České vysoké učení technické v Praze

Fakulta elektrotechnická

Katedra řídicí techniky

Diplomová práce

Automatické generování standardizovaných objektů přes rozhraní TIA Openness

Automatic generation of standardized objects via the TIA Openness interface

Autor: Tomáš Vayhel

Vedoucí práce: Ing. Martin Cicvárek

Vedoucí práce – konzultant: Ing. Martin Hlinovský, Ph.D.

Studijní program: Kybernetika a robotika

Praha 2024

24. KVĚTNA 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vayhel** Jméno: **Tomáš** Osobní číslo: **491860**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Automatické generování standardizovaných objektů přes rozhraní TIA Openness

Název diplomové práce anglicky:

Automatic generation of standardized objects via the TIA Openness interface

Pokyny pro vypracování:

1) Prostudujte rozhraní TIA Openness
2) Navrhněte postup generování objektů do TIA Portalu pomocí TIA Openness
3) Vytvořte aplikaci pro generování standardizovaných objektů
4) Ověřte funkčnost výsledného řešení

Seznam doporučené literatury:

[1[SIEMENS. TIA Portal Openness: API for automation of engineering workflows [online]. Siemens, 2021 [cit. 2023-08-09]. Dostupné z: https://cache.industry.siemens.com/dl/files/533/109798533/att_1069908/v1/TIAPortalOpennessenUS_en-US.pdf
[2] SIEMENS. TIA Portal Openness: Introduction and demo application [online]. Siemens, 2023 [cit. 2023-08-09]. Dostupné z: https://cache.industry.siemens.com/dl/files/692/108716692/att_1131753/v2/108716692_TIA_PortalOpenness_GettingStartedAndDemo_V17_en.pd
[3] SIEMENS. Selection Guide TIA Add-In / Openness Application / Modular Application Creator Equipment Module. Online. 2023. Dostupné z: https://cache.industry.siemens.com/dl/files/879/109816879/att_1139596/v1/109816879_SelectionGuide_Add-In_Openness-App_MAC.pdf.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Martin Cicvárek SIDAT, spol. s r. o., Jinonická 80, 158 00 Praha

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Martin Hlinovský, Ph.D. katedra řídicí techniky FEL

Datum zadání diplomové práce: **15.01.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce:
do konce letního semestru 2024/2025

Ing. Martin Cicvárek	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Podpis studenta

Datum převzetí zadání

Čestné prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze dne ....

Podpis autora práce

Poděkování

Tímto bych chtěl poděkovat Ing. Martinu Cicvárkovi, za jeho ochotu při vedení této práce a jeho cenné odpovědi na dotazy ohledně firemního programového standardu. Dále bych chtěl poděkovat školnímu garantovi práce, Ing. Martinu Hlinovskému, za jeho ochotu odpovídat na dotazy ohledně organizačních záležitostí spojených se zadáváním tématu této práce. Na závěr bych chtěl poděkovat firmě Sidat za to, že mi poskytla možnost realizovat tento projekt.

Abstrakt

Tato diplomová práce se zaměřuje na vývoj automatického generátoru instancí standardizovaných objektů ve vývojovém prostředí TIA Portal, využívajícího rozhraní TIA Portal Openness. Projekt byl realizován na míru dle specifických potřeb firmy Sidat a bude se dále využívat v jejích interních procesech.

V první části práce je představen teoretický základ nezbytný pro pochopení použitých metod a nástrojů, přičemž hlavní důraz je kladen na terminologii spojenou s vývojovým prostředím TIA Portal a funkcionalitu rozhraní TIA Portal Openness.

