

TIA PORTAL OPENNESS

(Funções)

Índice

1. ESTRUTURA.....	4
1.1. TIAPORTALOPENNESSDEMO.....	5
1.2. TIAOPENNESSHELPER	6
2. ORGANIZAÇÃO.....	8
3. CONEXÃO AO TIA PORTAL.....	9
4. TEMPLATES DE CÓDIGO.....	10
5. FUNCIONALIDADE “MAIN FOLDER FILES LIST”	11
6. FUNCIONALIDADE “PLC DB GENERATOR”	12
7. FUNCIONALIDADE “PLC TAGS”	14
8. FUNCIONALIDADE “GENERATE SYMBOLIC”	15
8.1. SELECT ROBOT	15
8.2. GENERATE SYMBOLIC	16
9. FUNCIONALIDADE “SEQUENCE GENERATOR”	18
9.1. CREATE EXCEL.....	19
10. FUNCIONALIDADE “RENAME PLC”	20
11. FUNCIONALIDADE “ROBOTLIST”	22
12. FUNCIONALIDADE “HARDWARE GENERATOR”	23
13. CLASSES IMPORTANTES.....	24
13.1. XLINTERFACE.....	24
13.2. PDFREADER.....	26
13.3. XMLPARSER.....	27
13.4. CACHEMANAGER	28
14. NOTAS FINAIS.....	29

Índice de Imagens

FIGURA 1 – ESTRUTURA DA SOLUTION "TIAPORTALOPENNESSDEMO"	4
FIGURA 2 - ESTRUTURA DO PROJETO "TIAPORTALOPENNESSDEMO"	5
FIGURA 3 - ESTRUTURA DO PROJECTO "TIAOPENNESSHELPER"	7
FIGURA 4 - ARQUITETURA MVVM	8
FIGURA 5 - FUNÇÃO "CONNECTTOTIA"	9
FIGURA 6 - FUNÇÃO "LOADPROJECTTREEVIEW"	9
FIGURA 7 - ESTRUTURA DE PASTAS DO PROGRAMA	10
FIGURA 8 - PASTA "TEMPLATES" NA VERSÃO COMPLETA DO PROGRAMA	10
FIGURA 9 - FUNÇÕES DOS BOTÕES DA "MAIN FOLDER FILES LIST"	11
FIGURA 10 - FUNÇÕES DOS BOTÕES DA "DBMAKER"	13
FIGURA 11 - FUNÇÕES DOS BOTÕES DA VIEW "PLC_TAPS"	14
FIGURA 12 - FUNÇÕES DOS BOTÕES DA VIEW "SELECTROBOT"	15
FIGURA 13 - FUNÇÃO "ADDBUTTONS"	15
FIGURA 14 - FUNÇÃO "NEWROBOT"	17
FIGURA 15 - FUNÇÕES DOS BOTÕES DA VIEW "TREEVIEWMANAGER"	18
FIGURA 16 - FUNÇÕES DOS BOTÕES DA VIEW "EXCELASKER"	19
FIGURA 17 - FUNÇÕES DOS BOTÕES DA VIEW "RENAMEPLC"	20
FIGURA 18 - FUNÇÃO "RENAME"	21
FIGURA 19 - FUNÇÃO DO BOTÃO "ROBOTLIST"	22
FIGURA 20 - FUNÇÕES DOS BOTÕES DA VIEW "HARDWAREGENERATOR"	23
FIGURA 21 - FUNÇÃO "EXCELToMATRIX"	24
FIGURA 22 - FUNÇÕES RESPONSÁVEIS POR IDENTIFICAR O TIPO DE FICHEIRO	24
FIGURA 23 - FUNÇÕES DE LEITURA DE FICHEIROS "SYMBOLIC" E "NETWORKLIST"	25
FIGURA 24 - FUNÇÕES CLASSE "PDFREADER"	26
FIGURA 25 - FUNÇÕES DE CRIAÇÃO DE FICHEIROS XML DO TIPO "PLCTAG"	27
FIGURA 26 - FUNÇÕES DA CLASSE "CACHEMANAGER"	28

1. Estrutura

Esta aplicação é composta por uma “*Solution*” com o nome “*TIAPortalOpennessDemo*” e dentro desta existem dois projetos com os nomes “*TiaOpennessHelper*” e “*TIAPortalOpennessDemo*”. É no projeto “*TiaOpennessHelper*” que estão presentes as principais classes que servem como auxílio ao projeto “*TIAPortalOpennessDemo*”.

