VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

AUTOMATICKÉ GENEROVÁNÍ PLC PROGRAMŮ PRO MODELY JEŘÁBŮ POMOCÍ TIA PORTAL OPENNESS
AUTOMATIC CREATION OF PLC PROGRAMS FOR CRANE MODELS USING TIA PORTAL OPENNESS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Václav Slanina

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jakub Arm

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: Václav Slanina ***ID:*** 164401
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Automatické generování PLC programů pro modely jeřábů pomocí TIA Portal Openness

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je vytvoření programů pro PLC S7-1200 v jednotlivých modelech jeřábů automatizovaně pomocí nástroje TIA Portal Openness dle XML předpisu.

1. Seznamte se s nástrojem TIA Portal Openess.
2. Vytvořte vzorový PLC program pro řízení modelu jeřábu a rozdělte jej na jednotlivé úlohy.
3. Nadefinujte vhodně schéma XML souboru sloužící jako předpis pro generování jednotlivých PLC programů co nejuniverzálněji.
4. Navrhněte a realizujte desktopovou aplikaci, která generuje PLC programy dle XML předpisu.
5. Zdokumentujte postup generování a konfiguraci PLC programů pomocí TIA Openness.

DOPORUČENÁ LITERATURA:

Firemní dokumentace Siemens pro TIA Portal Openness.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: Ing. Jakub Arm
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Tato práce se zabývá TIA Portal Openness, jeho možnostmi z hlediska automatizace generování programů pro PLC, kompilace programu a správy jednotlivých zařízení. Zaměří se na jeho základní vlastnosti, jako konfigurace zařízení, jejich spojování do sítí, export a import zdrojových souborů bloků programu a aplikuje je na generování programu pro laboratorní modely jeřábu. Pro generování byla vytvořena desktopová aplikace, jako praktický příklad pro snadné uchopení tohoto tématu. Z této práce lze získat základní přehled o tomto nástroji a možnostech jeho využití.

KLÍČOVÁ SLOVA

Siemens, TIA Portal Openness, PLC, automatizace, generování programu, hanojské věže, model jeřábu

ABSTRACT

This thesis deals with TIA Portal Openness, its potential in terms of automating PLC program generation, compilation and management of individual devices. It aims to its basic features, such as working with hardware and network configuration, exporting and importing source files of blocks, and demonstrates them on a simple application for crane models. For generation, a desktop application was created as a practical example for easy understanding of this topic. It is possible to get basic overview of this tool and it's aplication from this thesis.

KEYWORDS

Siemens, TIA Portal Openness, PLC, automation, program generation, tower of hanoi, crane model

SLANINA, Václav. *Automatické generování PLC programů pro modely jeřábů pomocí TIA Portal Openness*. Brno, Rok, 71 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. Jakub Arm

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Automatické generování PLC programů pro modely jeřábů pomocí TIA Portal Openness“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu Ing.Jakubu Armovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Na stránkách této bakalářské práce se budu zabývat dokumentací průběhu automatizovaného generování programů pro model Jeřábu skládajícího hlavolam známý jako hanojské věže. Celý systém je řízen PLC (Programable logic controler - programovatelný logický automat/procesor) od firmy Siemens. Tato firma vyvíjí svůj software pro návrh a kompilaci programů pro tato zařízení a pro další operace s nimi TIA Portal (Totally integrated automation - plně integrované prostředí pro automatizaci). Nejdříve tedy zopakuji základní paradigmata týkající se tohoto prostředí. Všechny informace i následná desktopová aplikace se týkají TIA Portalu V15, je proto možné že se mohou některé informace lišit od verze používané čtenářem.

V nejdůležitější části teoretického rozboru se budu zabývat nástrojem TIA Portal Openness, který nám poslouží pro náš cílový záměr. Jde o veřejnou API (Application Programming Interface - rozhraní pro programování aplikací), která umožňuje ovládání TIA Portal a generování programu podle XML předpisu z externí aplikace napsané v jazyce C# nebo Visual Basic a to na platformě .NET frameworku ve verzi 4.6.2 nebo vyšší. O vlastnostech jazyka c# použitých v desktopové aplikaci se ve zkratce také zmíním.

Dále se zdokumentuji proces generování programu PLC, tak aby bylo jasné, které prvky aplikace jsou zásadní, jaká řešení jsem použil a jak využívat metody, které jsem pro účely obsluhy aplikace napsal.

Teoretická část práce bude koncipována hlavně jako návod k základům a popis metod, které nabízí TIA Portal Openness, doplněná o mé postřehy. Zdrojem, z něž čerpám se stala dokumentace firmy Siemens, zejména manuál k TIA Portal Openness.

1 TIA Portal - programování PLC

Vzhledem k tomu, že výsledný program bude automaticky generovat program pro PLC, je dobré nejdříve znát základy samotného programování v TIA Portal.

1.1 Operační systém

Nachází se v každém CPU a organizuje všechny funkce a sekvence kódu, které nejsou spojeny se specifickou úlohou řízení.

Úlohy operačního systému zahrnují:[1]

- Processing a tzv. teplý restart
- Obnovení operačního obrazu vstupů a výstupů
- Volání programu uživatele
- Detekci přerušení a volání přerušení OB (organizačního bloku)
- Detekci chyb
- Obsluha paměťového prostoru

Operační systém je zahrnut v CPU již z výroby.[1]

1.2 Používané jazyky

Prostředí TIA Portal umožňuje jak psaní kódu ve formě textu, který má podobnou strukturu jako programovací jazyk PASCAL nebo ve formě grafických diagramů a to LAD nebo FBD.

1.2.1 SCL

Jde o strukturovaný vysokoúrovňový programovací jazyk SCL (Structured Control Language - strukturovaný řídící jazyk). Jeho gramatika je zanesena v normě DIN EN 61131-3.[1]

Prvky jazyka SCL využívá prvků vyšších programovacích jazyků v závislosti na tom jaké podporuje dané zařízení PLC, tj. časovače, vstupy a výstupy, výrazy atd.

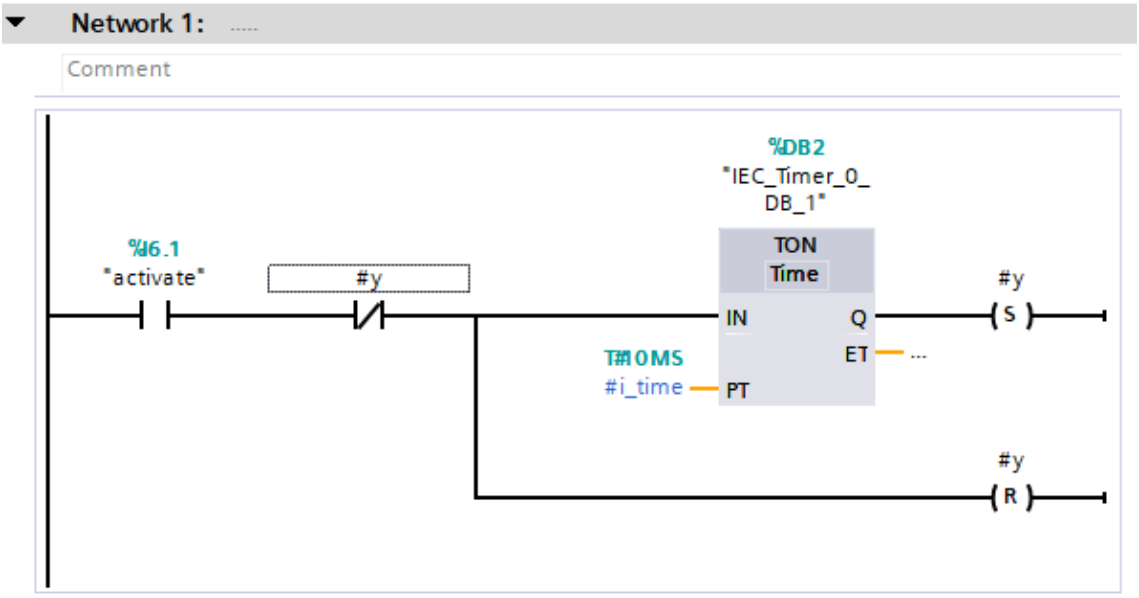
Použití

- Správa dat
- Optimalizace procesů
- Správa procedur

- Matematické operace

1.2.2 LAD

Jedná se o ladder logic, tj. žebříkový diagram s funkčními bloky. Tento typ programování je velmi názorný, napodobuje elektrické schéma, které ovládá výstupy, využívá však navíc funkcí a funkčních bloků, které umožňují zpřehlednit diagram a využívat některé části opakovaně jednoduchým způsobem.[1]



Obr. 1.1: Příklad programu v jazyce LAD

Programovací styly

Lineární programování Lze využít pro jednoduché programování s nízkou úrovní automatizace. Jde o obdobu procedurálního programování, kdy se provádí celý program postupně.[1]

Strukturované programování Pokud provádíme složitější program, je dobré využívat funkcí a funkčních bloků, které provádí jednotlivé podúlohy. [1]

To poskytuje různé výhody[1]:

- Zjednodušení složitých programů
- Jednotlivé sekce mohou být standardizovány a mohou být i vícekrát použity
- Jednodušší změny v programu
- Jednodušší vyhledání chyb

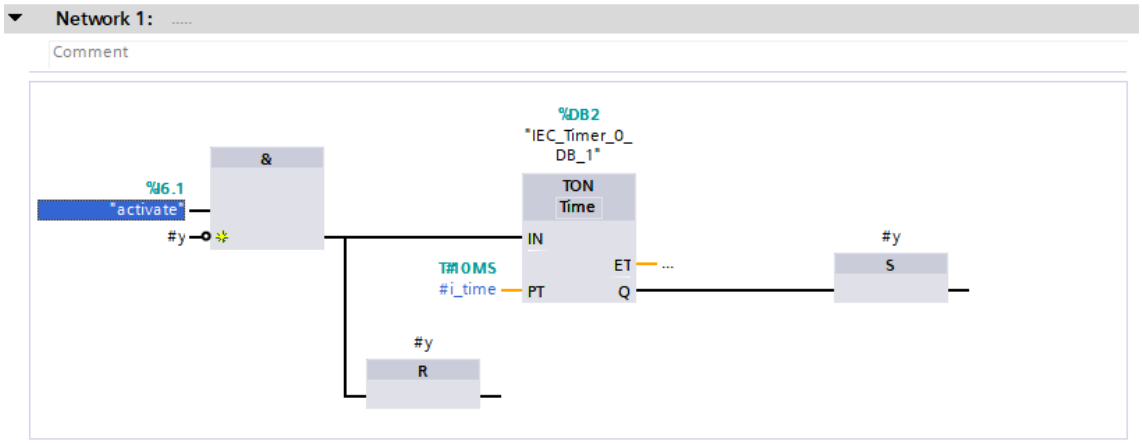
Hloubka bloků je však omezena použitým CPU. Já budu využívat CPU S7-1200, které umožňuje 16 bloků volaných z cyklického nebo startovního OB a 6 dalších bloků na každý OB obsluhující přerušení. Hloubka struktur je omezena na 8 a je nezávislá na použitém CPU.[1]

Přehled bloků

- Organizační bloky
  - definují samotnou strukturu programu uživatele
  - Zprostředkují interface mezi operačním systémem a programem
- Funkce
  - obsahuje tzv. programové rutiny, jde o opakující se úseky
- Funkční bloky
  - obsahují instance data bloky, jejich data mohou tak být použita i po jejich proběhnutí
- Bloky instancí (Instance data blocks)
  - blok sloužící k uložení dat uvnitř funkčního bloku
- Globální bloky dat
  - datové prostory použitelné jakýmkoli blokem

1.2.3 FBD

FBD (Function Block Diagram - diagram funkčních bloků) je druhá forma grafického jazyka, kterou TIA Portal nabízí. Podobně jako LAD je tedy názorná. Tento jazyk vychází ze schémat logických obvodů Boolovy algebry. Skládá se z jednotlivých sítí podobně jako LAD, na rozdíl od něj však pracuje s hradlovým systémem, tato hradla jsou stavěna do cesty signálům a upraví tak jejich cestu k výstupu. Jeho vlastnosti jsou poměrně intuitivní a podobné jako u LAD, proto zde nebudou dále rozváděny. [1]



Obr. 1.2: Příklad programu v jazyce FBD

2 C# a .NET framework

C# je vysokoúrovňový oběktově orientovaný programovací jazyk vyvíjený firmou Microsoft. Já budu pro psaní aplikací v tomto jazyce využívat vývojové prostředí Visual Studio 2017, které obsahuje platformu .NET ve verzi 4.7.0. To poskytuje pokročilý editor kódu, pohodlný designér pro aplikaci a integrovaný debuger, čehož při vývoji demo aplikace nepochybně využiji.[2]

2.1 Syntaxe jazyka

C# vychází svou konstrukcí z jazyka C++, pokud tedy jeho čtenář ovládá jazyk C, C++ nebo jazyk Java, bude pro něj poměrně snadné se v kódu tohoto jazyka zorientovat.

C# podporuje generické metody a typy, což poskytuje zvýšený výkon a bezpečnost kódu. Jde o obdobu šablon v C++, ovšem zde je rozšířena generika i na samotné objekty.

2.1.1 Objektové vlastnosti

C# jakožto objektově orientovaný jazyk podporuje tzv. zapouzdření, principy dědění a polymorfismu. Na rozdíl od C/C++ tento jazyk neobsahuje funkce, všechny procedury jsou tedy ve formě metod obsažených v jednotlivých objektech a to včetně metody Main().

2.1.2 Garbage Collection

Tato vlastnost .NET framework poskytuje automatickou správu paměti při dynamické alokaci paměti z takzvané haldy. K té dochází pokaždé když vytvoříme nový objekt. Garbage collector se při běhu a na konci programu postará o navrácení paměti zpět do rukou operačního systému a odalokuje paměť, kterou obsazují již nepoužívané objekty.[3]

2.2 Framework .NET

Tato platforma obsahuje mnoho knihoven, jejichž části jsou použity i v této práci. Některé generické metody a třídy si nyní přiblížíme. Tyto se nachází v oboru názvů System.Collections.Generic.

