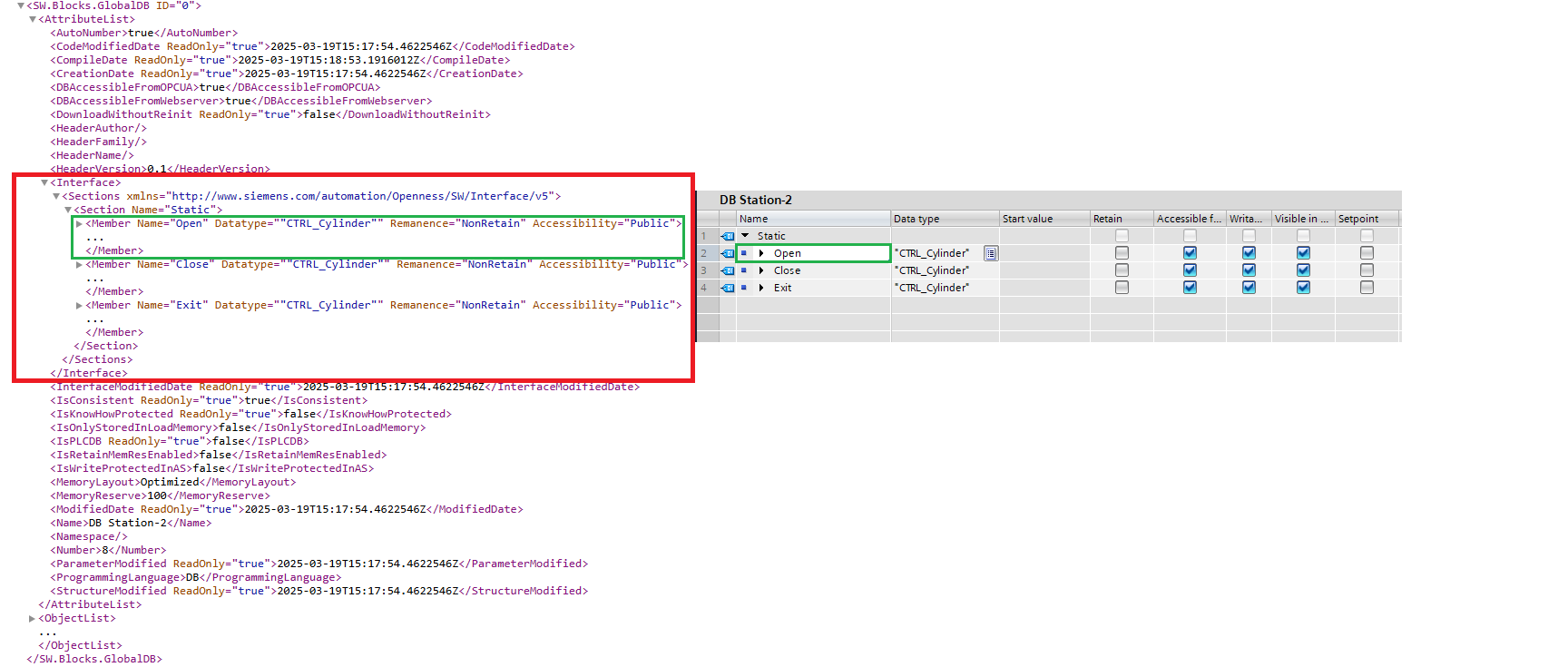
DB (Nome da estação)



Exemplo de uso de uma nova função:

****

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”.