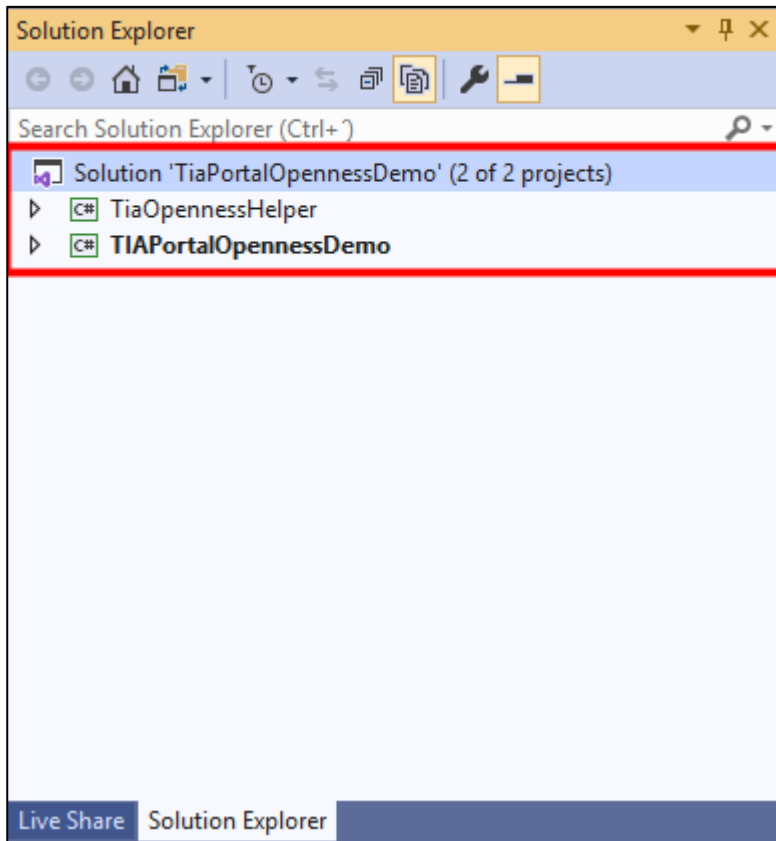


Figura 1 – Estrutura da Solution “*TiaPortalOpennessDemo*”

1.1. TIAPortalOpennessDemo

Este projeto é composto por seis pastas: “*Commands*”, “*Images*”, “*Services*”, “*Utilities*”, “*ViewModels*” e “*Views*”.

Foi acrescentado conteúdo às pastas:

- **Views:** onde estão criadas as janelas da aplicação, foram acrescentadas novas janelas;
- **ViewModels:** onde se encontram as classes que comandam os elementos de uma determinada janela, foram criadas novas classes para as novas janelas criadas e um ficheiro *schema* para verificar se o ficheiro XML “*Config*”, importado na funcionalidade “*Rename PLC*”, é ou não válido;
- **Images:** onde estão as imagens utilizadas pelo programa, foram importadas novas imagens.

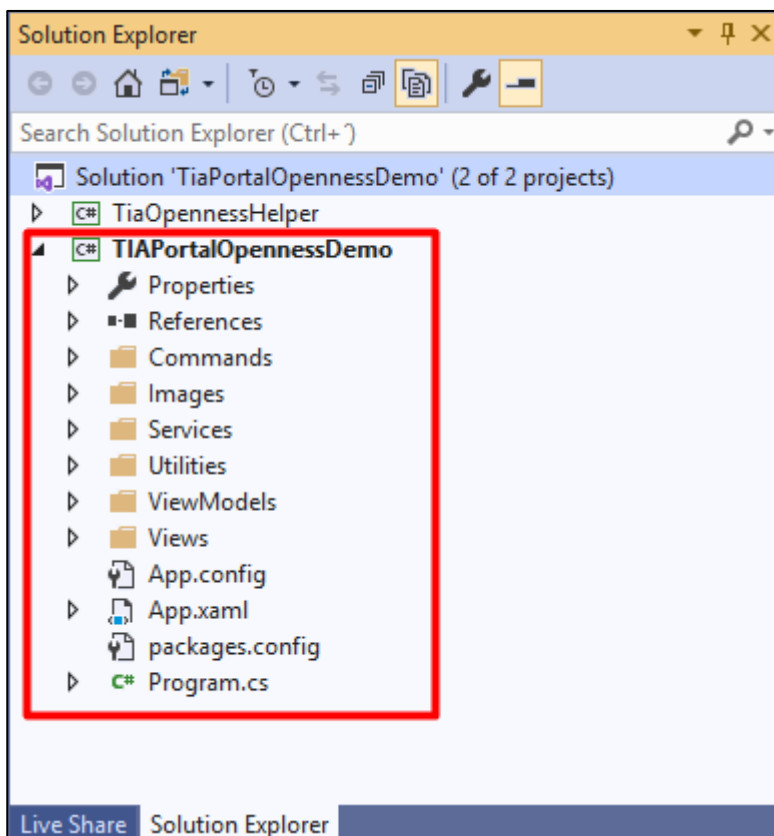


Figura 2 - Estrutura do projeto "TIAPortalOpennessDemo"

1.2. TiaOpennessHelper

Este projeto é composto por oito pastas: “*Enums*”, “*SequenceGenerator*”, “*Models*”, “*PLCDBGenerator*”, “*SCLParser*”, “*Utils*”, “*VWSymbolic*” e “*XMLParser*”.