2.2.1 Třída List<T>

Představuje seznam objektů, ke kterým se přistupuje pomocí indexu. Poskytuje metody pro hledání, seřazení a manipulaci s obsaženým seznamem. [4]

2.2.2 Třída Dictionary<TKey,TValue>

Tato třída se používá jako kolekce objektů, k nimž se přistupuje pomocí klíče. [7]

3 TIA Portal Openness

3.1 Co je to Openness

Je to tzv. veřejná API, tedy rozhraní pro programování aplikací v TIA Portal. Ve zkratce nám poskytuje kontrolu nad funkcemi tohoto softwaru, můžeme tak vytvářet skripty pro generování programů a úpravu jejich parametrů pro vaše systémy ovládané pomocí PLC od firmy Siemens.

Toto automatizované generování je tak ideálním nástrojem v nastupujícím světě, kdy se zvyšuje poptávka po individualizovaných výrobcích, pomocí jedné aplikace tak můžeme ovládat stroj podle zadaných podmínek s velkou variabilitou.

V následujících kapitolách si základně popíšeme, jaké nástroje TIA Portal openness poskytuje a demonstrujeme si i jejich využití na jednoduché aplikaci vytvořené ve Windows Presentation Foundatoin, tedy formulářové aplikaci vyvíjené na platformě .NET od jeho verze 3.0.

3.2 Zdroj dat pro aplikaci

Vzhledem k tomu, že aplikaci budeme vyvíjet na platformě .NET, je možné, aby skriptovací aplikace byla modifikována, a to různými způsoby. Zdrojem dat pro tuto modifikaci výsledného programu může být například formulář v prezentační vrstvě, kde si může uživatel navolit požadované a dovolené nastavení nebo je možné čerpat data ze souboru. V tomto souboru se nachází formátovaná databáze vstupních údajů a může být ve formátu XML nebo CSV.

3.3 Spouštění TIA Portalu z openness aplikace

3.3.1 Konfiguračním souborem

Tento způsob spouštění se dá využít pro většinu případů, kdy potřebujeme spustit TIA Portal.

V konfiguračním souboru aplikace, který musí být distribuován spolu s ní, je třeba vytvořit reference na knihovny, které v kódu využíváme. V tomto souboru můžeme využít následující část kódu. [6, str. 73]

Výpis 3.1: XML kód konfiguračního souboru

```
<?xml version="1.0"?>
<configuration>
 <runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
    <assemblyIdentity name="Siemens.Engineering" culture="neutral"
publicKeyToken="d29ec89bac048f84"/>
    <!-- doplňte cestu podle vaší instalace -->
       <codeBase version="14.0.1.0" href=
"FILE://C:\...\Siemens.Engineering.dll"/>
   </dependentAssembly>
   <dependentAssembly>
    <assemblyIdentity name="Siemens.Engineering.Hmi" culture="
       neutral"
publicKeyToken="d29ec89bac048f84"/>
    <!-- doplňte cestu podle vaší instalace -->
   <codeBase version="14.0.1.0" href=
"FILE://C:\...\Siemens.Engineering.Hmi.dll"/>
   </dependentAssembly>
  </assemblyBinding>
 </runtime>
</configuration>
```

Poté můžeme spustit TIA Portal pomocí vytvoření instance typu TiaPortal a to s uživatelským nebo bez uživatelského rozhraní, čímž dosáhneme vyšší efektivity. Bez uživ. rozhraní totiž ušetříme čas, který by jeho spuštění zabralo a toho lze dobře využít v masové výrobě. Ovšem pro ladění chyb v kódu je vhodné spouštět prostředí s uživ. rozhraním.[6, str. 74]

Výpis 3.2: Spouštění instance TiaPortal.

```
using (TiaPortal tiaPortal
  = new TiaPortal(TiaPortalMode.WithUserInterface))
 {//sem vložte svůj kód pro práci s grafickým rozhraním}
using (TiaPortal tiaPortal
  = new TiaPortal(TiaPortalMode.WithoutUserInterface))
 {//sem vložte svůj kód pro spuštění bez grafického rozhraní}
```

3.3.2 AssemblyResolve metodou

Tento způsob lze využít, pokud chceme svůj program distribuovat, cestu k instalaci TIA Portalu je možno zadat například z příkazové řádky nebo z prezentační vrstvy aplikace.[6, str. 75]

Výpis 3.3: Kód pro obsluhu události AssemblyResolveru.

```
private static Assembly MyResolver
  (object sender, ResolveEventArgs args)
{
 int index = args.Name.IndexOf(',');
 if (index == -1)
 {
  return null;
 }
 string name = args.Name.Substring(0, index) + ".dll";
 // doplňte podle adresáře své instalace
 string path = Path.Combine(@"C:\Program Files\Siemens...",
    name);

 string fullPath = Path.GetFullPath(path);

 if (File.Exists(fullPath))
 {
  return Assembly.LoadFrom(fullPath);
 }
 return null;
}
```

3.3.3 Přístup k již spuštěné instanci TiaPortal

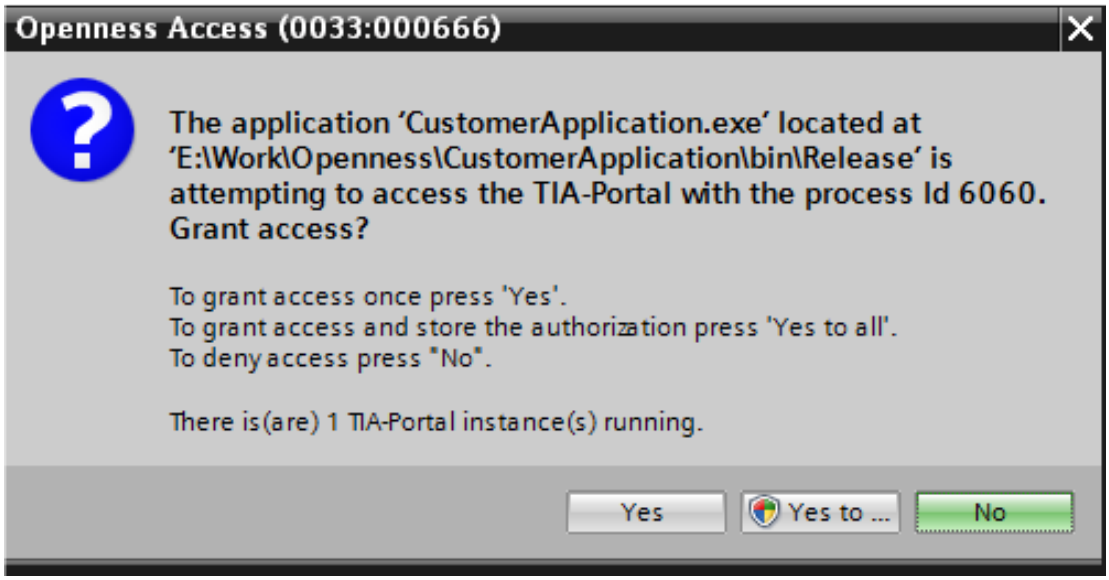
Jak spustit TIA Portal již víme. Spuštěných instancí však můžeme mít i více, a to s i bez spuštěného uživatelského rozhraní. Pokud známe ID požadované instance, můžeme jej využít k přístupu. Mezi jednotlivými procesy procházíme cyklem[6, str. 76]:

Výpis 3.4: přístup k již spuštěné instanci TiaPortal.

```
foreach (TiaPortalProcess tiaPortalProcess
   in TiaPortal.GetProcesses())
{
//...
}
```

3.3.4 Openness firewall

Po připojení k instanci se vás pokusí zastavit firewall TIA Portalu, přístup musí uživatel povolit a potvrdit zprávu na následujícím obrázku[6, str. 77]:



Obr. 3.1: Reakce firewallu na přístup aplikace k instanci TIA Portalu

3.4 Práce s projekty

3.4.1 Otevření projektu

Pokud máme spuštěný a připojený TIA Portal, můžeme se posunout dále k otevření projektu. K tomu se využívá metody *Projects.Open()*, do které je nutné přidat cestu k samotnému projektu. Pokud otevíráme projekt napsaný ve staré verzi TIA Portalu, můžeme použít metodu *OpenWithUpgrade()*. Pro práci s projektem využijeme třídu Project[6, str. 96]:

Výpis 3.5: Otevření projektu metodou Open().

```
Project project =tiaPortal.Projects.Open(new FileInfo(@"D:\
    Some
\Path\Here\Project.apXX"));
```

3.4.2 Uložení projektu

Projekt lze uložit pomocí metody *project.Save()*[6, str. 116].

3.4.3 Kompilace projektu

Pokud jste otevřeli, svým kódem upravili a uložili projekt, následuje kompilace, kterou API umožňuje. Nutnou podmínkou je, že všechna zařízení zahrnutá v projektu jsou „offline." Výsledek kompilace je navrácen jako objekt, podle typu objektu bude provedena hardwarová (HW) nebo softwarová (SW) kompilace. Podporovány jsou následující objekty[6, str. 110]:

- Device – HW a SW
  - Device s ochranou CPU – SW
- DeviceItem – HW
- CodeBlock – SW
- DataBlock – SW
- HmiTarget – SW
- PlcSoftware – SW
- PlcType – SW
- PlcBlockSystemGroup – SW
- PlcBlockUserGroup – SW
- PlcTypeSystemGroup – SW
- PlcTypeUserGroup – SW

Pro kompilaci lze použít následující kód, dle potřeby jej konfigurujte[6, str. 111]:

Výpis 3.6: kompilace instance PlcSoftware

```
public static void CompilePlcSoftware(PlcSoftware plcSoftware
    )
{
 ICompilable compileService
  = plcSoftware.GetService<ICompilable>();
 CompilerResult result = compileService.Compile();
}
```

Výpis 3.7: kompilace instance CodeBlock

```
public static void CompileCodeBlock(PlcSoftware plcSoftware)
{
 CodeBlock block
  = plcSoftware.BlockGroup.Blocks.Find("MyCodeBlock")
    as CodeBlock;
 if (block != null)
 {
  ICompilable compileService
   = block.GetService<ICompilable>();
  CompilerResult result = compileService.Compile();
 }
}
```

Dále je nutné získat výsledek kompilace. V příloze ve výpisu **??** je příklad pro konzolovou aplikaci, výsledek si však můžeme uložit do svého souboru nebo na stavový řádek naší aplikace[6, str. 112].

3.4.4 Vytvoření prázdného projektu

Z Openness aplikace lze také vytvořit nový prázdný projekt. V tomto projektu si můžeme vytvořit nové zařízení, přidat mu novou tabulku tagů a zkopírovat si sem bloky z jiného projektu. K tomuto slouží metoda *Create()*[6, str. 100] pod instancí typu ProjectComposition. Jako argumenty vyžaduje instanci typu DirectoryInfo, která obsahuje odkaz na místo, kde se mají vytvořit soubory pro nový projekt, a jméno projektu. Jméno projektu se zadává jako datový typ *string*.

Já budu pracovat v již předvytvořeném projektu, proto se vytvářením nových dále nebudeme zabývat.

3.4.5 Nastavení TIA Portalu

Public API umožňuje přístup k některým nastavení samotného prostředí TIA Portal[6, str. 101].

Jsou to:

- jazyk uživatelského rozhraní *UserInterfaceLanguage*
- vyhledávání v projektu (lze jej takto zakázat nebo povolit) *SearchInProject*

Přistupovat k nim lze z instance typu TiaPortal. Zabývat se jimi však dále nebudeme.

3.4.6 Knihovny

Knihovny lze použít k uložení typů, jejich předchozích verzí, kopií již existujících bloků nebo kopií technologických objektů a zařízení. V našem případě je využijeme jako zdroj pro vytvoření *Axis_1* a zařízení pro HMI(Human machine interface - rozhraní člověk-stroj).

Přístup ke knihovnám

Typy globálních knihoven: [6, str. 121]

- Systémové: tyto globální knihovny jsou zahrnuty již v instalaci TIA Portalu a využívají koncovky .as14. Slouží pouze ke čtení.
- Podnikové (Corporate global library): tyto jsou vybrány správcem, aby byly nahrány spolu se spuštěním TIA Portalu. Slouží pouze ke čtení.
- Uživatelské: Tyto knihovny mohou být otevřeny v módu pouze pro čtení, ale i pro čtení a zápis. Pokud byla tato knihovna vytvořena ve staré verzi TIA Portal, lze z ní pouze číst.

Dostupnost knihoven: Informace o dostupnosti knihoven zpřístupní třída typu *GlobalLibraryInfo*. Získat tyto informace lze díky metodě *tiaInstance.GlobalLibraries.GetGlobalLibraryInfos()*. Přistupovat k parametrům jednotlivých knihoven můžeme pomocí cyklu:

Výpis 3.8: Cyklus pro získání informací o knihovnách

```
var availableLibraries =
 tiaInstance.GlobalLibraries.GetGlobalLibraryInfos();
foreach (GlobalLibraryInfo info in availableLibraries)
{
 info./*vlastnost*/;
}
```

Práce s knihovnami

Otevření knihovny, uložení a zavření knihovny: Pro práci s knihovnami používejte třídu *GlobalLibrary* nebo *UserGlobalLibrary*, pokud jde o uživatelskou knihovnu. Pro otevření metodu obsaženou v námi otevřené instanci TiaPortal *tia.GlobalLibraries.Open(...)* nebo *tia.GlobalLibraries.OpenWithUpgrade(...)*.

Pro uložení vaší práce a zavření použijte metody *userLib.Save()* a *userLib.Close()*. Pamatujte, že zapisovat lze pouze do uživatelských knihoven.

Procházení otevřenými knihovnami: K tomuto opět využijte cyklu *foreach*[6, str. 121]:

Výpis 3.9: Cyklus pro procházení knihovnami

```
foreach (GlobalLibrary globLib in tiaInstance.GlobalLibraries
    )
{
//práce s knihovnami
}
```

Přístup ke složce s master kopiemi: Co je to master copy si vysvětlíme záhy, nyní však k samotnému přístupu do této složky. Přistupovat k ní můžeme jako k vlastnosti instance projektu: *project.ProjectLibrary.MasterCopyFolder*, odkaz na ni můžeme uložit do instance typu MasterCopySystemFolder. Pojmenujme ji masterCopyFolder [6, str. 128].

3.4.7 Master copy

Z některých objektů lze vytvořit tzv. Master copy, což je obraz objektu v knihovně, ze kterého lze zpětně vygenerovat objekt nový. V tomto obraze bohužel není možné měnit strukturu objektů, lze toho tak využít například k rychlému kopírování datových bloků nebo tagů.