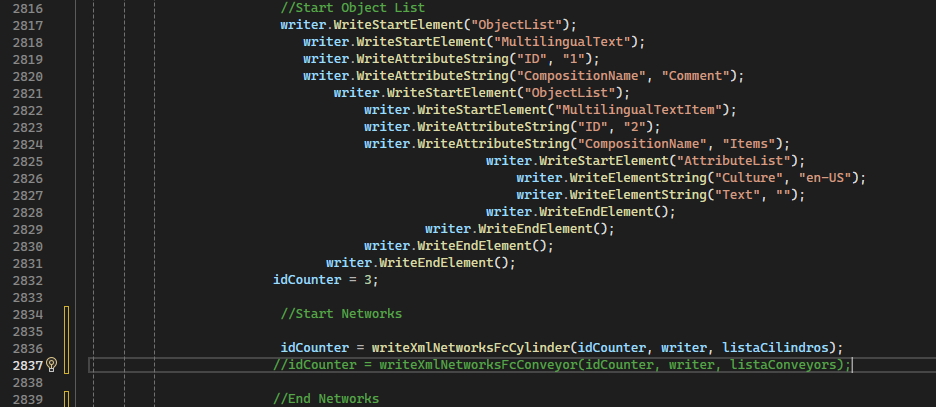
Exemplo de Código XML de membros de um DataBlock

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:](#CodigXMLMembroDB)

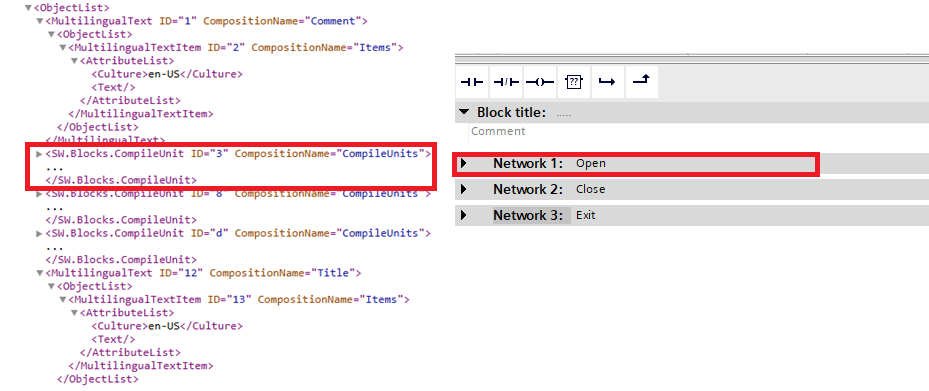
### 3 - FC’s

No caso das Fc’s seria o mesmo. Seria Necessário alterar as funções [writeXmlWiresCylinder()](#writeXmlWires) e [writeXmlNetorksFcCylinder()](#writeXmlNetworksFcCylinder) ou criar funções novas.



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

Código XML das Networks de uma FC



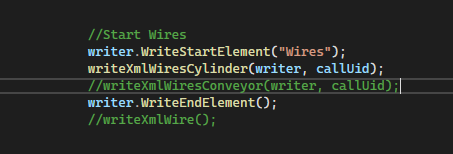
Os ID’s são todos sequenciais mas têm de estar na forma Hexadecimal. Por isso deve ser usada a função [intToHex().](#intToHex)

Exemplo de utilização de ID’s:

[Código XML de uma Network de Cilindro:](#CodigoXMLNetworkFCCilindro)

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

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



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



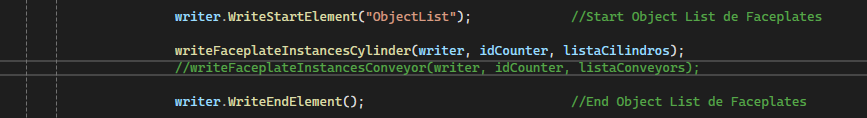
#### 2 - Ligação com DB

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.