Foi acrescentado ao projeto as pastas:

- **SequenceGenerator**: Contém todas as classes/janelas utilizadas pela funcionalidade “*Sequence Generator*”;
- **PLCDBGenerator**: Onde estão as classes, e a janela, utilizadas pela funcionalidade “*PLC DB Generator*”;
- **VWSymbolic**: Composta por classes que servem de auxílio à funcionalidade “*Generate Symbolic*” e um ficheiro *schema*;

Para além das classes criadas nas novas pastas, foram ainda criadas outras para servir de auxílio nas diversas funcionalidades do programa:

- Na pasta “*Utils*” foram criadas as classes:
 - **CacheManager**: Utilizada pela funcionalidade “*Main folder files list*” onde estão as funções responsáveis pela manipulação de informação em *cache*;
 - **FolderInfo**: Utilizada como uma estrutura organizada de ficheiros permitindo guardar o caminho do ficheiro e o seu nome;
 - **uFindVisualChild**: Usada somente pela funcionalidade “*Generate Symbolic*”, para facilitar a busca por elementos selecionados do tipo “*Combobox*” nas tecnologias do robô.
- Na pasta “*XMLParser*” foi criada uma pasta com o nome “*XMLEditor*” com três classes que servem de ajuda para a criação do *hardware* do robô pela funcionalidade “*Hardware Generator*”.

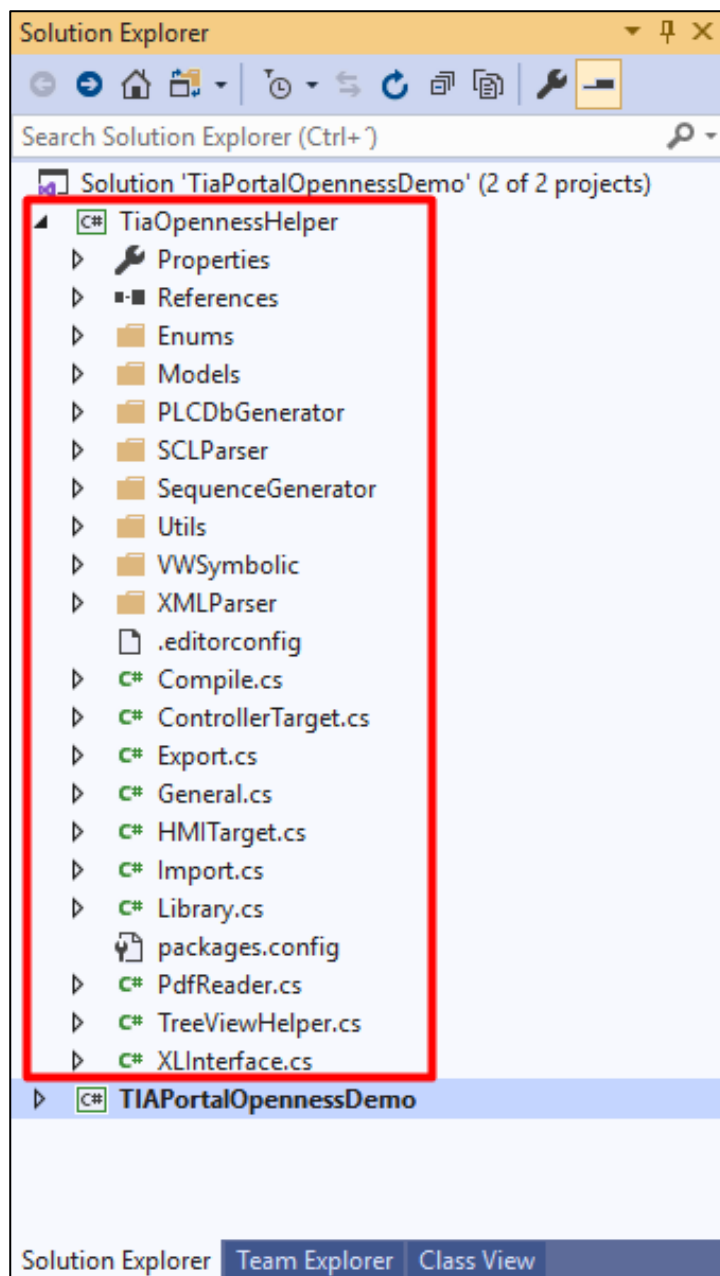


Figura 3 - Estrutura do projecto "TiaOpennessHelper"

2. Organização

Sendo esta aplicação programada utilizando a arquitetura “*Model-View-ViewModel*”, ou “MVVM”, é necessário que todas as janelas tenham uma classe que controle os seus componentes e as suas funcionalidades.

Na pasta “*ViewModels*”, do projecto “*TIAPortalOpennessDemo*”, podemos encontrar as classes que controlam algumas das janelas presentes na pasta “*Views*”. Essas classes designam-se por “*ViewModels*”.

Algumas janelas, ou *views*, não têm associado um *viewmodel* próprio. Temos o caso das janelas “*CreateFolderDialog*”, “*FileBrowserControl*”, “*ImportCaxControl*”, “*Overlay*” e “*SettingsView*”, em que todas elas têm o *viewmodel* “*MainWindowViewModel*” como base.