Další část práce se věnuje návrhu a implementaci celkové aplikační struktury, která je rozdělena do několika procesů za účelem dosažení automatického generování požadovaných instancí objektů. Tento proces zahrnuje přípravu Excel souboru pro zadání, kontrolu a zpracování uživatelských vstupů, které specifikují požadované instance objektů. Zpracovaný uživatelský vstup je poté přenesen ve formě příkazového souboru na cílové zařízení s prostředím TIA Portal, kde je spuštěna vytvořená aplikace generátoru. Tato aplikace zpracuje příkazový soubor a na základě XML předloh ze vzorového projektu vytvoří nové XML soubory, které jsou následně importovány přes rozhraní TIA Portal Openness do požadovaného projektu v TIA Portalu.

Práce rovněž popisuje další využití uživatelského vstupu pro tvorbu .xlsx souborů, které lze manuálně importovat na operátorské panely řady Comfort nebo Comfort Unified ve vývojovém prostředí TIA Portal za účelem vytvoření HMI tagů, alarmových hlášení a textových listů navázaných na vygenerované instance objektů.

V závěru je prezentována ukázka celého procesu generování z pohledu uživatele a rozbor množství ušetřeného času při přípravě nových projektů v prostředí TIA Portal, včetně naměřených časových úspor pro jednotlivé fáze generování v různých testovacích scénářích.

Klíčová slova: Automatické generování objektů, TIA Portal, TIA Portal Openness, Programovatelné logické automaty (PLC), Průmyslová automatizace

Abstract

This thesis focuses on the development of an automatic generator for instances of standardized objects in the TIA Portal development environment, utilizing the TIA Portal Openness interface. The project was tailored to the specific needs of the company Sidat and will continue to be used in their internal processes.

The first part of the thesis introduces the theoretical foundation necessary for understanding the methods and tools used, with an emphasis on the terminology associated with the TIA Portal development environment and the functionality of the TIA Portal Openness interface.

The next part of the thesis is dedicated to the design and implementation of the overall application structure, which is divided into several processes to achieve the automatic generation of the desired object instances. This process includes the preparation of an Excel file for entering, checking, and processing user inputs that specify the required object instances. The processed user input is then transferred in the form of a command file to the target device

with the TIA Portal environment, where the developed generator application is launched. This application processes the command file and, based on XML templates from a sample project, creates new XML files, which are then imported via the TIA Portal Openness interface into the desired project in TIA Portal.

The thesis also describes the further use of user input for the creation of .xlsx files, which can be manually imported into Comfort or Comfort Unified series operator panels in the TIA Portal development environment to create HMI Tags, alarm messages, and text lists linked to the generated object instances.

In conclusion, the thesis presents a demonstration of the entire generation process from the user's perspective and an analysis of the time saved when preparing new projects in the TIA Portal environment, including measured time savings for individual generation phases in various testing scenarios.

Keywords: Automatic object generation, TIA Portal, TIA Portal Openness, Programmable Logic Controllers (PLC), Industrial Automation

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

S růstem každé společnosti nebo skupiny přichází dříve či později nevyhnutelná potřeba standardizovat zavedené pracovní postupy. Hlavním důvodem bývá snaha minimalizovat opakující se činnosti tak, aby se mohli zaměstnanci soustředit převážně na aktuálně řešenou problematiku. Další výhodou standardizace je skutečnost, že zaměstnanec potřebuje znát pouze jeden hlavní koncept, což mu usnadňuje pochopení a zapojení se do projektů, na kterých nepracoval od začátku. To pak zaměstnanci umožňuje se rychleji začlenit do samotného vývojového procesu.

V této práci budeme mluvit o standardizaci technologických objektů v průmyslovém prostředí, konkrétně jejich reprezentaci pomocí bloků kódu ve vývojovém prostředí TIA Portal, které zajišťují ovládání a zpracování dat například pohonů, senzorů měření, ventilů a dalších technologických objektů.

Ve firmě Sidat, pro jejíž interní využití byla tato práce vytvořena, vznikla snaha zjednodušit i proces nasazování standardizovaných struktur projektů. Tento proces je totiž stále poměrně zdlouhavý kvůli množství potřebných instancí standardizovaných objektů a počtu jejich parametrů, které bývá potřeba specifikovat. Pro středně velké nebo větší projekty může tvorba těchto instancí jednotlivci zabrat i týdny práce.