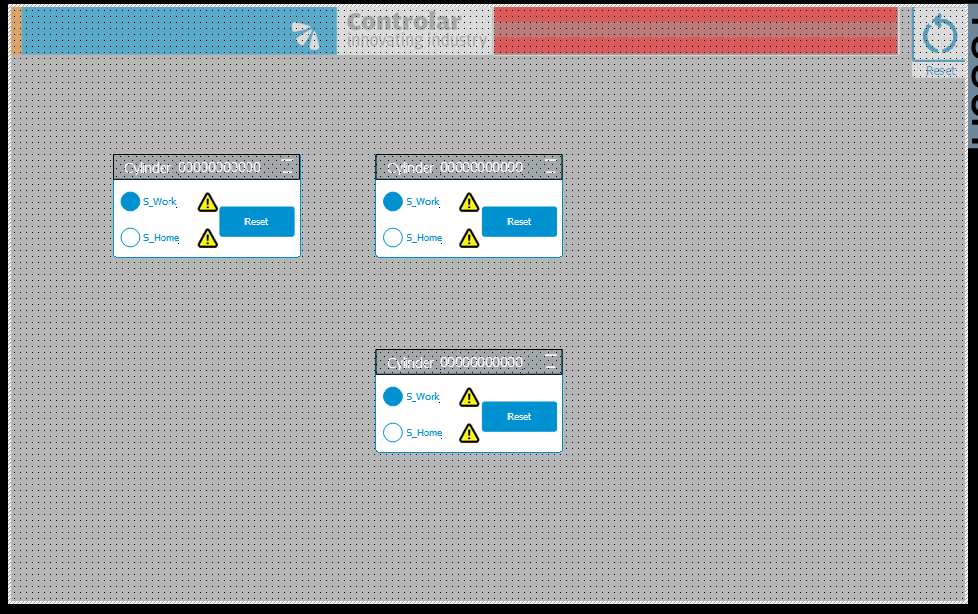
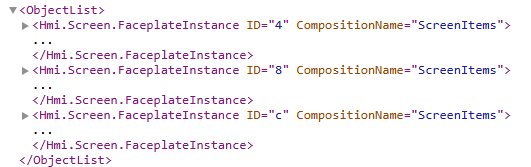
Para a posterior importação funcionar estas DB’s e FB’s devem já existir no projeto.

### 4 - Screens

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



Também usaria um sistema de separação de peças por tipo.



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



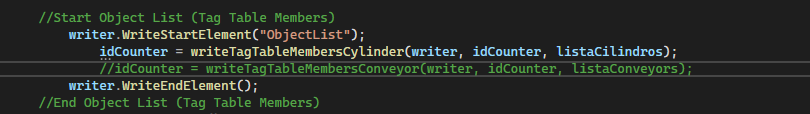
[Código em XML de Faceplates de Cilindro:](#CodigoXmlFaceplates)

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.

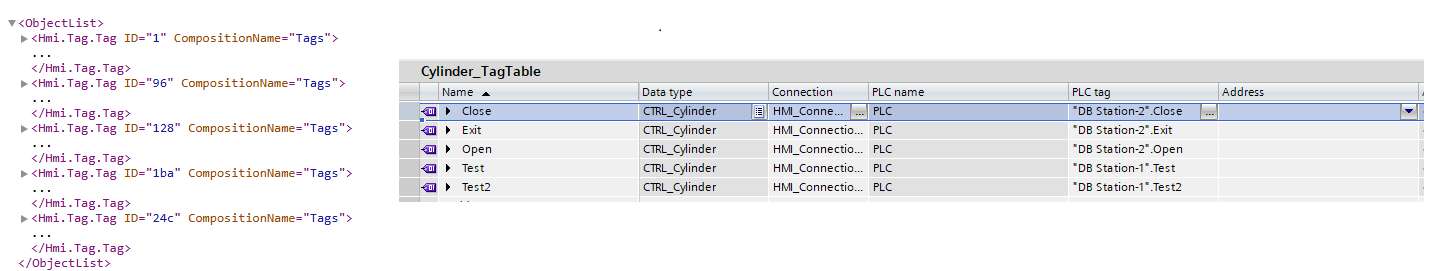
### 5 - Tag Table da HMI

Sem dúvida a parte mais difícil.

Será necessário criar funções semelhantes a [writeTagTableMembersCylinder()](#writeXmlHmiTagTableCylinder)



Também usaria um sistema de separação de peças por tipo.