As janelas “*PLC_Taps*”, “*RobotView*”, “*TreeViewManager*”, “*ExcelAsker*”, “*DBMaker*” e “*OptionsRobotView*” não têm um *viewmodel* associado e, assim sendo, toda a sua programação é feita no código fonte, ou *sourcecode*, da *view*.

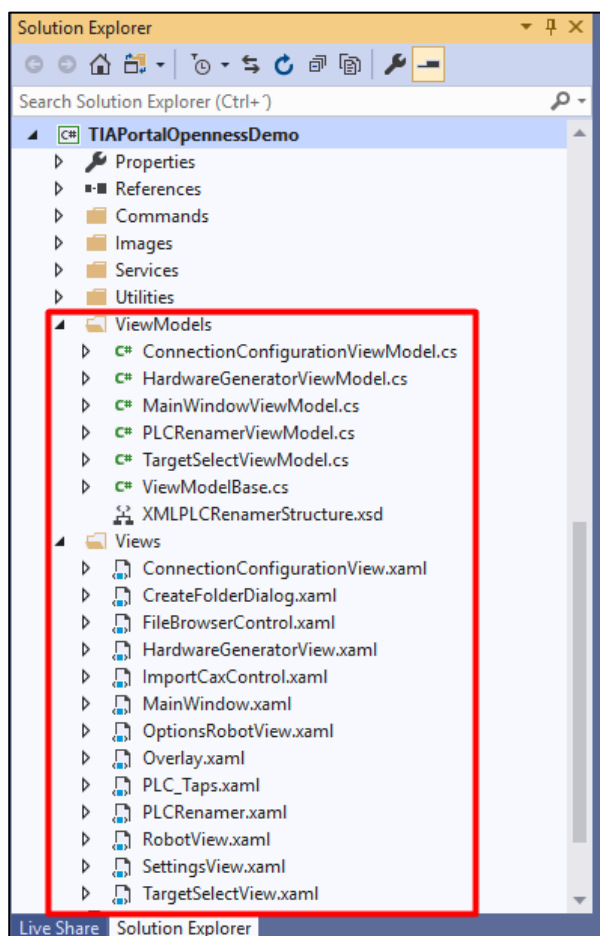


Figura 4 - Arquitetura MVVM

3. Conexão ao TIA Portal

A estrutura de ficheiros carregada do TIA Portal para o TIA Portal Openness foi modificada.

Ao invés de serem carregados todos os elementos que constituem o projeto do TIA Portal, são apenas carregados os que serão utilizados pelas funcionalidades do TIA Portal Openness. Com esta modificação, existe uma melhoria significativa no tempo que leva a conexão a ser feita.

Os elementos carregados são organizados hierarquicamente com a ajuda de uma *treeview*. Esta *treeview* é gerada pela função “*LoadProjectTreeView*”, presente na classe “*MainWindowViewModel*”.

A conexão ao TIA Portal é feita pela função “*ConnectToTIA*”, também presente na classe “*MainWindowViewModel*”. Esta classe recebe como parâmetro de entrada uma variável do tipo booleano com o objetivo de indicar se a conexão irá ser feita exclusivamente para a funcionalidade “*Rename PLC*”. Se a conexão for feita apenas para renomear o PLC, então a *treeview* gerada quando se faz uma ligação ao TIA Portal, não será criada.

```
/// <summary> Connect to TiaPortal  
private void ConnectToTia(bool renamePLC)...
```

Figura 5 - Função “*ConnectToTia*”

```
/// <summary>Loads the project TreeView.  
private void LoadProjectTreeView()...
```

Figura 6 - Função “*LoadProjectTreeView*”

4. Templates de código

Para facilitar a criação de ficheiros XML e Excel por parte do programa, foram criados vários *templates*. Estes *templates* devem sempre acompanhar a pasta do programa para que tudo funcione corretamente.



Name	Date modified	Type	Size
 Templates	20/01/2020 08:13	File folder	
 TiaPortalOpenness	20/01/2020 08:51	File folder	

Figura 7 - Estrutura de pastas do programa

Na versão completa do programa, que inclui o projeto em *Visual Studio*, a pasta “*Templates*” encontra-se na pasta “*TiaPortalOpennessDemo*” -> “*bin*”.





Name	Date modified	Type	Size
 Debug	20/01/2020 08:45	File folder	
 Release	17/01/2020 08:16	File folder	
 Templates	17/01/2020 08:16	File folder	
 x64	17/01/2020 08:16	File folder	

Figura 8 - Pasta “*Templates*” na versão completa do programa

5. Funcionalidade “*Main folder files list*”

Todas as ações desta funcionalidade estão programadas na *viewmodel* “*MainWindowViewModel*”.