Vytvoření master copy z bloku

V tomto případě poněkud předbíháme, protože přístup k blokům bude řešen až v další sekci. K práci s master copy lze využít třídu MasterCopy. Vytvořit ji lze metodou obsaženou v instanci typu MasterCopySystemFolder:
masterCopyFolder.MasterCopies.Create(zdrojpromastercopy).

Zdroji pro master copy se mohou stát: [6, str. 142]

- Device - HW
- DeviceItem - HW
- DeviceUserGroup - HW
- CodeBlock - SW
- DataBlock - SW
- PlcBlockUserGroup - SW
- PlcTag - SW
- PlcTagTable - SW
- PlcTagTableUserGroup - SW
- PlcType - SW
- PlcTypeUserGroup - SW
- VBScript - HMI

- VBScriptUserFolder - HMI
- Screen - HMI
- ScreenTemplate - HMI
- ScreenTemplateUserFolder - HMI
- ScreenUserFolder - HMI
- Tag - HMI
- TagTable - HMI
- TagUserFolder - HMI

Vytvoření bloku z master copy

K tomuto lze využít metody *CreateFrom(mastercopy)*, která je obsažena v příslušné vlastnosti objektu, kterým takto lze manipulovat, například: *Device newDevice = project.Devices.CreateFrom(copyOfDevice)*[6, str. 145].

3.5 Práce se zařízeními

3.5.1 Editor pro zařízení a sítě

Jde o editor uživatelského rozhraní TIA Portal, ve kterém jsou vytvářeny programy atd. Z TIA Portal Openness jej samozřejmě přímo ovládat nelze, pokud je však rozhraní spuštěno, lze se do něj pomocí naší aplikace přepnout a zobrazit v něm některé položky. Například zařízení, tabulku tagů atd. Slouží k tomu metody *project.ShowHwEditor()* a *device.ShowInEditor()*.

Tyto metody lze zavolat s jedním parametrem a to[6, str. 155]:

- *View.Topology*
- *View.Network*
- *View.Device*

3.5.2 Připojení k zařízení

K tomuto je třeba si vytvořit novou instanci typu *Device*. Poté lze zařízení vytvořit (*Project.Devices.Create(IdentifikátorModeluZařízení, JménoZařízení)*), nebo se k němu připojit metodou *First()*, nebo *Find("JménoZařízení")*. [6, str. 208]

3.5.3 Hardwarová konfigurace

Většina jednoúčelových systémů se skládá z PLC a vstupně/výstupních karet. Přidání lze docílit metodou *DeviceItem HardwareObject.PlugNew()*.

Do některých instancí typu *DeviceItem* lze přípojný modul přidat, ale u některých to není možné. Toto se dá determinovat pomocí metody *bool HardwareObject.CanPlugNew()*.

Oběma těmto metodám se předávají 3 parametry:

- *string typeIdentifier* = Order number zařízení, např.:"OrderNumber:6ES7 232-4HA30-0XB0/V1.0"
- *string name* = jméno modulu, pokud zadáme *null* přidá se implicitní jméno
- *int positionNumber* = pozice v racku vůči PLC

3.5.4 Přístup k software PLC

Pokud chceme přistupovat k blokům, tabulkám tagů nebo kompilovat program, musíme se dostat k instanci, která nám toto umožní. Taková instance je typu - PlcSoftware. Podmínkou je přiřazené zařízení k projektu a přístup k jeho instanci. Použijeme k tomu metodu *GetService<>()*. Výsledný kód pak vypadá takto:

Výpis 3.10: Získání instance pro přístup k software zařízení PLC

```
var target =
 ((IEngineeringServiceProvider)devItem).GetService<
    SoftwareContainer>();
 if (target != null && target.Software is PlcSoftware)
 {
  return (PlcSoftware)target.Software;
 }
```

3.5.5 Přístup k obsahu HMI

Pro práci s vizualizací je nutné získat odkaz na její Software obsah. Toto je analogické k získání PLC software. Je k tomu tedy třeba metoda na získání služby: *GetService<>()*[6, str. 157].

Výpis 3.11: Získání instance pro přístup k software zařízení HMI

```
var target =
 ((IEngineeringServiceProvider)devItem).GetService<
    SoftwareContainer>();
 if (target != null && target.Software is HmiTarget)
 {
  return (HmiTarget)target.Software;
 }
```

3.6 Práce se sítěmi

3.6.1 Vytvoření nové subsítě

Po vytvoření zařízení je třeba je spojit do sítě. Taková síť musí být vytvořená pomocí metody *Create()*. Metodě je třeba předat parametry *string typeIdentifier* a *string name*. Jako identifikátory typu subsítě se dají použít následující výrazy[6, str. 166]:

- System:Subnet.Ethernet
- System:Subnet.Profibus
- System:Subnet.Mpi
- System:Subnet.Asi

K sítím se přistupuje z kompozice *Subnets* v instanci typu Project. Využít k tomu můžeme například cyklu *foreach*, nebo některé z metod přístupu, jako například *First(), Find()*

3.6.2 Spojení zařízení

Manuální spojení v TIA Portalu se provádí spojením ("drag and drop") dvou portů zařízení.

V TIA Openness se toto provádí podobně. Důležitou součástí je získání DeviceItemu, který podporuje službu typu NetworkInterface. Instanci, která se bude odkazovat na požadovaný interface získáme pomocí generické metody GetService<>().

Následně je třeba se spojit s potřebnou instancí. K tomu využijeme třídu Node, která implementuje metodu *void Node.ConnectToSubnet()*. Jako parametr jí musíme předat danou subsíť. Na subsíti tak vznikne nový uzel, který odpovídá portu daného zařízení.

3.6.3 Atributy instance typu Node

Atributy jsou odlišné podle typu uzlu. V našem případě se budeme bavit o uzlu typu Ethernet.

Uvedeme si ty nejdůležitější [6, str. 179]

- string Name - jméno uzlu (jen pro čtení)
- string MacAddress - Mac adresa odpovídajícího portu
- string Address - IPv4 adresa portu
- string SubnetMask - maska podsítě

3.7 Tabulky tagů

3.7.1 Přístup k tabulce tagů

Pro získání přístupu je třeba vytvořit instanci, do které uložíme kompozici tabulky: *PlcTagTableComposition tagTables = plcSoftware.TagTableGroup.TagTables*. V této instanci poté můžeme použít metody *Find()*, *First()*, popřípadě procházet skrze tuto kompozici cyklem *foreach*. Odstranit tabulku lze metodou *Delete()*[6, str. 327-335].

3.7.2 Přístup k jednotlivým tagům

Přístup k jednotlivým tagům v tabulce poskytuje vlastnost instance *tagTable.Tags*. Ta obsahuje seznam existujících tagů, procházet jimi proto můžeme cyklem, popřípadě použít metod *Find()*, *First()* atd.

Nástroje pro práci s nimi nám poskytuje třída *PlcTag*. API nám povoluje přístup k atributům tagů, upravovat, přidávat a vytvářet nové můžeme přímo z naší aplikace. Tyto metody mají přetížení s návratovou hodnotou typu *PlcTag*, je tak možné vytvořený tag přímo přiřadit do instance pro další práci s ním. [6, str. 336]

3.7.3 Atributy tagů

Jak již jsme se již zmínili, z API je možné upravovat atributy tagů. Tyto atributy jsou[6, str. 336]:

- Name (read only)
- Data type name
- Logical address
- Comment
- ExternalAccessible
- ExternalVisible
- ExternalWritable

3.7.4 Konstanty

Veřejná API umožňuje i přístup ke konstantám, které jsou obsažené v tabulkách tagů. Přístup a práce s nimi je podobná jako u tagů. Rozdíl je, že s nimi pracujeme pomocí třídy *PlcUserConstantComposition* a *PlcUserConstant*. Přistupovat můžeme i k systémovým konstantám, vytvářet a mazat však můžeme pouze uživatelské.

Konstanty mají následující vlastnosti:[6, str. 337]

- Jméno (Name - pouze pro čtení)
- Datový typ (Data type name)

- Hodnota (Value)

3.8 Bloky

Tato část bude nejdůležitější. Spustili jsme TIA Portal, připojili se k zařízení a vytvořili instanci pro práci se software PLC, nyní máme přístup k blokům, které tvoří program.

Nejdříve si popíšeme, jak lze s bloky provádět jednoduché operace, jak je exportovat do XML souboru a jak z tohoto souboru vygenerovat blok nový. Poté se budeme zabývat samotnou strukturou exportovaného souboru. To je předpoklad pro generování programů, které nebudou předpřipravené, ale automaticky generované například externím generátorem kódu jazyce XML.

3.8.1 Skupiny bloků

Programové bloky je možné seskupovat do skupin. Každé zařízení má jednu systémovou skupinu, která sdružuje všechny ostatní skupiny a bloky. Přistupovat k ní můžeme instancí typu *PlcBlockSystemGroup*. Tu získáme jako vlastnost instance *plcSoftware.BlockGroup*.

Ke skupině můžeme přistupovat přes instanci *PlcBlockUserGroupComposition*[6, str. 272]:

Výpis 3.12: Přístup ke skupinám bloků

```
PlcBlockUserGroupComposition userGroupComposition =
  plcsoftware.BlockGroup.Groups;
PlcBlockUserGroup plcBlockUserGroup =
 userGroupComposition.Find("MyUserfolder");
```

3.8.2 Přístup k blokům

Pokud přistupujeme k blokům v systémové skupině, stačí použít metodu *plcsoftware.BlockGroup.Blocks.Find("MyBlock")* popřípadě *First()* a přiřadit ji do instance požadovaného typu, základně *PlcBlock*.

Pokud víme o jaký typ bloku konkrétně jde, lze PlcBlock přetypovat na:

- *Codeblock*
  - *FB* - funkční blok
  - *FC* - funkce
  - *OB* - organizační blok
- *DataBlock* - bloky pro proměnné
  - *ArrayDB*

– *GlobalDB*
– *InstanceDB*

Docílíme tak získání přístupu ke specifickým atributům, které tyto typy poskytují.

Mazání bloků: Mazat bloky lze metodou *block.Delete()*. Po vytvoření kopií tak můžeme vytvořit čistý projekt bez konfliktů.

Podporované formáty bloků jsou:

- STL - generuje se soubor s koncovkou .awl
- SCL (Structured Control Language - strukturovaný řídící jazyk)
- DB (datový blok)
- UDT (User Defined Types - Uživatelem definované datové typy)

Přístup se provádí pomocí instance typu *PlcExternalSource*. Generovat pak lze soubory metodou *plcExternalSource.GenerateBlocksFromSource()*[6, str. 285].

Výpis 3.13: Ukázka externího souboru generovaného z formátu STL(*.awl)

```
FUNCTION_BLOCK "program"
{ S7_Optimized_Access := 'TRUE' }
VERSION : 0.1

BEGIN
NETWORK
TITLE =
      CALL "Block_1", "Block_1_DB"
      (  activate_i                  := "activate" ,
         y                           := "a"
      );

END_FUNCTION_BLOCK
```

3.9 Export a import bloků

Export bloků do souborů formátu XML poskytuje daleko větší variabilitu, co se týče generování programu. Podporují totiž i jazyky FBD, LAD a GRAPH [6, str. 377].

V následující sekci si popíšeme strukturu takového XML souboru.

3.9.1 Export bloku

Nejdříve je však nutné provést samotný export. K tomu slouží metoda *Export()*. Jako argumenty je nutné dodat instanci typu *FileInfo*, která bude obsahovat cestu k cílové složce a parametr *ExportOptions*, kterým můžeme výslední dokument uzamknout jen pro čtení (*ExportOptions.WithReadOnly*) nebo s defaultními hodnotami (*ExportOptions.WithDefaults*)[6, str. 436].

3.9.2 Import bloku

Metodu *Import()* lze zavolat z instance *PlcBlockComposition*. Jako argumenty požaduje cestu ke zdrojovému XML souboru ve formě instance typu *FileInfo* a parametr *ImportOptions*, kterým jde nastavit, zdali se má již existující blok se stejným jménem přepsat (*ImportOptions.Override*) či má metoda předat výjimku (*ImportOptions.None*). [6, str. 481]

3.9.3 Struktura XML souboru

Každý XML dokument se obecně skládá z několika částí, a to ze záhlaví a těla dokumentu. V záhlaví jsou informace o kódování a verzi jazyka, tělo je však tím zásadním.

Struktura těla XML souboru odpovídá určitým definovaným schématům, všechna najdeme v adresáři:
C:\Program Files\Siemens\Automation\Portal V15\PublicAPI\V15\Schemas

Skládá se ze dvou hlavních částí:[6, str. 434]

- Interface - Jde o seznam vstupních, výstupních a dalších proměnných, jejich datových typů a pokud jde o konstanty, tak i jejich hodnot. Využívá schématu SW.InterfaceSections_v2.xsd.
- Compile unit - Zde musíme vybrat správné schéma podle jazyka, který má cílový blok využívat:
  - GRAPH -> SW.PlcBlocks.Graph.xsd
  - LAD/FBD -> SW.PlcBlocks.LADFBD.xsd
  - STL -> SW.PlcBlocks.STL.xsd

Soubor k funkčnímu bloku

Nyní si ukážeme jak vypadá vygenerovaný XML soubor z prázdného funkčního bloku.

Po vygenerování tohoto souboru si můžeme všimnout základní struktury a sice dvou částí:

- DocumentInfo - obsahuje informace o stavu systému, kterým byl tento blok vygenerován, jako datum, parametr nastavení exportu *ExportSetting* a instalované produkty.
- Kompozice bloku - SW.Blocks.FB, ta má ID="0"- ID je hexadecimální číslo. Musí být unikátní, ale nezáleží na pořadí, jak je přiřazováno.

Kompozice bloku obsahuje seznam objektů a seznam atributů. Tyto atributy jsou dostupné i z veřejné API. Seznam objektů je dostupný pouze v tomto souboru.

Seznam objektů v prázdném funkčním bloku obsahuje pouze objekty textu komentáře a titulku bloku. Titulek bloku je vždy na konci.