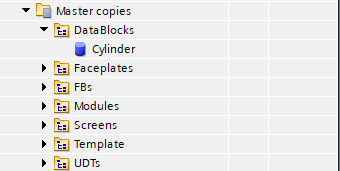
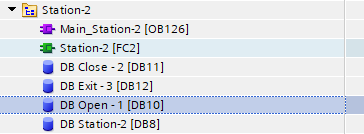
Esta é a parte mais difícil pois os membros da TagTable de HMI são muito extensos e pela necessidade de ver cada peça caso a caso pode ser um trabalho demorado.

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()](#getDataBlockFromLibrary) já está preparada para isso.

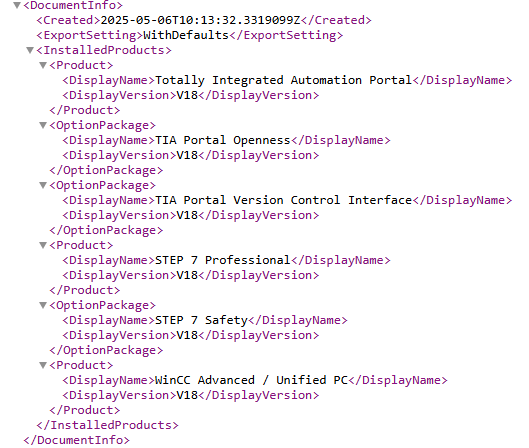
A função importa a DB de instância do tipo pretendido caso a DB exista como uma MasterCopy da Global Library na pasta “DataBlocks”. A DB na Global Library deve ter o mesmo nome do tipo da peça.



## Excertos de Código em XML

### 1 – Código em XML escrito pela função [writeXmlDocumentInfo()](#writeXmlDocumentInfo)

### 2 – Código em XML escrito peça função [writeXmlDocumentInfoTagTable()](#writeXmlDocumentInfoTagTable)

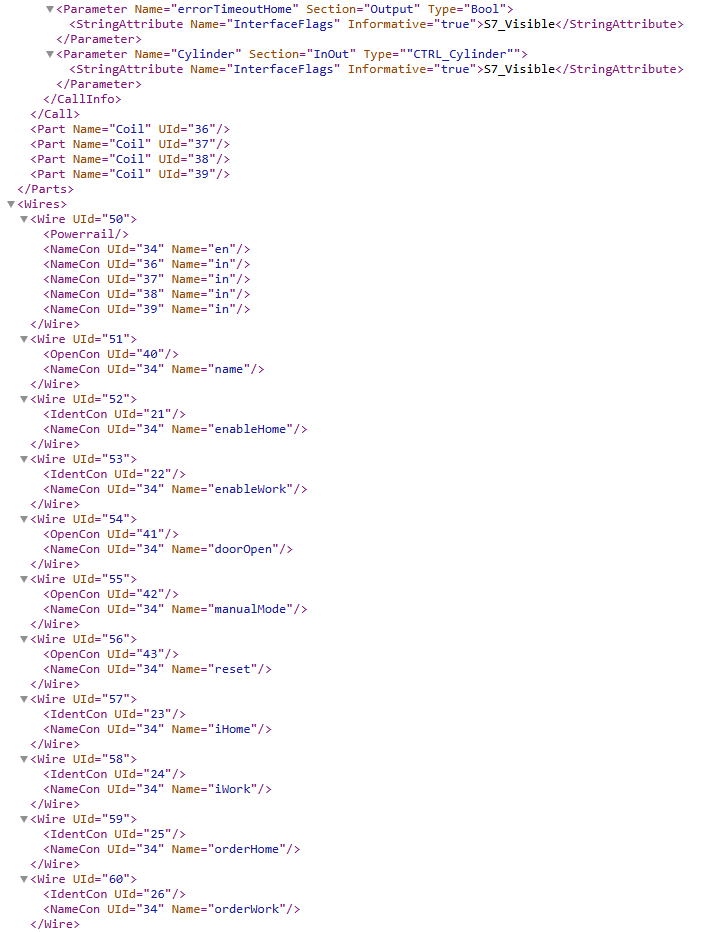


3 - Código em XML de apenas um membro da função [writeXmlInterfaceDbCylinder()](#writeXmlInterfaceDbCylinder)