Existem quatro possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region Main folder files list
/// <summary> Handles the Executed event of the RefreshMainTreeCommand control.
private void RefreshMainTreeCommand_Executed(object sender, EventArgs e)...
```

```
/// <summary> Handles the Executed event of the EditMainTreeFileCommand control.
private void EditMainTreeFileCommand_Executed(object sender, EventArgs e)...
```

```
/// <summary> Handles the Executed event of the ChooseFolderCommand control.
private void ChooseFolderCommand_Executed(object sender, EventArgs e)...
```

```
/// <summary> Handles the Executed event of the ImportMainTreeFileCommand contro ...
private void ImportMainTreeFileCommand_Executed(object sender, EventArgs e)...
```

```
#endregion
```

Figura 9 - Funções dos botões da “*Main folder files list*”

6. Funcionalidade “*PLC DB Generator*”

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*DBMaker*”, dentro da pasta “*PLCDBGenerator*”.

Existe um grande número de variáveis criadas nesta *view*, isto porque muitas delas não são utilizadas apenas localmente mas também por outras classes dentro da pasta “*PLCDBGenerator*”.

Dentro da pasta “*PLCDBGenerator*” podemos encontrar as classes:

- **EngAssist:** Funciona como uma estrutura de dados para armazenar as informações referentes à *Worksheet* “*EngAssist*”;
- **Frag:** Funciona como uma estrutura de dados para armazenar as informações do robô (perfil, estação, função) e o nome da *Worksheet* utilizada para recolher estas informações da folha de Excel “*Schnittstelle*”;
- **NetworkDBMaker:** Onde estão armazenadas as funções necessárias para a criação de todas as bases de dados (ficheiros XML) geradas por esta funcionalidade;
- **PLC_Tap:** Funciona como uma estrutura de dados para armazenar as informações de uma *tag*;
- **ReplaceActions:** Funciona como estrutura de dados para organizar quais as ações que serão substituídas em cada *Worksheet* do ficheiro “*PLC DB*”, guardando qual o texto a substituir e o texto pelo qual será substituído;
- **UserConfig:** Funciona como estrutura de dados para armazenar o nome e o valor de cada entrada na *Worksheet* “*User Config*” do ficheiro “*PLC DB*”;
- **Variable:** Funciona como estrutura de dados para armazenar as variáveis presentes em cada *Worksheet* do ficheiro “*PLC DB*”, guardando o nome da variável, o tipo, o comentário e a ação.

Existem três possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region Button Events
/// <summary> Button Event that creates the data base of SPS, Schutzkreis and Sa ...
private void Button_CreateDB(object sender, RoutedEventArgs e) {...}

/// <summary> Button that saves the current grid view values in the current sele ...
private void Button_SaveCurrentValues(object sender, RoutedEventArgs e) {...}

/// <summary> Button used to clear the grids
private void Button_ClearGrid(object sender, RoutedEventArgs e) {...}
#endregion
```

Figura 10 - Funções dos botões da “DBMaker”

7. Funcionalidade “*PLC Tags*”

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*PLC_Taps*”, dentro da pasta “*Views*”.

Existem três possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region Button Actions
/// <summary> Saves the Datagridview values on the Excel file
private void Save(object sender, RoutedEventArgs e)...

/// <summary> Clears the Datagridview values
private void Clear(object sender, RoutedEventArgs e)...

/// <summary> Creates the xml Version of the plc tags
private void CreateXML(object sender, RoutedEventArgs e)...
#endregion
```

Figura 11 - Funções dos botões da *view* “*PLC_Taps*”

8. Funcionalidade “*Generate Symbolic*”

8.1. Select Robot

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*OptionsRobotView*”, dentro da pasta “*Views*”.

Todas as informações retiradas da folha Excel, são armazenadas em listas e enviadas para esta *view*.

Existem duas possíveis ações que podem ser realizadas nesta funcionalidade: Criar um novo robô ou selecionar um já criado na folha Excel.

```
/// <summary> Initialise Robot View Window with Robot from Excel Constructor  
private void InitRobotViewRobFromExcel(RobotInfo robInfo)...  
  
/// <summary> Handles "BtnCreateNew_Click" event  
private void BtnCreateNew_Click(object sender, RoutedEventArgs e)...
```

Figura 12 - Funções dos botões da *view* “*SelectRobot*”

Os botões são criado dinamicamente pela função “*AddButtons*” conforme a quantidade de robôs presente na folha Excel selecionada. Esta função trata de formatar corretamente os botões e apresenta-os no *design* da *view*.