Výpis 3.14: Seznam objektů prázdného funkčního bloku

```
<ObjectList>
 <MultilingualText ID="1" CompositionName="Comment">
  <ObjectList>
   <MultilingualTextItem ID="2" CompositionName="Items">
    <AttributeList>
     <Culture>en-US</Culture>
     <Text>TOTO JE KOMENTÁŘ BLOKU</Text>
    </AttributeList>
   </MultilingualTextItem>
  </ObjectList>
 </MultilingualText>
 <MultilingualText ID="3" CompositionName="Title">
  <ObjectList>
   <MultilingualTextItem ID="4" CompositionName="Items">
    <AttributeList>
     <Culture>en-US</Culture>
     <Text>TOTO JE TITULEK</Text>
    </AttributeList>
   </MultilingualTextItem>
  </ObjectList>
 </MultilingualText>
</ObjectList>
```

Objekty volání funkčních bloků Jak již víme, každý LAD program se skládá z jednotlivých "sítí", které mají podobu větve paralelního obvodu, který má LAD napodobovat. Novou síť přidáme jako *SW.Blocks.CompileUnit*, která má dva parametry. Těmi jsou ID a CompositionName, podobně jako u objektu titulku viz. výpis 3.14.

Výpis 3.15: Seznam objektů prázdného funkčního bloku

```
<SW.Blocks.CompileUnit ID="A" CompositionName="CompileUnits">
        <AttributeList>
           <NetworkSource>
            ...zde je prostor pro definování obsahu sítě...
           </NetworkSource>
 <ProgrammingLanguage>LAD</ProgrammingLanguage>
        </AttributeList>
        <ObjectList>\dots</ObjectList>
      </SW.Blocks.CompileUnit>
```

Obsah sítě vzhledem k tomu, že nejjednodušší konfigurace sítě je "natvrdo"připojená cívka, předvedeme si jednoduchou konfiguraci sítě na ní.

Bude třeba si vytvořit seznam součástek (Parts) a propojení (Wires)

Obr. 3.2: Jednoduchá kofigurace sítě

Tato jednoduchá síť vyžaduje pouze dvě spojení (viz. obr. 3.2). První bude obsahovat označení <Powerrail/> a označení pro vstup do "cívky"a druhé spojení propojí cívku a proměnnou "a". Výsledný kód pak bude vypadat takto:

Výpis 3.16: Jednoduchá konfigurace sítě

```
<NetworkSource>
<FlgNet
xmlns=
"http://www.siemens.com/automation/Openness/SW/NetworkSource/FlgNet
   /v1">
<Parts>
 <Access UId="21" Scope="GlobalVariable">
 <Part Name="Coil" UId="22"/>
</Parts>
<Wires>
 <Wire UId="23">
  <Powerrail/>
  <NameCon Name="in" UId="22"/>
 </Wire>
 <Wire UId="24">
  <IdentCon UId="21"/>
  <NameCon Name="operand" UId="22"/>
 </Wire>
</Wires>
</FlgNet>
</NetworkSource>
```

Všimněte si, že je na jednotlivé součástky referováno pomocí UId.

4 Program pro ovládání modelu jeřábu

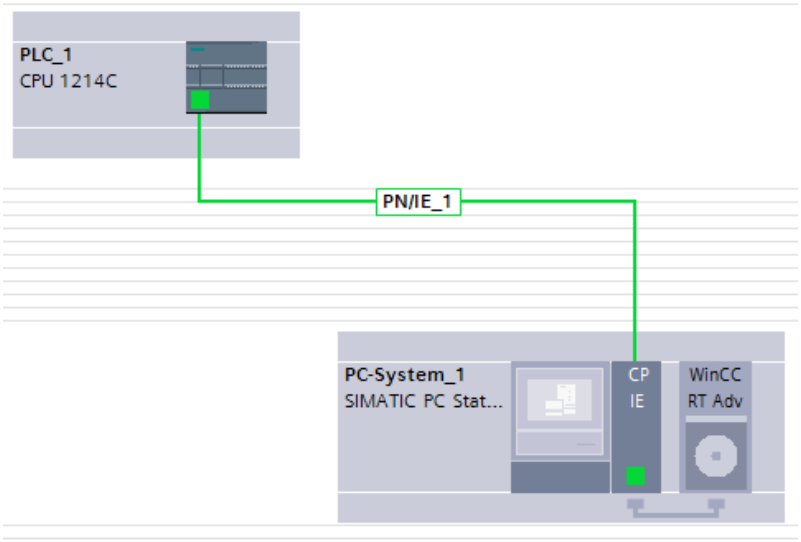
V této kapitole popíšu, jak funguje vzorový program pro ovládání modelu Jeřábu. Program implementuje algoritmus pro skládání hlavolamu hanojských věží.

4.1 Hardwarová konfigurace

Pro realizaci programu jsem měl k dispozici PLC s CPU 1214C DC/DC/DC s IO moduly AQ1 signal board s firmware ve verzi 1.0 a SM 1234 AI4/AQ2 s firmware 2.0, které poskytují dostatečné množství vstupů a výstupů pro model.

Pro HMI jsem využil WinCC RT Advanced, který mi pomocí WinCC runtime dovolil spustit toto rozhraní z počítače.

HMI a PLC jsou spojeny pomocí PROFINET subsítě, komunikace je tedy adresována pomocí PORFINET jmen.



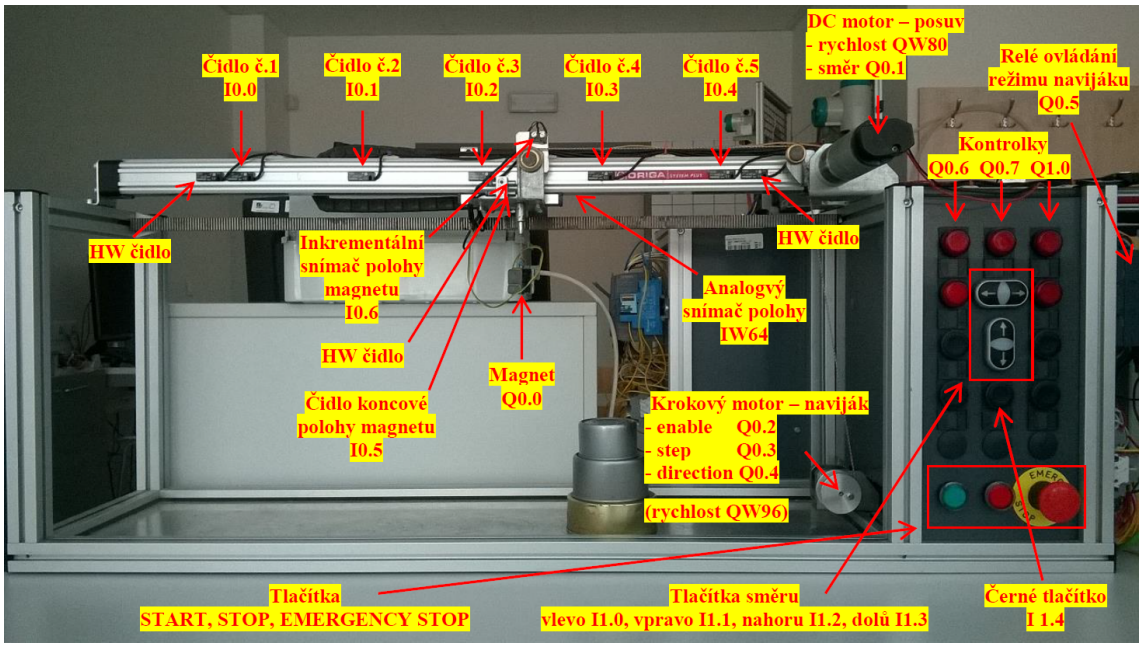
Obr. 4.1: Konfigurace PLC

Na obrázku 4.2 si může čtenář prohlédnout model jeřábu.[8]

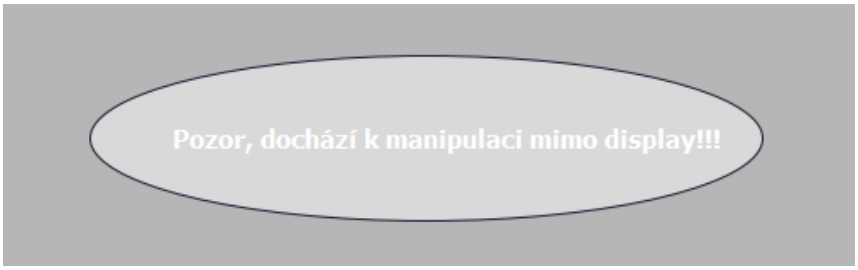
4.2 Manuální ovládání

Manuální ovládání může být blokováno virtuálním tlačítkem, které povolí automatický mód běhu. Protichůdné přikazy jsou vyblokovány. Pohyb je zablokován čidly č.1 a 5 na okrajích pojezdu. Viz. obr. 4.2

Při používání fyzických tlačítek směru je aktivováno upozornění na displeji HMI.



Obr. 4.2: Popis pracoviště



Obr. 4.3: Upozornění na virtuálním paneluupozorneniHMI

4.3 Ovládání z HMI

Než popíšu algoritmus hanojských věží, projdu ovládací rozhraní, skrze které se algoritmus spustí. Odpovídající obrázek je v příloze: C.1

- 1. oblast - odsud je možnost pomocí virtuálních tlačítek provádět manuální operace s jeřábem. Případné protichůdné příkazy jsou vyblokovány.
- 2. oblast - tato skupina tlačítek umožňuje nastavení automatického chodu zařízení
  - Tlačítko “Mód” umožňuje přepínat mezi automatickým a manuálním módem.
  - Tlačítky pro startovací a koncové polohy se konfiguruje chod automatu. Pokud jsou tyto polohy stejné, algoritmus je přeskočen na konec.

  - Start - toto tlačítko nastaví stavový automat do výchozího stavu a povolí jeho běh
  - Pauza - pozastaví běh automatu
- 3. oblast je vizualizací, ukazující aktuální polohu magnetu a jednotlivých částí hlavolamu.

4.4 Stavový automat

Toto je hlavní část automatického módu programu. Řeší algoritmus hanojských věží. Jeho pravidla jsou následující:

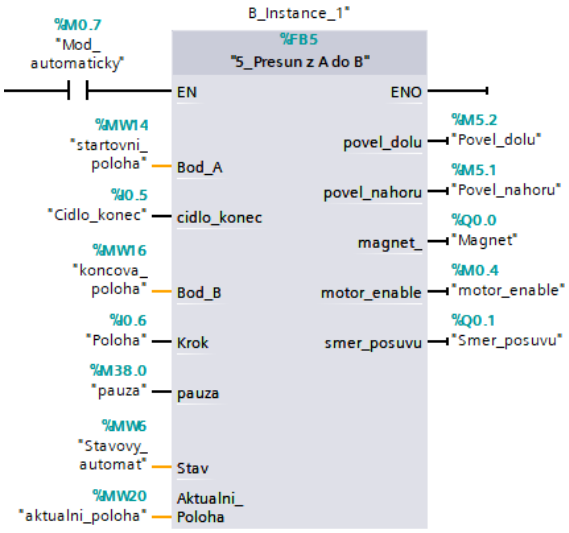
- V jednom tahu lze přemístit pouze jeden kotouč.
- Jeden tah sestává z vzetí vrchního kotouče z některé věže a jeho položení na vrchol jiné.
- Je zakázáno položit větší kotouč na menší.

V mém případě jsou k dispozici 3 kovové kotouče, řešení proto má 7 tahů. Stavů použijeme 11. Definujeme li počáteční polohu, jako polohu **A**, pomocnou, polohu jako polohu **B** a koncovou polohu, jako polohu **C**, potom jsou jednotlivé stavy:

- 0. stav - kontrola a nastavení proměnných
- 1. stav - přesun do referenční polohy zcela vlevo, zcela nahoře
- 2. stav - řesun nejmenšího kotouče z A do C
- 3. stav - přesun středního kotouče z A do B
- 4. stav - přesun malého kotouče z C do B
- 5. stav - přesun velkého kotouče z A do C
- 6. stav - přesun malého kotouče z B do A
- 7. stav - přesun středního kotouče z B do C
- 8. stav - přesun malého kotouče z A do C
- 9. stav - přesun do referenční polohy zcela vlevo, zcela nahoře
- 10. stav - reset pracovních proměnných

Jednotlivé stavy jsou prováděny sekvenčně za sebou, tak jak byly popsány výše.

Obsluhu přesunu a magnetu provádí funkční blok “5_Presun z A do B,” kterému jsou jen předány vstupní a výstupní parametry. Příklad jeho viz. obr. 4.4.



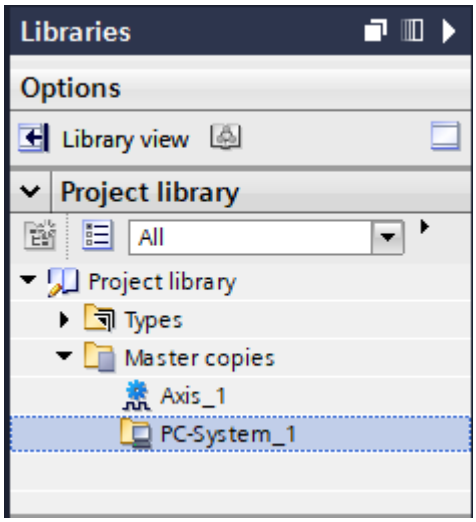
Obr. 4.4: Volání funkčního bloku “5_Presun z A do B”

5 Automatický generátor

Tento program napsaný v jazyce C# na platformě .NET, ovládaný pomocí jednoduché formulářové aplikace ve Windows presentation foundation - knihovna tříd pro tvorbu grafického rozhraní je určený pro generování programu pro ovládání modelu jeřábu. Dále se budeme věnovat popisu činnost této aplikace.