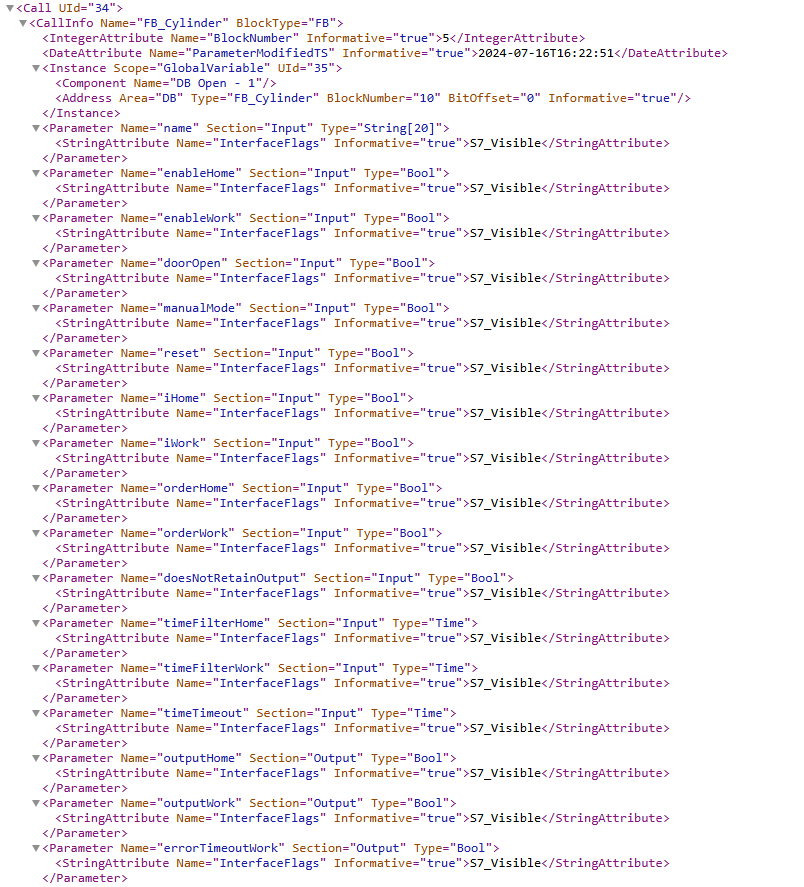




### 4 – Código XML de uma Network de Cilindro escrito pela função

[writeXmlNetworksFcCylinder()](#writeXmlNetworksFcCylinder)