```
/// <summary> Add buttons to window  
private void AddButtons()...
```

Figura 13 - Função “*AddButtons*”

8.2. Generate Symbolic

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*OptionsRobotView*”, dentro da pasta “*RobotView*”.

Esta funcionalidade utiliza as classes dentro da pasta “*VWSymbolic*” como auxílio na execução das suas ações. Esta pasta é composta por um ficheiro *schema*, para verificar se a formatação de um ficheiro XML do tipo “*symbolic*”, aberto pela “*Main folder files list*”, é ou não válido, e pelas classes:

- **Robot:** Uma classe estática com o objetivo de preservar os valores inseridos nas listas utilizadas para gerar o simbólico de um robô. Os valores destas listas são preenchidos na *viewmodel* “*MainWindowViewModel*”, dentro da função “*LoadSymbolic*”;
- **RobotBase:** Funciona como estrutura de dados para armazenar as informações de uma *tag* presente na *Worksheet* “*Grund*” de um ficheiro de Excel do tipo “*Symbolic*”. Esta estrutura de dados guarda o simbólico da *tag*, o seu tipo, o endereço e o comentário;
- **RobotInfo:** Funciona como estrutura de dados para armazenar as informações de um simbólico já criado. Esta estrutura de dados guarda o nome do robô, o tipo de segurança (“*Range Monitoring*” ou “*Operation*”), o endereço inicial, as tecnologias utilizadas e o seu tipo (“*Basic Slave*” ou “*Laser Slave*”);
- **RobotSafeOperation:** Funciona como estrutura de dados para armazenar as informações de uma *tag* presente na *Worksheet* “*Rob Safe Operation*” de um ficheiro de Excel do tipo “*Symbolic*”. Esta estrutura de dados guarda o simbólico da *tag*, o seu tipo de dados, o endereço e o comentário;
- **RobotSafeRangeMonitoring:** Funciona como estrutura de dados para armazenar as informações de uma *tag* presente na *Worksheet* “*Rob Safe Range Monitoring*” de um ficheiro de Excel do tipo “*Symbolic*”. Esta estrutura de dados guarda o simbólico da *tag*, o seu tipo de dados, o endereço e o comentário;
- **RobotTechnologie:** Funciona como estrutura de dados para armazenar as informações de uma *tag* relacionada à tecnologia de um robô. Esta estrutura de dados guarda o nome da tecnologia, o número do FB, o tipo de segurança (“*Range Monitoring*” ou “*Operation*”), o simbólico, o tipo de dados, o endereço e o comentário;

- **SymbolicManager:** Utilizada como auxílio na execução das ações da *view* “*RobotView*”. É responsável por gerar os ficheiros XML dos simbólicos e importar os mesmos para o TIA Portal.

O simbólico é gerado pela função “*New Robot*”, presente na classe “*SymbolicManager*”, chamada depois de clicado o botão “*Create Robot*” na *view*.

```
/// <summary> Creates new Robot  
public void NewRobot(int startAddress, string name, string robSafe, List<string> technologies, string type, bool importToTia)...
```

Figura 14 - Função "NewRobot"

9. Funcionalidade “*Sequence Generator*”

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*TreeViewManager*”, dentro da pasta “*SequenceGenerator*”.

Nesta pasta podemos encontrar as classes:

- **ExcelManager:** Contém funções utilizadas maioritariamente na manipulação de ficheiros Excel do tipo “*Sequence*”;
- **GrafcetManager:** É composta por funções utilizadas para preparar o XML do *Graphcet*
- **NetworkManager:** É composta por funções utilizadas para gerar o XML do *Graphcet*;
- **Step:** Funciona como estrutura de dados para armazenar as informações de uma *step* do ficheiro Excel do tipo “*Sequence*”;
- **StepHandler:** Funciona como estrutura de dados para organizar e facilitar a criação do ficheiro XML gerado pela funcionalidade.
- **WorkSheet:** Funciona como estrutura de dados para facilitar a leitura de um ficheiro do tipo “*Sequence*”. Esta estrutura é responsável por guardar o nome de uma *Worksheet* e as *steps* criadas na mesma.

Existem quatro possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region BUTTON EVENTS
/// <summary> Creates a new Excel with the DataGrid Values
private void Btn_Add_Click(object sender, RoutedEventArgs e)...

/// <summary> Event that clears the DataGrid
private void Btn_Clear_Click(object sender, RoutedEventArgs e)...

/// <summary> Event that calls the method to Generate the NetWork and Grafcet
private void Btn_GenerateNetNGraf_Click(object sender, RoutedEventArgs e)...

/// <summary> Saves the current DataGrid values in the chosen path
private void Btn_Save_Click(object sender, RoutedEventArgs e)...
#endregion
```

Figura 15 - Funções dos botões da *view* “*TreeViewManager*”

9.1. Create Excel

Todas as ações desta funcionalidade estão programadas no *sourcecode* da *view* “*ExcelAsker*”, dentro da pasta “*SequenceGenerator*”.

Existem duas possíveis ações que podem ser realizadas nesta funcionalidade: “*Create*” e “*Cancel*”.

```
/// <summary> Button Create Click Event
private void Button_Create(object sender, RoutedEventArgs e)...

/// <summary> Button Cancel Click Event
private void Button_Cancel(object sender, RoutedEventArgs e)...
```

Figura 16 - Funções dos botões da *view* “*ExcelAsker*”

10. Funcionalidade “*Rename PLC*”

Todas as ações desta funcionalidade estão programadas no *viewmodel* “*PLCRenameViewModel*”, dentro da pasta “*ViewModels*”.