5.1 Pracovní projekt v TIA Portalu

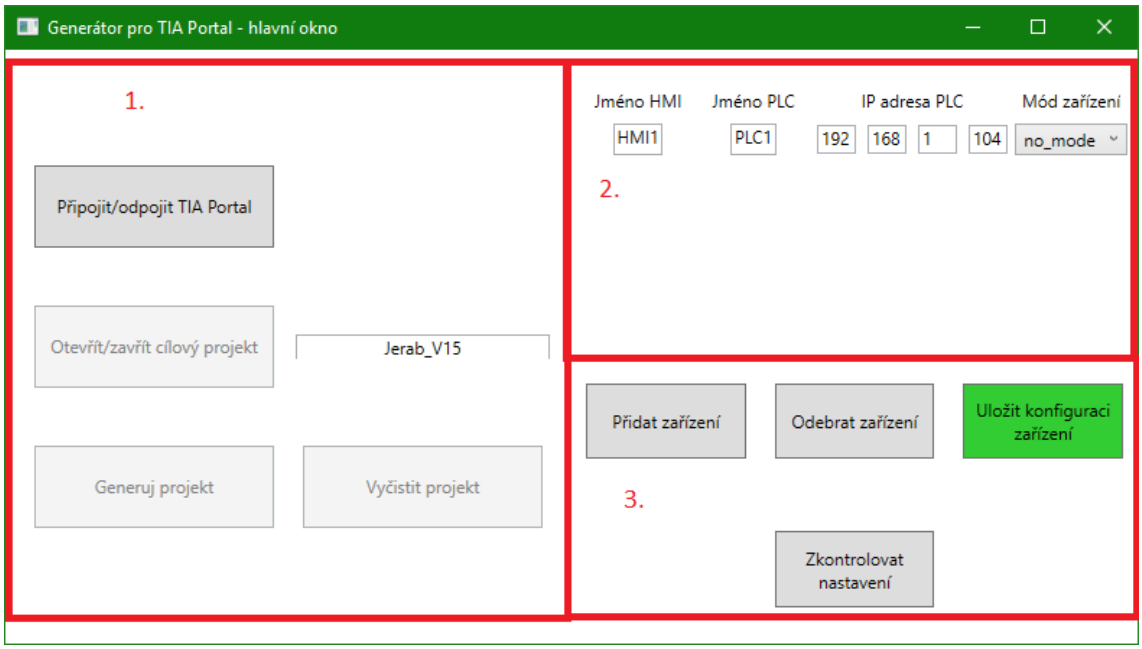
Kvůli nemožnosti vytvoření některých objektů pomocí dedikovaných metod(tím se budeme zabývat dále v textu), jsem musel pracovat v předpřipraveném projektu. Tento projekt obsahuje ve své knihovně *MasterCopy* technologického objektu Axis_1 a zařízení pro HMI.



Obr. 5.1: Knihovna v pracovním projektu

5.2 Prezentační vrstva

Tato aplikace se skládá ze dvou částí. Tou první je prezentační vrstva ve formě formuláře napsaného ve WPF(Windows presentation foundation - knihovna tříd pro tvorbu grafického rozhraní).



Obr. 5.2: Prezentační vrstva demo aplikace

Jak je znázorněno na obr. 5.2, formulář se skládá z několika oblastí, které obsluhují jednotlivé části programu.

- 1. oblast slouží pro obsluhu cílového projektu a obsahuje prvky:
  - Tlačítko *Připojit/odpojit TIA Potral* buď spustí novou instanci tiaportalu, nebo se připojí k první instanci, která je k dispozici.
  - Tlačítkem *Otevřít/zavřít projekt* se otevře nebo zavře v již zmíněné instanci TIA Portalu projekt jehož jméno je napsáno do *TextBoxu* vpravo od tohoto tlačítka.
  - Projekt musí být umístěn v adresáři C:\Users\User\Documents \Automation.
  - Tlačítko *Generuj program* spustí skript, který podle zadané konfigurace v 2. oblasti vygeneruje program.
  - Před generováním projektu se je třeba ujistit jestli je projekt zcela prázdný (vyjímaje knihovnu), to zajistí v prvním kroku již tlačítko *Otevřít/zavřít projekt*. Pokud chcete generovat projekt znovu, je třeba použít tlačítko *Vyčistit projekt*.
- 2. oblast slouží pro vytvoření konfigurace pro generování programu do cílového projektu a obsahuje prvky uspořádané do matice, projděme si je po sloupcích
  - Do pole pro název HMI se zadává PROFINET jméno pro Human machine interface - rozhraní člověk-stroj se kterým se má HMI zařízení vygenerovat
  - Pole PLC slouží pro zadání PROFINET jména ovládacího PLC

  - IP adresa se zadává po jednotlivých oktetech. Jak vidíte jedná se o IPv4. Jednotlivá pole hlídají jestli hodnota uvnitř nepřesáhla 255.
  - Mód zařízení je vysouvací menu, které určije v jakém módu se má generovat program do výsledného projektu, může nabývat hodnot:
    * no_mode: Negeneruje se žádný program
    * Full - generuje se kompletní program pro ovládání jeřábu
    * "auto" - generuje se program, který umožňuje jen automatický běh programu pro ovládání jeřábu
    * manual: generuje se program, který umožňuje jen manuální ovládání jeřábu
- 3. oblast umožňuje uložit konfiguraci nastavenou v oblasti 2. a má tlačítkový charakter:
  - Stiskem tlačítka *Přidat zařízení* se přidá řádek pro konfiguraci zařízení
  - Tlačítkem *Odebrat zařízení* odeberete řádek pro konfiguraci zařízení
  - *Uložit konfiguraci* uloží konfiguraci do programové paměti. Tato pamět má formu listu strojů, které obsahují charakteristické vlastnosti. Tím se budeme zabývat dále v textu.

5.3 Uložení konfigurace stroje

Ačkoli by toto na první pohled mohlo vypadat nechronologicky, budeme se nejdříve zabývat 3. oblastí. Před začátkem generování je totiž třeba vytvořit konfiguraci stroje a uložit ji do programové paměti. K tomu využíváme třídy *List<Stroj>*, jako vlastnosti jedné hlavní třídy - **Stroje**. Ta obsahuje dvě členské třídy:**Stroj** a **Funkcionality**.

Uložení obsluhuje událost vyvolaná kliknutím na tlačítko "Uložit konfiguraci," ukládající celou konfiguraci naráz pomocí jednoho z konstruktorů třídy Stroj.

5.3.1 Třída Stroj

Každý jeden řádek v konfiguraci vlastně znamená jeden stroj, který je nakonfigurován. Stroj je celý jeden model sestávající z PLC, modelu a HMI. Třída Stroj implementuje jeho vlastnosti, jako je PROFINET jméno PLC a HMI, IP adresa PLC a mód funkce celého modelu, také implementuje jednu obslužnou metodu a tři konstruktory.

Vlastnosti

Tato třída implementuje 5 vlastností, jedna je tedy navíc. Krom té jedné jsou všechny typu string. Tato vlastnost se týká IP adresy a jde o pole celočíselných hodnot, syn-

chronizuje se s podobnou vlstností, která se jmenuje Ipaddress_string, tuto používáme dále v kódu generátoru. Umožňuje tak nastavování IP adresy po jednotlivých bytech.

Konstruktory

V kódu jsem využil v podstatě jen jeden, protože ukládám všechny parametry stroje najednou.

Tento konstruktor má 4 parametry:

- string aJmeno - jde o PROFINET jméno pro HMI
- string aIPaddress - zadává IP adresu jako string a to celou najednou
- string aPLC - PROFINET jméno PLC
- string aMod - mód nahraného programu pro PLC

Pokud je některý předaný parametr **null**, konstruktor vrátí výjimku ArgumentNullException, která obsahuje jméno parametru, který ji vyvolal.

5.3.2 Třída Funkcionality

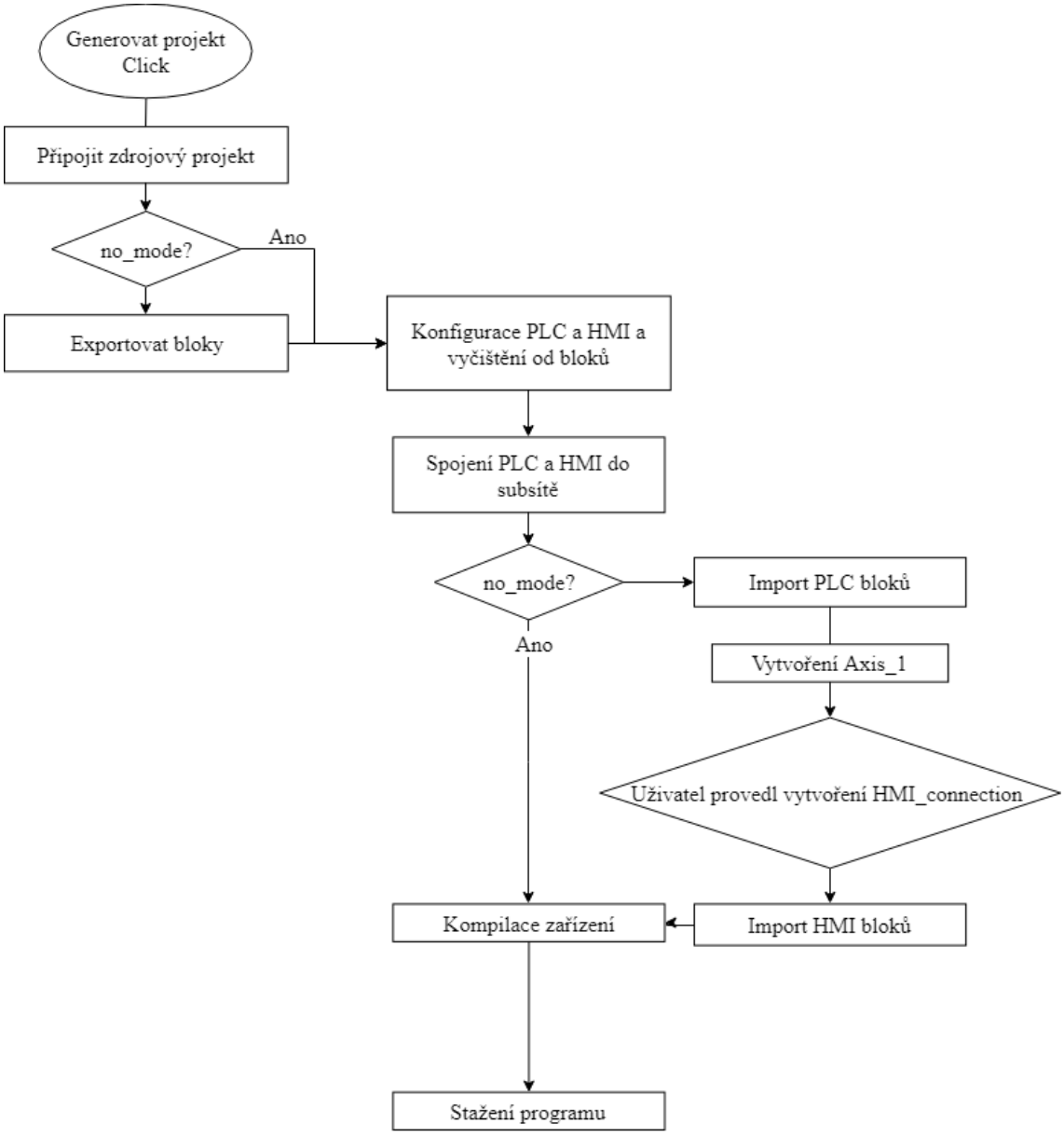
Třda Stroj obsahuje několik vlastností, jednou z nich je mód, ve kterém se má generovat výsledný program pro PLC. Ten je v podstatě přenesen ze zdrojového projektu a v třídě Funkcionaliny je implementována jako vlastnost instance Funkcionalita typu Dictionary<string,string>.

Třída Funkcionality implementuje tři konstruktory. Já jsem z nich využil dva a to implicitní, kdy se vytvoří základní sada klíčů a hodnot ve slovníku Funkcionalita a explicitní, kdy je vytvořena nejen základní sada, ale přidá se ještě jedna speciální hodnota, podle toho jak ji zadá uživatel tohoto konstruktoru. Toho využívám zde: 5.6.1.

Další konstruktor, který Třída obsahuje je kopykonstruktor, jehož funkcí je zkopírovat sadu z jiného slovníku.

5.4 Generování programu

V dalších sekcích se budeme zabývat nejdůležitější částí programu a to událostmi spouštěnými tlačítky v 1. části prezentační vrstvy. Projdeme si je jednotlivě, tak jak by je měl použít uživatel aplikace. V diagramu 5.3 je znázorněn průběh hlavní události, kterou je dán povel pro generování programu.



Obr. 5.3: Zjednodušený vývojový diagram generování PLC programu

5.5 Obslužná událost pro tlačítko “Přípojit/odpojit portal”

Tato událost funguje podobně jako bistabilní klopný obvod typu D. Jeho stavy jsou: TIA Portal připojen a odpojen a zpětná vazba je zajištěna členskou instancí *TiaPortal cilovyPortal*. Hodnota **null** znamená, že TIA Portal je odpojen.

Aby zde opravdu hodnota null byla zajišťuje událost, *cilovyPortal.Disposed*, která jsem přidělil metodu *Portal_disposed*, která toto zajistí.

Cílový portál zde spouštíme medodou *Run_portal()*, které předáme hodnoty withUI=**true** a poradi_instance=*Destinace.cil*. Tím jsem zajistil, že metoda manipuluje s první instancí, která je k dispozici a že tuto instanci spustí s uživatelským rozhraním.

5.5.1 Obslužná metoda události odpojení cílového TIA Portalu

Tato metoda zajistí, že jsou do členských instancí *cilovyPortal* a *cilovyProjekt* zapsány hodnoty **null**. Toto je důležité pro rychlé rozpoznávání, jestli je cílový TIA Portal připojen nebo odpojen.

Výpis 5.1: Ukázka volání metody pro export.

```
private void Portal_disposed (object sender, EventArgs args)
{
 cilovyPortal = null;
 cilovyProjekt = null;
 btn_connect_dispose_portal.Background = Brushes.OrangeRed;
 Console.Write("Cílový projekt odpojen");
}
```

Odpojení zdrojového portalu obsluhuje podobná metoda. Ta zapisuje do členských instancí určených pro zdrojový TIA Portal.

5.6 Obslužná událost pro tlačítko “Otevřít/zavřít projekt”

V této obslužné metodě pro stisk tlačítka se provádí buď otevření nebo zavření cílového projektu. Zároveň se zde zaregistruje metoda pro událost indikující zavření hlavního okna a ukončení Generátoru a událost indikující vyvolání dialogu pro potvrzení v TIA Portalu. Jde o takový dialog, který vyžaduje jakoukoli odpověď. Jak jsem ji využil si popíšeme zde: 5.7.1

5.6.1 Otevření cílového projektu

Pro tento úkon jsem připravil připravena speciální hodnotu ve slovníku Funkcionality. Najdeme ji pod klíčem "target" a její hodnota obsahuje cestu k zadanému cílovému projektu v políčku napravo od tlačítka "Otevřít/zavřít projekt." Otevření probíhá za podmínky, že instance s odkazem na cílový projekt(*Project cilovyProjekt*) má hodnotu **null**.

5.6.2 Zavření cílového projektu

Za opačné podmínky, tedy, že instance *Project cilovyProjekt* odkazuje na neodpojený projekt se provede zavření projektu a projekt se prohlásí za odpojení nahráním hodnoty **null** do této instance.