Início



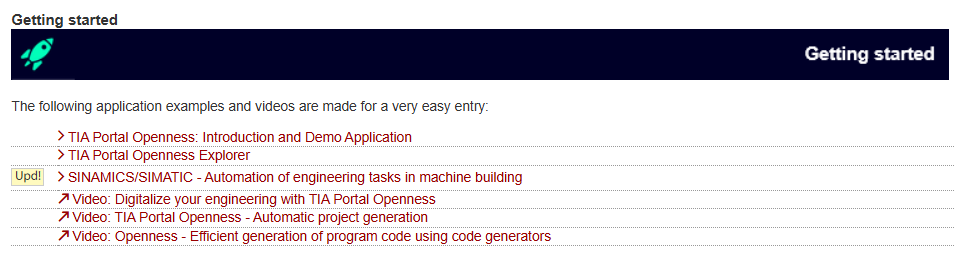
Fim

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

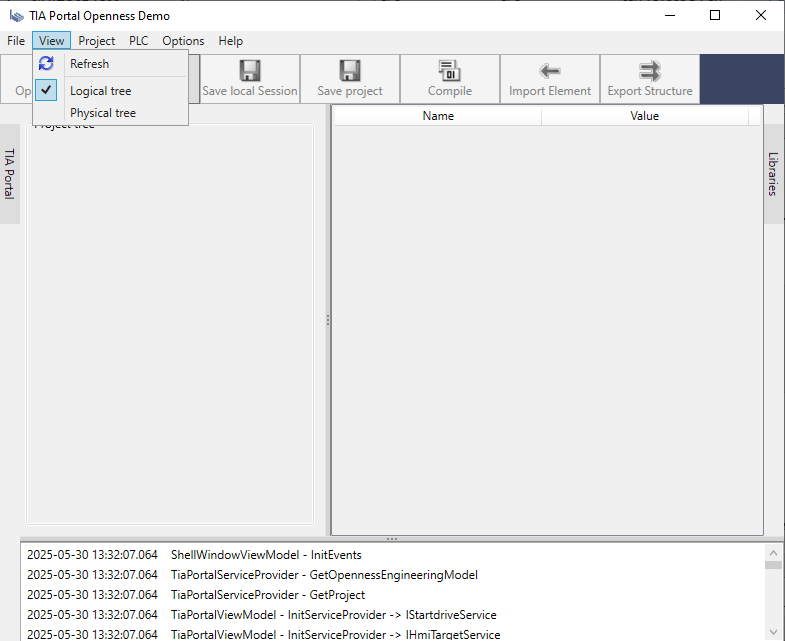
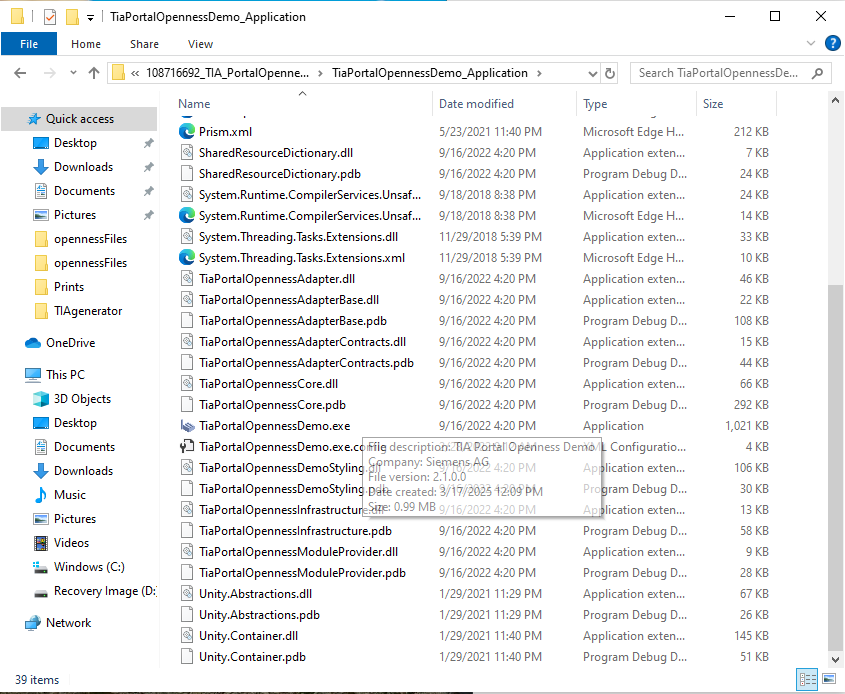