Existem quatro possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region Commands
/// <summary> Event handler rename button click
private void RenameCommand_Executed(object sender, EventArgs e)...
```



```
/// <summary> Event handler openlog button click
private void OpenLogCommand_Executed(object sender, EventArgs e)...
```



```
#region Config Commands
/// <summary> Event handler export config button click
private void ExportConfigCommand_Executed(object sender, EventArgs e)...
```



```
/// <summary> Event handler import config button click
private void ImportConfigCommand_Executed(object sender, EventArgs e)...
```

```
#endregion
#endregion
```

Figura 17 - Funções dos botões da view “RenamePLC”

A principal função desta funcionalidade é a “*RenamePLC*”, chamada quando clicado o botão “*Rename*”. Esta função é responsável por alterar os IP’s, exportar os elementos, eliminar os elementos do TIA Portal, editar o XML, renomear o *hardware* e importar os ficheiros alterados.

```
/// <summary> Rename PLC
private void RenamePLC()
{
    var parent = (current as PlcBlockUserGroup).Parent.Parent;
    var groups = (current as PlcBlockUserGroup).Groups;

    Change Devices IP

    if (onlyChangeIP) return;

    Export Elements

    Delete Elements (TIA PORTAL)

    Edit XML

    Rename Devices

    Import Element
}
```

Figura 18 - Função “Rename”

11. Funcionalidade “*RobotList*”

Todas as ações desta funcionalidade estão programadas no *viewmodel* “*MainWindowViewModel*”, dentro da pasta “*ViewModels*”.

Esta funcionalidade é acionada através do botão “*RobotList*”, que, quando clicado, utiliza a função “*GenerateRobotListCommand_Executed*” que trata de gerar os ficheiros XML dos robôs da folha de Excel “*Schnittstelle*”.

```
/// <summary> Handles the Executed event of the GenerateRobotListCommand control ...  
private void GenerateRobotListCommand_Executed(object sender, EventArgs e)...
```

Figura 19 - Função do botão “*RobotList*”

12. Funcionalidade “*Hardware Generator*”

Todas as ações desta funcionalidade estão programadas no *viewmodel* “*HardwareGeneratorViewModel*”, dentro da pasta “*ViewModels*”.

Esta funcionalidade utiliza as classes dentro da pasta “*XMLParser*” -> “*XMLEditor*” como auxílio para gerar o *hardware* do PLC.

Existem três possíveis ações que podem ser realizadas nesta funcionalidade, todas elas são ativas por meio de botões.

```
#region Commands
/// <summary> Event handler generate hardware button click
private void GenerateHardwareCommand_Executed(object sender, EventArgs e) ...

/// <summary> Event handler choose library path button click
private void ChooseLibraryPathCommand_Executed(object sender, EventArgs e) ...

/// <summary> Event handler choose library type button click
private void ChooseLibTypeCommand_Executed(object sender, EventArgs e) ...
#endregion
```

Figura 20 - Funções dos botões da view “*HardwareGenerator*”

13. Classes importantes

13.1. XLInterface

A classe “*XLInterface*” encontra-se dentro do projecto “*TiaOpennessHelper*” e é onde estão grande parte das funções de manipulação de ficheiros Excel.

Para aumentar a rapidez de leitura de cada *worksheet*, a informação contida na mesma, é guardada numa matriz. Esta conversão é feita pela função “*ExcelToMatrix*” que recebe como parâmetro de entrada uma *worksheet*.

```
/// <summary> Transform excel sheet in a matrix  
public static object[,] ExcelToMatrix(Worksheet xlWorksheet)...
```

Figura 21 - Função “*ExcelToMatrix*”

É também nesta classe onde se encontram as funções responsáveis por identificar um determinado tipo de ficheiro.

```
#region Identify File Type  
/// <summary> Check if excel file is a sequence  
public static bool IsSequence(object[,] matrix)..  
  
/// <summary> Check if excel file is a PLC Database  
public static bool IsPlcDb(object[,] matrix)..  
  
/// <summary> Check if excel file is a Symbolic  
public static bool IsSymbolic(object[,] matrix)..  
  
/// <summary> Check if excel file is a Robot WorkBook  
public static bool IsSchnittstelle(object[,] matrix)..  
  
/// <summary> Check if the excel file contains a Plc Tag sheet  
public static bool IsPlcTags(object[,] matrix)..  
  
/// <summary> Check if the excel file is a NetWorkList  
public static bool IsNetworkList(object[,] matrix)..  
  
/// <summary> Check file type  
public static string CheckFileType(string path)..  
#endregion
```

Figura 22 - Funções responsáveis por identificar o tipo de ficheiro

Esta classe também contém funções de manipulação de ficheiros do tipo “*Symbolic*” e “*NetworkList*”.

```
#region Network List Excel File
/// <summary> Get all devices from Network List Excel File
public static List<DeviceData> GetAllDevicesNetworkList(Worksheet xlWorksheet)...
#endregion

#region VWSymbolism Excel File
/// <summary> Get robot base properties from sheet
public static List<List<RobotBase>> GetRobotBase(object[,] matrix, Range lastCellRange)...

/// <summary> Get robot technologies from sheet
public static List<List<RobotTechnologie>> GetRobotTechnologies(object[,] matrix, Range lastCellRange, string type)...

/// <summary> Get robot safe range monitoring from sheet
public static List<List<RobotSafeRangeMonitoring>> GetRobotSafeRangeMonitoring(object[,] matrix, Range lastCellRange)...