5.7 Obslužná událost pro tlačítko "Generuj program"

Na kód této události se lze přesunout v editoru Visual Studio dvojklikem na tlačítko "Generuj program." Na začátku jsem připravil několik kontrol:

- kontrola, je-li přidáno zařízení. K tomuto jsme použili příkaz **if**, protože toto vyhodnocení je rychlejší než výjimka.
- dotaz na uživatele, jestli má být provedeno automatické potvrzení všech dotazů, které provede TIA Portal ohledně kompilace. Nastavuje proměnnou typu bool, kterou zpracovává obslužná metoda události, která se v tomto případě spustí.

5.7.1 Obslužná metoda události pro potvrzení dotazů TIA Portal

Pokud je registrována a vyvolána událost *TiaPortal.Confirmation*, je zavolána tato metoda. O svém zavolání uživatele informuje na příkazovou řádku.

Pokud je členská proměnná *bool confirm_all* nastavena na hodnotu **true**, provede se kontrola, jedná-li se o dotaz týkající se kompilace. Pokud ano, je pomocí parametru typu ConfirmationEventArgs předán výsledek, který daný dotaz v TIA Portalu potvrdí.

Výpis 5.2: Obsluha události TiaPortal.Confirmation

```
private void target_portal_confirmation
(object sender, ConfirmationEventArgs args)
{
 Console.WriteLine
  ("\t\t\tPortal confirmation occured: {0}\n\t{1}\n\n",
   args.Caption,
```

```
    args.Text);
            if (confirm_all)
            {
                if (args.Caption == "Compile")
                {
                    args.Result = ConfirmationResult.Yes;
                    args.IsHandled = true;
                }
            }
}
```

5.7.2 Připojení zdrojového projektu

Připojení k TIA Portalu

Použil jsem způsob spouštění pomocí Konfiguračního souboru metody popsaný v sekci 3.3.1. Metoda *Run_portal()* má dva parametry a to *bool withUI* a *Destinace* poradi_instance

Destinace je výčtový typ, který determinuje, jestli chce uživatel připojovat zdrojový nebo cílový projekt. Může nabývat hodnot:

- zdroj - taková instance je v pořadí až druhá, nebo se spustí jako nová
- cil - taková instance je v pořadí první, v našem případě, pokud se spouští jako nová, tak je spuštěna s UI, jinak se připojí k první, která je k dispozici

Na konzoli se také vytiskne informace o spuštění metody a o počtu spuštěných instancí TIA Portal.

Otevřít zdrojový projekt

Tato část kódu provede kontrolu, zdali je otevřený nějaký projekt. Pokud ano, zavře ho a otevře požadovaný viz. sekce 3.4.1. Pokud v instanci není žádný otevřený projekt, otevře se požadovaný projekt.

Metoda, kterou zde využíváme (*OpenProject()*) předává jako návratovou hodnotu "odkaz"na otevřený projekt. Přijímá dva parametry:

- *TiaPortal Tiaportal* - Instance TIA Portalu, ve které se má projekt otevřít, pokud je **null**, metoda vrací také **null**
- *string project_path* - Jméno projektu, je třeba ji zadat s koncovkou *.ap15

Může se stát, že projekt se zadaným názvem neexistuje. V takovém případě je nadhozena výjimka typu *EngineeringTargetInvocationException*.

5.7.3 Metoda Begin_Export()

Tato metoda obslouží export všech potřebných bloků a grafik z projektu a cestu k nim předá pomocí *out* parametrů:

- *Project Project* - Projekt ze kterého se exportují potřebné bloky
- *out List<string> graphicsPaths* - List cest, které vedou k exportovaným grafikám projektu
- *out List<List<string> > PLC_software* - List listů cest, které vedou k exportovaným blokům
- *out List<Axis_params> axisparams_list* - List parametrů Technologického objektu
- *out List<List<string> > HMI_software* - List listů cest, které vedou k exportovaným blokům

Nejdříve je však vždy nutné provést kompilaci, při kompilaci software TIA Portal automaticky vyřeší případnou nekonzistenci bloků.

Tato metoda vyexportuje vše díky tomu, že grafiky jsou přístupné z kompozice *Project.Graphics*, bloky z kompozice *Blocks*, tabulky tagů z kompozice *TagtableGroup.TagTables* v instanci typu *PlcSoftware*. V případě HMI jsou exportovány obrazovky z kompozice *ScreenFolder.Screens* a tabulky tagů z kompozice *TagFolder.TagTables*.

Tato metoda umí také vyexportovat spojení (kompozice *Connections*). Bohužel pouze ta, která byla vytvořena uživatelem a ne takzvaně integrovaná spojení. To jsou taková spojení, která vznikla asociováním HMI a PLC v TIA Portalu v Devices & networks. Aplikace uživatele sama navede na správný postup, jak takové spojení vytvořit.

Výpis 5.3: Ukázka volání metody pro export.

```
program.Export (new FileInfo(path), ExportOptions.WithDefaults);
```

Metoda sama pozná, jestli se jedná o zařízení typu PLC nebo HMI tak, že zavolá metody *GetPlcSoftware()* a *GetHmiTarget()* a pomocí větvení zkoumá navrácenou hodnotu. Pokud *GetPlcSoftware()*==**null**, pak se jedná o HMI, pokud *GetHmiTarget()*==**null**, jedná se o zařízení typu PLC.

Jednotlivé cesty se pak ukládají do listů typu *List<string>*, ty se pak seřadí do výstupních listů. Třídu *List<T>* používáme pro její metodu *Sort()*. Bloky v PLC jsou pojmenovány tak, aby prvním znakem bylo číslo, které označuje pořadí volání jednotlivých bloků v PLC programu. Pro import je toto klíčová vlastnost, protože přidávání jednotlivých bloků tomuto musí odpovídat.

Závěrem je pomocí nástroje "Linq to XML" provedena úprava názvů HMI spojení v XML souboru, tak aby odpovídaly uživatelem vytvořeným spojením. Proč je to nutné si vysvětlíme v sekci 5.7.7

Výpis 5.4: Konfigurace PLC

```
foreach(string tag_path in HMI_TagTables_list)
{
 //zde provedu úpravu HMI_connection v tagtable
           XDocument xDocument = XDocument.Load(tag_path);
Console.Write("\t\tPřipojuji se na {0} - {1} za účelem úpravy
   spojení...",
 tag_path, xDocument.Root.Name);
           foreach (XElement connection in xDocument.Descendants("
              Connection"))
           {
                       Console.WriteLine("\t\tPopojuji se k
                          elementu {0}", connection.Name);
                    foreach (XElement name in connection.Elements
                       ())
            {
            name.Value = string.Format("HMI_Connection_{0}",
                counter);
                    }
  }
            xDocument.Save(tag_path);
            Console.WriteLine("...hotovo");
}
```

5.7.4 Automatická hardwarová konfigurace

Tato část kódu je vepsána přímo v obslužné metoge pro událost “Generuj program”, protože je poměrně jednoduchá a přehledná. Zároveň je tak možné ji snáze pozměnit při debuggingu programu.

Přidání PLC

Pro přidání PLC jsem využil metody *DeviceComposition.CreateWithItem()*, její parametry jsou:

- string typeIdentifier - identifikační string pro typ zařízení, v našem případě má hodnotu "OrderNumber:6ES7 214-1AG40-0XB0/V4.2"
- string name - PROFINET jméno má hodnotu, kterou zadáte ve formuláři do kolonky PLC
- string devicename - název zařízení, má hodnotu **null**, to znamená, že je PLC přiřazeno implicitně

Je důležité upozornit, že generujeme zařízení s firmwarem V4.2. Je to protože tento firmware podporuje technologické objekty ve verzi 6.0. Viz. sekce 5.7.7

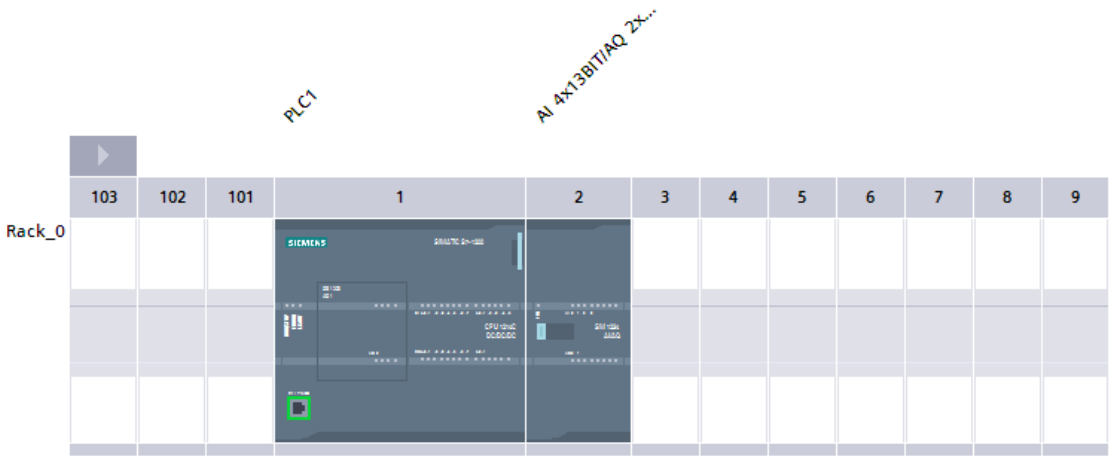
V HW konfiguraci je třeba počítat s tím, že samotný modul PLC nemá dostatečný počet vstupů a výstupů. Je tedy přidat vstupně-výstupní moduly. K tomuto jsem použil metody *HardwareObject.PlugNew()*. Více se můžete dočíst v této kapitole 3.5.3

Výpis 5.5: Konfigurace PLC

```
Device PLC_generated
 = cilovyProjekt.Devices.CreateWithItem(
   "OrderNumber:6ES7 214-1AG40-0XB0/V4.2",
   stroj.PLC, \textbf{null});
foreach (DeviceItem devitem in PLC_generated.DeviceItems)
{
 Console.WriteLine("\n\nDo {0}", devitem.Name);
 for (int i = 0; i < 40; ++i)
  {
   if (devitem.CanPlugNew("OrderNumber:6ES7 232-4HA30-0XB0/V1.0",
    "AQ 1x12BIT_1", i))
 {
 devitem.PlugNew("OrderNumber:6ES7 232-4HA30-0XB0/V1.0",
   "AQ 1x12BIT_1", i);
 Console.WriteLine(
 "\tOrderNumber: 6ES7 232 - 4HA30 - 0XB0 / V1.0 zapojeno do: "
 + i);
 break;
 }
   if (devitem.CanPlugNew("OrderNumber:6ES7 234-4HE32-0XB0/V2.0",
     "AI 4x13BIT/AQ 2x14BIT_1", i))
 {
 devitem.PlugNew("OrderNumber:6ES7 234-4HE32-0XB0/V2.0",
   "AI 4x13BIT/AQ 2x14BIT_1", i);
 Console.WriteLine(
 "\tOrderNumber:6ES7 234-4HE32-0XB0/V2.0 zapojeno do: "
 + i);
 break;
  }
 }
}
```

Přidání HMI

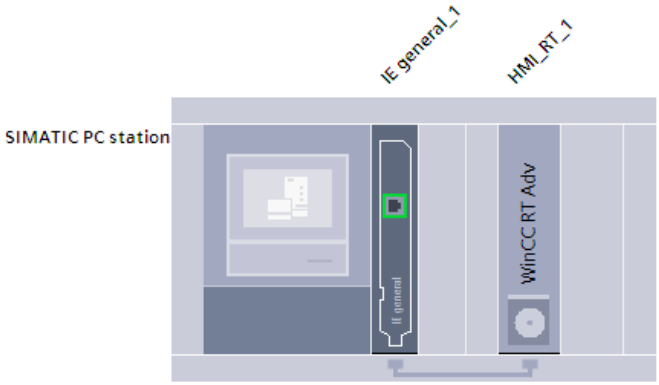
Pro účely přidání HMI jsem použil *MasterCopy*, která je dopředu umístěná v knihovně. Důvodem pro toto je, že jako HMI bylo v původním projektu pro model jeřábu navržen modul WinCC RT Advanced spouštěný pomocí runtime v rámci TIA Portal. Tento modul nelze přidat pomocí metody *PlugNew()*. Využil jsem k tomu tedy metody *DeviceComposition.CreateFrom(MasterCopy)*.



Obr. 5.4: Konfigurace PLC

Výpis 5.6: Konfigurace HMI

```
Device HMI_generated
 = cilovyProjekt.Devices.CreateFrom(
   cilovyProjekt.ProjectLibrary.MasterCopyFolder.MasterCopies.Find(
   "PC-System_1"));
```



Obr. 5.5: Konfigurace HMI

5.7.5 Metoda DevicesToSubnet()

Další krok, který logicky navazuje na hardwarovou konfiguraci je spojení zařízení do subsítě. Pro tento účel jsem napsal statickou metodu *DevicesToSubnet()*.

Parametry metody:

- Device device1 - tato instance obsahuje odkaz na zařízení PLC
- Device device2 - v programu je metoda volána tak, že tento parametr je odkaz na zařízení HMI
- Subnet subnet - odkaz na subsíť, na kterou se mají zařízení napojit
- string IPv4_dev1 - IP adresa portu 1. zařízení (tedy PLC)

Jak metoda pracuje je popsán v kapitole 3.6.2. Základem je získat uzel, který je obsažený v příslušné instanci *DeviceItem*. Tato instance musí poskytovat službu *NetworkInterface*. Instanci této služby získáme generickou metodou IEngineeringServiceProvider.GetService<T>(). Pokud *DeviceItem* neposkytuje tuto službu, metoda vrátí hodnotu **null**.

NetworkInterface pak obsahuje kompozici *NetworkInterface.Nodes*, která obsahuje všechny uzly. Ty jsem pak na *Subnet* připojil metodou *ConnectToSubnet()* viz. sekce 3.6.2

Výpis 5.7: Úryvek z metody DevicesToSubnet()

```
foreach (DeviceItem devitem in device1.DeviceItems)
{
foreach (DeviceItem item in devitem.DeviceItems)
 {
  NetworkInterface itf
 = ((IEngineeringServiceProvider)item).GetService<NetworkInterface
    >();
 if (itf != null)
 {
 foreach (Node node in itf.Nodes)
 {
              node.ConnectToSubnet(subnet);
              node.SetAttribute("Address", IPv4_dev1);
              //přidání IP adresy
 }
 }
}
}
```

5.7.6 Deklarace použitých listů

Tento kód jsem zde uvedl, aby bylo jasné jak vypadají instance pro uložení cest k souborům.