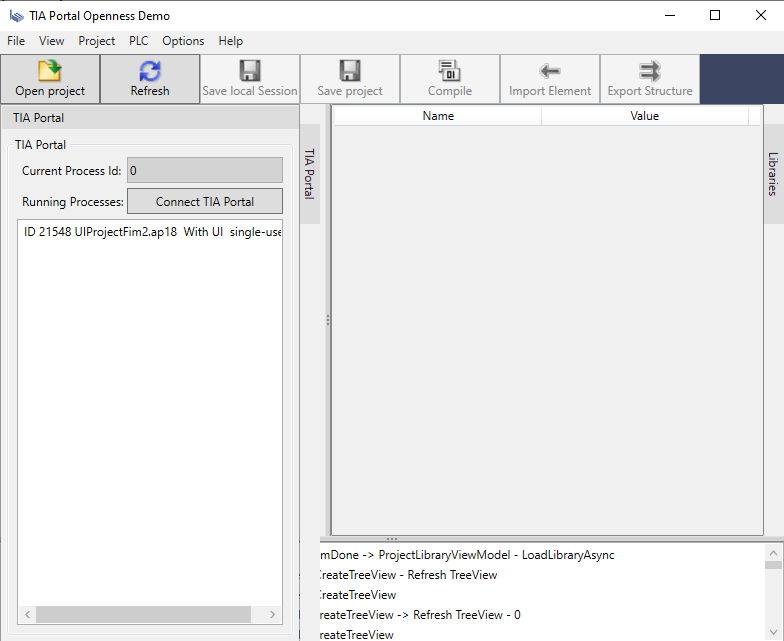
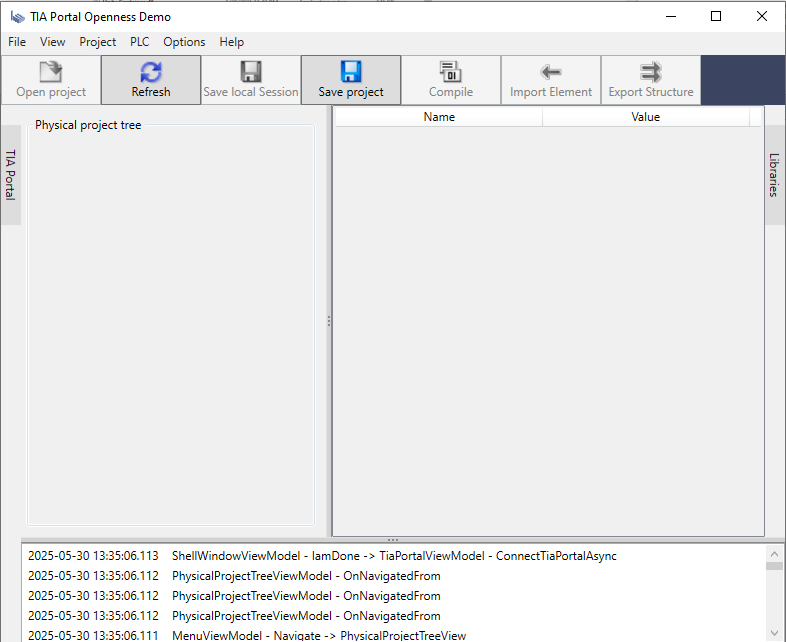
## Softwares adicionais

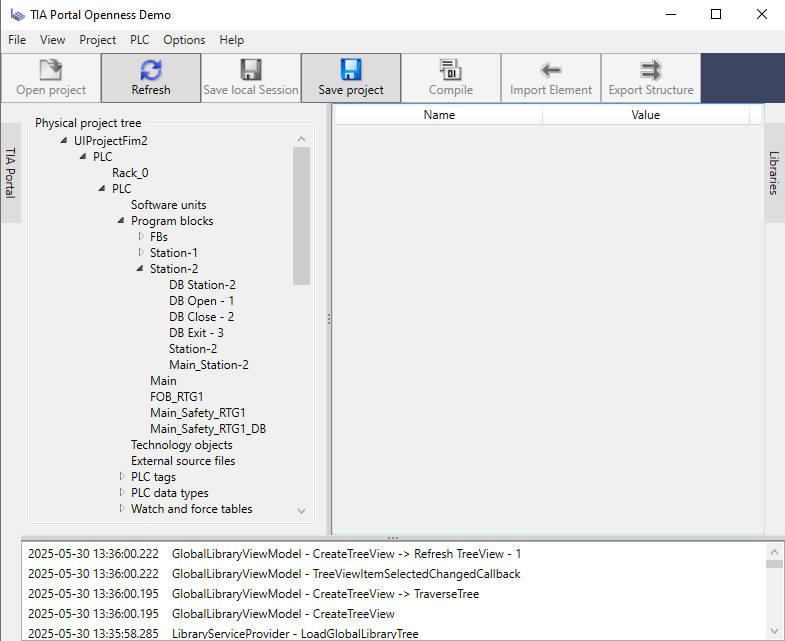
### Obtenção de código de XML de Objetos

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