/// <summary> Get robot safe operation from sheet
public static List<List<RobotSafeOperation>> GetRobotSafeOperation(object[,] matrix, Range lastCellRange)...

/// <summary> Get created robots info from sheet
public static List<RobotInfo> GetCreatedRobotsInfo(object[,] matrix)...
#endregion
```

Figura 23 - Funções de leitura de ficheiros “*Symbolic*” e “*NetworkList*”

13.2. PdfReader

A classe “*PdfReader*” encontra-se dentro do projecto “*TiaOpennessHelper*” e é onde se encontram as funções de manipulação de ficheiros PDF. É utilizada especificamente para leitura do *EPlan*.

```
namespace TiaOpennessHelper
{
    public partial class OpennessHelper
    {
        /// <summary> Get all page numbers that contains Geräteliste
        public static List<int> GetGaretelistePages(string fileName)...

        /// <summary> Return the name of a module
        public static string GetHWPart(string orderNumber)...

        /// <summary> Look for a device on EPlan PDF and returns it's information
        public static List<List<string>> HWInfo(string path, string deviceFGroup, string deviceIdentifier, List<int> garetelistPages)...

        #region Private Methods
        /// <summary> Get Hardware Raw information from the pdf text
        private static List<List<string>> GetHWRawInformation(string[] text, string HWName, string HWIdentifier)...

        /// <summary> Check if a given string is a Part Number
        private static bool IsPartNumber(string part)...

        /// <summary> Get all page numbers that contains a string
        private static List<int> GetPagesByString(string fileName, String searchText, List<int> garetelistPages)...

        /// <summary> Read text from PDF file
        private static StringBuilder ReadPdf(string path, List<int> pages)...
        #endregion
    }
}
```

Figura 24 - Funções classe “*PdfReader*”

13.3. XmlParser

A classe “*XmlParser*” encontra-se dentro do projecto “*TiaOpennessHelper*”, na pasta “*XMLParser*”, e é onde se encontram grande parte das funções de manipulação de ficheiros XML.

Para além das funções gerais, esta classe também contém funções responsáveis pela criação de ficheiros do tipo “*PLCTag*”.

```
#region PLC Tags
/// <summary> Insert a new tag
public static void InsertTag(XmlDocument doc, XmlDocument tagDoc, string symbolic, string dataType, string address, string comment, bool externalAccessible, bool externalVisible, bool externalWritable)...

/// <summary> Convert XML Tags to a List of PLC Tag's
public static void XmlToPlcTags(string path)...

/// <summary> Check if Xml doc is a PLC Tags doc
public static bool IsPlcTags(string path)...

/// <summary> Create tag for Fail
public static XElement CreateTag(string symbolic, string datatype, string comment, string address)...
#endregion
```

Figura 25 - Funções de criação de ficheiros XML do tipo "PLCTag"

13.4. CacheManager

A classe “*CacheManager*” encontra-se dentro do projecto “*TiaOpennessHelper*”, na pasta “*Utils*”, e é onde se encontram as funções responsáveis pela manipulação de conteúdo em *cache*. É responsável por guardar, procurar, definir, limpar ou remover objetos em *cache*.

Esta classe é utilizada pela funcionalidade “*Main Folder Files List*”, para verificar se um determinado ficheiro já foi aberto e tem informações em *cache*.

```
namespace TiaOpennessHelper
{
    public partial class OpennessHelper
    {
        /// <summary> Get cache data
        public static object GetCacheData(string CacheKey)...

        /// <summary> Set cache data
        public static void SetCacheData(string CacheKey, object StoreItem)...

        /// <summary> Clear cache data
        public static void ClearCacheData()...

        /// <summary> Dispose specific cache data
        public static void DisposeCacheData(string CacheKey)...

        /// <summary> Get CacheKey that contains a certain string
        public static object GetCacheDataContaining(string str)...
    }
}
```

Figura 26 - Funções da classe “*CacheManager*”

14. Notas finais

Existem muitas funções e classes presentes nos dois projetos que não foram mencionadas neste ficheiro porque são originais da aplicação “*TiaPortalOpenness*” e podem ser encontradas na documentação do “*TiaPortalOpenness*”.

Grande parte das funções estão corretamente comentadas, sendo facilmente detetada a sua funcionalidade.

Sendo esta aplicação programada com a arquitetura MVVM, a maneira correta de aceder aos seus campos é pelo método “*Binding*”. Por exemplo, caso se queira aceder ao texto de um controlo do tipo “*TextBox*”, é necessário programar da seguinte maneira:

1. Declarar o controlo na *view*:

```
<TextBox Text="{Binding Texto}" />
```

2. Declarar na *viewmodel* a variável “*Texto*”:

```
private string _texto;
public string Texto
{
    get { return _texto; }
    set
    {
        if (!string.Equals(_texto, value))
        {
            _texto = value;
            RaisePropertyChanged("Texto");
        }
    }
}
```

3. Pode-se aceder ao texto dentro da *TextBox* criada pela variável “*_texto*” e modificá-lo pela variável “*Texto*”.