Výpis 5.8: Instance třídy List, použité pro předání cest ke XML souborům

```
List<string> projectGraphics = new List<string>();
List<List<string>> PLCblock_paths = new List<List<string>>();
List<List<string>> HMIblock_paths = new List<List<string>>();
List<Axis_params> axis_params = new List<Axis_params>();
```

5.7.7 Import software do zařízení

Pro import se používá metoda popsaná v této sekci 3.9.2. Jako parametr *FileInfo* se vytvoří nová instance konstruktorem *FileInfo(path)* kde path je string, který je uložen v příslušném listu, který je předán metodou *Begin_Export()* viz. sekce 5.7.3.

Import grafik

HMI panel využívá grafik, které byly přidány uživatelem, je tedy nutné přenést ze zdrojového projektu i je. V tomto případě voláme metodu *Import*, která je členem kompozice *cilovyProjekt.Graphics*. Cesty jsou uloženy v Listu *projectGraphics* viz. 5.8.

Rozdíl mezi přidáváním souborů do PLC a HMI je v umístění potřebných kompozic instancí.

Import do PLC

Do PLC je nutné přidávat importem bloky postupně, opačně než jak jsou volány v programu. Proto je vhodné zavést standartní pojmenování PLC bloků ve formátu (N)(TYPE)_block_name, kde N označuje hloubku zanoření bloku, TYPE označuje typ bloku (FB,FC,DB,OB).

Pokud je tato podmínka splněna, může započnout import do PLC. Jestli je formát dodržen není přímo námi kontrolováno, ovšem dojde k tomu, že metoda *Import* nadhodí výjimku, kterou zpracuje blok kódu **try-catch**.

Aby byly bloky v Listu správně seřazeny, jsou zavolány jeho členské metody *List<string>.Sort()* a *List<string>.Reverse()*. Viz. výpis 5.8, cesty k blokům jsou uloženy do instancí tříd typu *List<List<string»*, a jsou utříděny tak, aby každý *List<string>* obsahoval jen cesty k souborům určitého typu. Jsou tak zvlášť odděleny bloky, tabulky atd. O co se vlastně jedná je určeno tak, že pokud se pokoušíme

zavolat metodu *Import()* a typ souboru nesouhlasí s ostatními soubory v kolekci, které tato metoda patří, je nadhozena výjimka. Tato výjimka vede k postupnému zanořování do dalších bloků **try-catch**, dokud nedojde ke správné kolekci. Pokud soubor neodpovídá ani jedné kolekci, je na konci nadhozena výsledná výjimka.

Výpis 5.9: Algoritmus pro import do PLC

```
if (stroj.Mod != Stroj_funkcionalita_enum.no_mode.ToString())
{
 foreach (List<string> paths in PLCblock_paths)
 {
 paths.Sort();
 paths.Reverse();
 foreach (string file in paths)
 {
 try
 {
 PLC_generated_software.BlockGroup.Blocks.Import(new FileInfo(file)
     , ImportOptions.Override);
 }
 catch
 {
  try
  {
  PLC_generated_software.TagTableGroup.TagTables.Import(new
      FileInfo(file), ImportOptions.Override);
  }
  catch
  { throw; }
 }
 }
 }
}
```

Přidání TO_PositioningAxis jsem provedl pomocí metody *TechnologicalInstanceDBComposition.CreateWith(MasterCopy)*, protože metodou *Create()* novou TO_PositioningAxis nelze. Takto lze přidat technologický objekt již nakonfigurovaný a stačí mu přidat jen několik málo parametrů viz. výpis 5.10.

Import do HMI

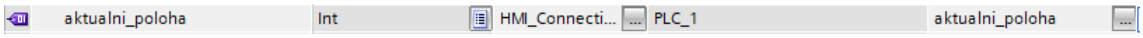
Import do HMI je podobný. Rozdíl je pouze v tom, že cesty k požadovaným XML souborům uloženy v *List<List<string> > HMIblock_paths* a v umístění cílových

Výpis 5.10: Nastavení parametrů pro TO_PositioningAxis

```
axis.Parameters.Find("_Actor.Interface.PTO").Value = 1;
axis.Parameters.Find("_Actor.Interface.PTO_OutputA").Value = "
    Navijak_step";
axis.Parameters.Find("_Actor.Interface.PTO_OutputB").Value = "
    Navijak_smer_nahoru";
axis.Parameters.Find("_Actor.Interface.EnableDriveOutput").Value =
    "Navijak_enable";
axis.Parameters.Find("_Actor.Interface.PTO_SignalType").Value = 2;
axis.Parameters.Find("_Actor.Interface.PTO_OutputBEnable").Value =
    true;
```

kompozic. Ty jsou umístěny ve složkách: *hmiTarget.ScreenFolder.Screens* pro obrazovky a *hmiTarget.TagFolder.TagTables* pro tabulky HMI tagů. Spojení jsou přístupná z kompozice *hmiTarget.Connections*.

Import tabulek tagů: Zde se na moment zastavím. HMI tagy jsou od PLC tagů poněkud odlišné. Jsou totiž přes spojení navázány na PLC tagy. Toto spojení musí ale existovat ještě před importem, jinak se celý TIA Portal zhroutí.



Obr. 5.6: Deklarace HMI tagu

Generátor uživatele na toto upozorní a nabídne řešení. Je třeba být přepnut do Devices & networks -> Network view. Zde kliknout na Connections. Viz. obr. C.2.

Po té tažením uživatel vytvoří spojení, jak je naznačeno na obrázku C.3

Potvrzením vyskakovacího okna viz. obr. C.4 uživatel dá povel k pokračování programu. Pokud klikne na zrušit, import do HMI se vůbec neprovede.

5.7.8 Konec metody

Na závěru metody je provedena kompilace PLC i HMI a její výsledky jsou vytištěny na konzolový výstup, program se pokusí stáhnout software do PLC a je odpojena zdrojová instance TIA Portalu.

6 Závěr

V této bakalářské práci jsem se zabýval vývojem desktopová aplikace, schopné automaticky generovat program pro modely jeřábu pomocí TIA Portal Openness. V rámci seznámení se s tímto nástrojem jsem vypracoval popis jeho funkcí, které by mohl uživatel využít.

Jako zdroj XML předpisu pro generování programu jsem využil hotový vzorový projekt v TIA Portalu a použitím vhodných metod (viz. sekce 5.7.3) jsem tento předpis vygeneroval a vhodně upravil. Rozdělení na jednotlivé úlohy jsem provedl tak, aby byly části pro automatický a čistě manuální běh odděleny a vytvořil jsem z nich další dva zdrojové projekty. Aplikace si pak vybere správný projekt podle toho, jaká je pro celé ústrojí modelu nadefinována funkcionalita.

Tyto soubory s předpisem jsou po té importovány do předpřipraveného projektu a celý program je sestaven. O průběhu je uživatel mé aplikace informován na konzolu.

Běh aplikace bohužel není plně automatický. Sám provede nejdůležitější úkony, jako hardwarová konfigurace, spojení zařízení do sítě, export a import bloků a technologických objektů, v oblasti spojení HMI a PLC a propojení HMI a PLC tagů potřebuje pomoc od uživatele. Sama ho navede na správný postup. Viz. sekce 5.7.7.

Nahrání software musí provést uživatel většinou také sám. Stahování programu bylo úspěšné pouze v případě stažení konfigurace do simulace PLC. Bohužel ke správnému nastavení stahování zevrubný zdroj informací chybí nebo se mi jej nepodařilo dohledat.

Všechny knihovny TIA Portal Openness se až na výjimky uvedené v předchozím odstavci a na případy, kdy došlo k neočekávaným chybovým stavům chovaly korektně a dle očekávání. Hierrchie tříd byla pro člověka pohybujícího se v prostředí TIA Portal snadno uchopitelná.

Literatura

[1] Siemens AG : *SIMATIC STEP 7 and WinCC Engineering V15* [online]. poslední aktualizace 12. 2017 [cit. 21. 5. 2018]. Dostupné z URL: <https://support.industry.siemens.com/cs/document/109755202/simatic-step-7-basic-professional-v15-and-simatic-wincc-v15?dti=0&lc=en-WW>

[2] Microsoft: *Introduction to the C# Language and the .NET Framework* [online]. poslední aktualizace 20. 7. 2015 [cit. 21. 5. 2018]. Dostupné z URL: <https://docs.microsoft.com//en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net\protect\discretionary{\char\hyphenchar\font}{}{}-framework>

[3] Microsoft: *Garbage Collection* [online]. poslední aktualizace 30. 3. 2017 [cit. 21. 5. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/>

[4] Microsoft: *Třída List* [online]. [cit. 21. 5. 2018]. Dostupné z URL: <https://msdn.microsoft.com/cs-cz/library/6sh2ey19(v=vs.110).aspx>

[5] Microsoft: *Třída Dictionary* [online]. [cit. 21. 5. 2018]. Dostupné z URL: <https://msdn.microsoft.com/cs-cz/library/xfhwa508(v=vs.110).aspx>

[6] Siemens AG : *SIMATIC Automating projects with scripts* [online]. poslední aktualizace 5. 2017 [cit. 21. 5. 2018]. Dostupné z URL: <https://cache.industry.siemens.com/dl/files/163/109477163/att_926042/v1/TIAPortalOpennessenUS_en-US.pdf>

[7] Microsoft: *Třída Dictionary* [online]. [cit. 21. 5. 2018]. Dostupné z URL: <https://msdn.microsoft.com/cs-cz/library/xfhwa508(v=vs.110).aspx>

[8] Microsoft: *ŠTOHL, PH.D., Ing. Radek, Ing. Miroslav JIRGL, PH.D., Ing. Jakub ARM a Ing. Štefan MIŠÍK. Laboratorní cvičení BPPA. FEKT VUT v Brně, 2016* [online]. Laboratorní úloha, Brno: 2016, FEKT VUT v Brně [cit. 21. 5. 2018]. Dostupné z URL: <https://www.vutbr.cz/www_base/priloha.php?dpid=143344>

Seznam symbolů, veličin a zkratek

WPF	Windows presentation foundation - knihovna tříd pro tvorbu grafického rozhraní
PLC	Programable logic controler - programovatelný logický automat/procesor
TIA Portal	Totally integrated automation - plně integrované prostředí pro automatizaci
API	Application Programming Interface - rozhraní pro programování aplikací
SCL	Structured Control Language - strukturovaný řídící jazyk
LAD	Ladder logic diagram - "žebříkový"diagram napodobující paralelní elektrické schema
FBD	Function Block Diagram - diagram funkčních bloků
STL	Statement List - seznam instrukcí
UDT	User Defined Types - Uživatelem definované datové typy
HMI	Human machine interface - rozhraní člověk-stroj

Seznam příloh

A Doplňující seznamy k aplikaci "Generátor openness"

A.1 Seznam použitých knihoven

Výpis A.1: Použité knihovny v aplikaci

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.IO;
using Siemens.Engineering;
using Siemens.Engineering.Compiler;
using Siemens.Engineering.HW;
using Siemens.Engineering.HW.Features;
using Siemens.Engineering.SW;
using Siemens.Engineering.SW.Blocks;
using Siemens.Engineering.Download;
using Siemens.Engineering.Download.Configurations;//toto v návodu
    chybí
using Siemens.Engineering.Connection;
using Siemens.Engineering.SW.Tags;
using Siemens.Engineering.SW.TechnologicalObjects;
using Siemens.Engineering.Hmi;
using Siemens.Engineering.Hmi.Tag;
using Siemens.Engineering.Hmi.Screen;
using Siemens.Engineering.Hmi.Communication;
using Siemens.Engineering.Hmi.Globalization;
```

A.2 Seznam použitých tříd

Tab. A.1: Přehled použitých tříd .NET

WPF	Linq a soubory	System
Assembly	DirectoryInfo	Console
Brushes	Environment	Dictionary
MessageBox	File	Ilist
MessageBoxButton	FileInfo	KeyValuePair
MessageBoxImage	OpenFileDialog	List
MessageBoxResult	Path	String
Visibility	SpecialFolder	
Window	XDocument	
	XElement	

Tab. A.2: Přehled použitých tříd TIA Portal Openness

Projekt a zařízení	PLC	HMI
Device	CodeBlock	Connection
DeviceComposition	DeviceItem	HmiTarget
ExportOptions	DeviceItemComposition	Screen
GlobalLibrary	PlcBlock	TagTable
GlobalLibraryInfo	PlcBlockGroup	
ImportOptions	PlcSoftware	
MultiLingualGraphic	PlcTag	
Project	PlcTagComposition	
ProjectComposition	PlcTagTable	
TiaPortal	Software	
TiaPortalMode	SoftwareContainer	
TiaPortalProcess	TechnologicalInstanceDB	

Tab. A.3: Přehled použitých tříd pro služby TIA Portal Openness

Služby	SLužby pro sítě
CompilerResult	NetworkInterface
CompilerResultMessage	NetworkPort
CompilerResultMessageComposition	Node
DownloadOptions	Subnet
DownloadProvider	
DownloadResult	
DownloadResultMessageComposition	
ICompilable	
IEngineeringServiceProvider	

Tab. A.4: Přehled použitých tříd

Event argumenty	Výjimky
ConfirmationEventArgs	ArgumentNullException
EventArgs	ArgumentOutOfRangeException
MouseButtonEventArgs	Exception
ResolveEventArgs	InvalidOperationException
RoutedEventArgs	IOException
TextChangedEventArgs	KeyNotFoundException
	NonRecoverableException
	NullReferenceException
	SecurityException
	UnauthorizedAccessException
	EngineeringSecurityException
	EngineeringTargetInvocationException

Tab. A.5: Přehled použitých mnou definovaných tříd

Stroj_funkcionalita_enum
Destinace
Axis_params
Stroje
Stroj
Funkcionality

B Doplňující zdrojové kódy

B.1 Metoda pro zpracování výsledků kompilace

Výpis B.1: Metoda pro zpracování výsledku kompilace

```
private void WriteCompilerResults(CompilerResult result)
{
 Console.WriteLine("State:" + result.State);
 Console.WriteLine("Warning Count:" + result.WarningCount);
 Console.WriteLine("Error Count:" + result.ErrorCount);
 RecursivelyWriteMessages(result.Messages);
}
private void RecursivelyWriteMessages
 (CompilerResultMessageComposition messages, string
 indent = "")
{
 indent += "\t";
 foreach (CompilerResultMessage message in messages)
 {
  Console.WriteLine(indent + "Path: " + message.Path);
  Console.WriteLine
   (indent + "DateTime: " + message.DateTime);
  Console.WriteLine
   (indent + "State: " + message.State);
  Console.WriteLine
   (indent + "Description: " + message.Description);
  Console.WriteLine
   (indent + "Warning Count: " + message.WarningCount);
  Console.WriteLine
   (indent + "Error Count: " + message.ErrorCount);
  RecursivelyWriteMessages
   (message.Messages, indent);
 }
}
```

B.2 Třída Stroje

Výpis B.2: Třída Stroje

```
public class Stroje
{
//konstanty
private const int IP_length = 4;
//parametry
protected List<Stroj> list_Stroju;
//vlastnosti
public List<Stroj> List_Stroju { get => list_Stroju; set =>
    list_Stroju = value; }
//obsluzne metody

//c-tory
public Stroje() => List_Stroju = new List<Stroj>();//implicitni c-
    tor vytvori prazdny list
public Stroje(List<Stroj> list_Stroju)// copy c-tor
{
List_Stroju = list_Stroju ?? throw new ArgumentNullException(nameof
    (list_Stroju));
}
public Stroje(Stroj astroj)
{
List_Stroju.Add(astroj);
}
public class Stroj...
public class Funkcionality...
```

B.3 Třída Stroj

Výpis B.3: Třída Stroj

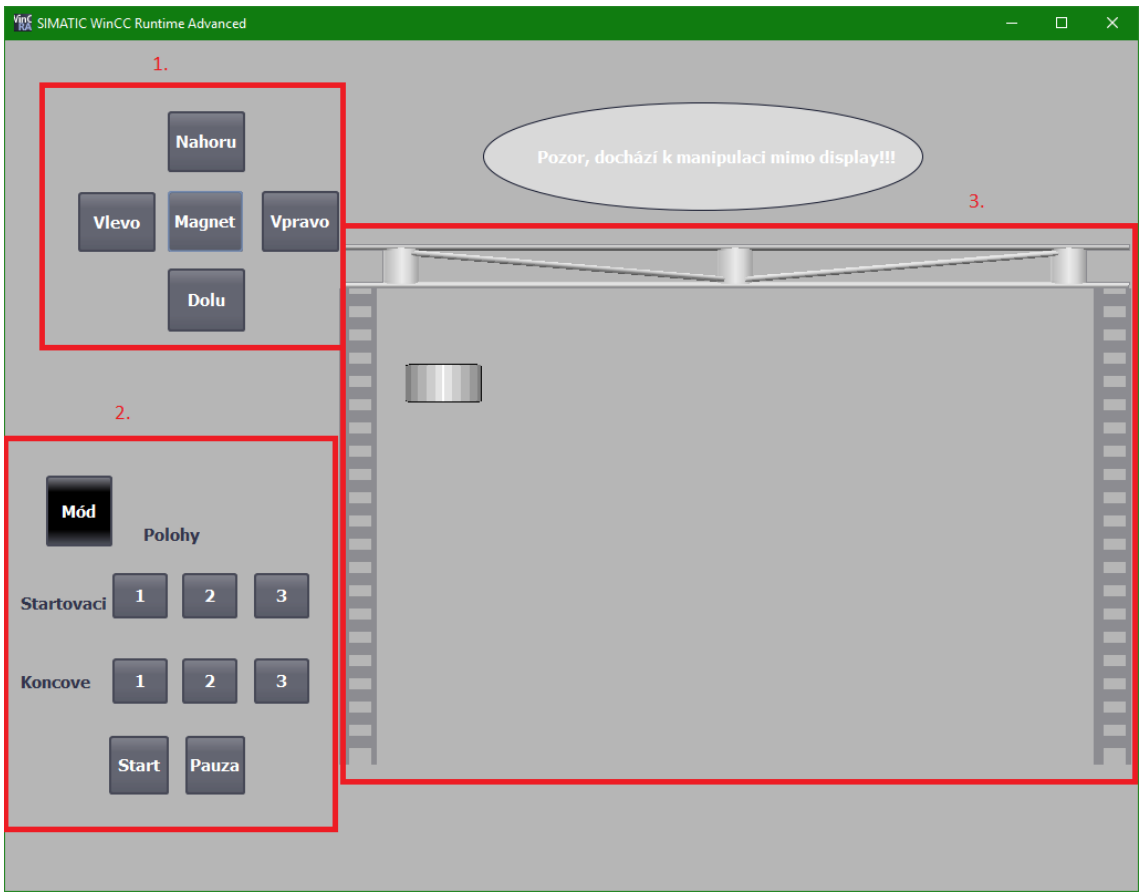
```
public class Stroj
{
 private int[] ipaddress;

            //vlastnosti
 public string JmenoHMI { get; set; }
 public int[] IPaddress
 {
 get => ipaddress;
 set
 {
 ipaddress = value;
 Ipaddress_string = string.Format("{0}.{1}.{2}.{3}", value[0],
    value[1], value[2], value[3]);
 }
}
 public string JmenoPLC { get; set; }
 public string Mod { get; set; }
 public string Ipaddress_string { get; set; }
...
public Stroj(string aJmenoHMI, string aIPaddress, string aJmenoPLC,
    string aMod)
{
 JmenoHMI = aJmenoHMI ?? throw new ArgumentNullException(nameof(
    aJmenoHMI));
 Ipaddress_string = aIPaddress ?? throw new ArgumentNullException(
    nameof(aIPaddress));
 JmenoPLC = aJmenoPLC ?? throw new ArgumentNullException(nameof(
    aJmenoPLC));
 Mod = aMod ?? throw new ArgumentNullException(nameof(aMod));
 }
}
```

B.4 Třída Funkcionality

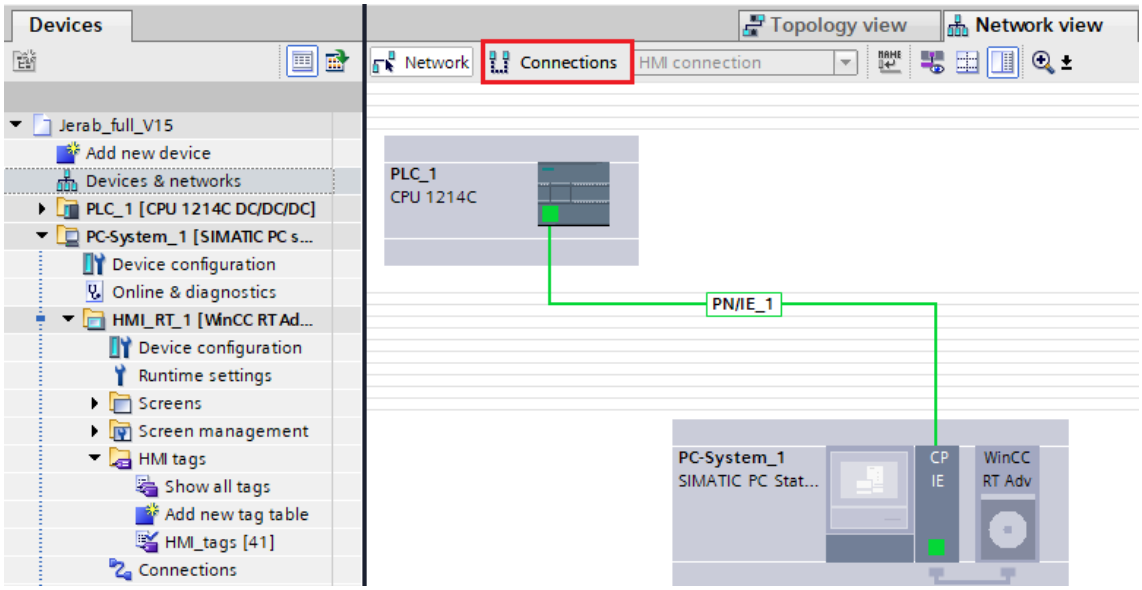
Výpis B.4: Třída Funkcionality

```
public class Funkcionality
{
 protected Dictionary<string, string> funkcionalita = new
    Dictionary<string, string>();

 public Dictionary<string, string> Funkcionalita { get =>
    funkcionalita; set => funkcionalita = value; }//čtení slovníku
    realizováno vlastností

 public Funkcionality(string atkey, string atval)
 {
  Funkcionalita.Add(atkey, atval);
  Funkcionalita.Add(Stroj_funkcionalita_enum.no_mode.ToString(),
     null);
  Funkcionalita.Add(Stroj_funkcionalita_enum.manual.ToString(),
     string.Format(@"\Automation\{0}\{0}.ap15", "Jerab_manual_V15")
     );
  Funkcionalita.Add(Stroj_funkcionalita_enum.auto.ToString(),
     string.Format(@"\Automation\{0}\{0}.ap15", "Jerab_auto_V15"));
  Funkcionalita.Add(Stroj_funkcionalita_enum.full.ToString(),
     string.Format(@"\Automation\{0}\{0}.ap15", "Jerab_full_V15"));
 }
}
```

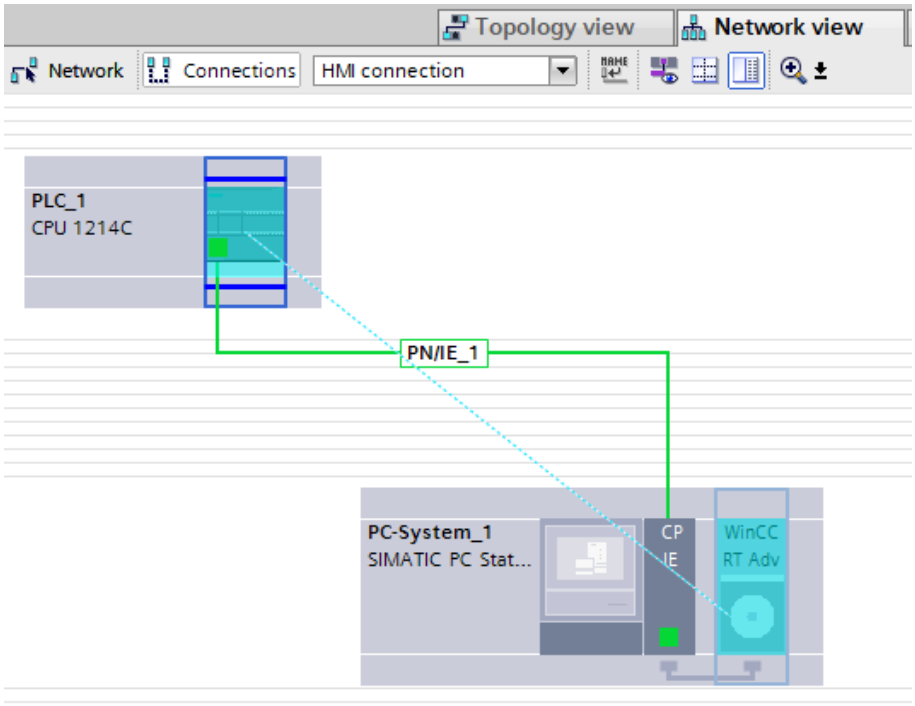
C Doplňující obrázky



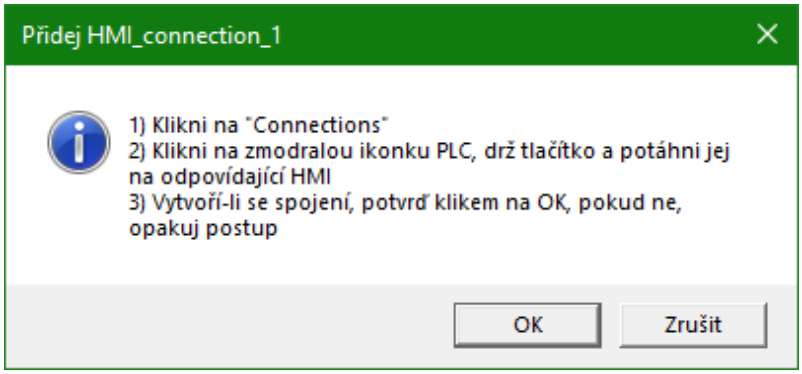
Obr. C.1: Obrazovka ovládacího rozhraní



Obr. C.2: Pohled na Devices and networks



Obr. C.3: Vytvoření spojení PLC a HMI



Obr. C.4: Vyskakovací okno pro potvrzení vytvoření spojení

D Obsah přiloženého ZIP souboru

- `/` ..... kořenový adresář přiloženého CD/ZIP souboru
  - `Generator_openness` ..... Složka s projektem ve Visual Studio
    - `Generator_openness` ..... Složka se zdrojovými soubory hlavní aplikace
      - `bin` ..... Složka s binárními soubory hlavní aplikace
        - `Debug` . Složka pro soubory pro vývoj aplikace, obsahuje mimo jiné i exe soubor
      - `obj`
        - `Debug` . Složka pro soubory pro vývoj aplikace, obsahuje mimo jiné i exe soubor
      - `Properties` ..... Složka se sestavovacími soubory projektu
      - `App.config`
      - `App.xaml`
      - `App.xaml.cs`
      - `Generator_openness.csproj`
      - `MainWindow.xaml`
      - `MainWindow.xaml.cs`
      - `packages.config`
      - `Stroj_funkcionalita_enum.cs`
      - `Stroje.cs`
      - `Window1.xaml`
      - `Window1.xaml.cs`
    - `Libs` ..... Složka s knihovnami projektu
    - `Packages` ..... (prázdná) složka s přídavnými balíčky pro projekt
    - `Src` ..... Složka se zdrojovými soubory přídavných balíčků
    - `Generator_openness.sln`
  - `Jerab_auto_V15.zap15` Archiv s projektem v TIA Portal pro ovl. modelu jeřábu v automatické verzi
  - `Jerab_full_V15.zap15` Archiv s projektem v TIA Portal pro ovl. modelu jeřábu v plné verzi
  - `Jerab_manual_V15.zap15` ..... Archiv s projektem v TIA Portal pro ovl. modelu jeřábu v manuální verzi
  - `Jerab_V15.zap15`. Archiv s projektem v TIA Portal pro generování ovl. programu modelu jeřábu
  - `xslani02_bakalarska_prace.pdf` ..... Elektronická verze dokumentační části bakalářské práce
  - `readme.txt` ..... soubor s popisem obsahu CD