Proto vznikla iniciativa tento proces zautomatizovat. Vývojové prostředí TIA Portal od společnosti Siemens totiž poskytuje programové rozhraní TIA Portal Openness, které umožňuje externí programové připojení k TIA Portalu a manipulaci s obsahem otevřených projektů.

Hlavním cílem této práce je vytvořit aplikaci, která bude pomocí rozhraní TIA Portal Openness generovat instance firemně standardizovaných objektů, a tím ulehčit práci programátorům při opakujících se úkonech. Úspěšnou realizací pak dosáhneme snížení počtu hodin, které musí programátor strávit při přípravě standardizované struktury u nových projektů. Ušetřený čas pak lze využít pro další programově tvůrčí činnost a obratem tak zvýšit i konkurenceschopnost společnosti.

V následujících kapitolách bude popsána základní terminologie, struktura projektu s principem jeho implementace a ověření funkcionality spolu s odhadem množství ušetřeného času použitím projektu.

Pro lepší představu o celkové struktuře práce, která bude detailněji popsána v následujících kapitolách, lze základní koncept projektu shrnout následovně:

- **Zadání uživatelského vstupu** - Uživatel zadá vstupní data o podobě požadovaných instancí standardizovaných objektů prostřednictvím aplikace Excel.
- **Zpracování dat** - Data z tabulek Excelu budou pomocí makra v jazyce VBA převedena na příkazový soubor, který lze přenášet mezi zařízeními.
- **Tvorba XML souborů** - Příkazový soubor bude zpracován v aplikaci napsané v jazyce C# a na jeho základě budou vytvořeny XML soubory dle připravených knihovních předloh.
- **Import do TIA Portalu** - Vzniklé XML soubory se pomocí rozhraní TIA Openness importují do projektu v prostředí TIA Portal.

2. Teoretická část

Tato kapitola dokumentu uvádí základní terminologii potřebnou pro lepší orientaci čtenáře napříč prací. Není tedy nutné číst ji celou a stačí se zaměřit pouze na podkapitoly, které čtenář potřebuje pro doplnění svých znalostí.

V této kapitole se seznámíme se základní terminologií a koncepty spojenými s vývojovým prostředím TIA Portal, strukturou firemního standardu, rozhraním TIA Portal Openness a využitím XML souborů. Tato část slouží jako teoretický základ pro pochopení praktické implementace popsané v dalších kapitolách.

2.1. TIA Portal

V této kapitole popíšeme nezbytnou terminologii ohledně prostředí TIA Portal, kterou využíváme napříč naší prací. Cílem této části textu je poskytnout čtenáři nezbytný kontext bez přílišných detailů. Tento projekt se nezabývá tvorbou samotného programu, ale modifikací již existujícího kódu k tvorbě nových instancí standardizovaných objektů.

TIA Portal (Tottaly Integrated Automation portal) je plně integrované vývojové automatizační prostředí od firmy SIEMENS [2]. Příkladem integrovaného softwaru jsou:

- SIMATIC Step 7 je inženýrský software sloužící ke konfiguraci a programování řídících jednotek řady SIMATIC s7-1200, s7-1500 a s7-300/400. [2]
- SIMATIC WinCC slouží ke konfiguraci a programování vizualizačních aplikacích na panelech SIMATIC, průmyslových počítačích SIMATIC a běžných počítačích s profesionálním vizualizačním softwarem WinCC Runtime Advanced. [2]

2.1.1. PLC Tagy

Tag v terminologii programovatelných logických automatů (PLC) označuje unikátní identifikátor proměnné v databázi, který se překládá na konkrétní vnitřní adresu dle typu zařízení [2].

U projektu ve vývojovém prostředí TIA Portálu jsou tagy seskupeny v tzv. tagových tabulkách. Jde o list obsahující definice jednotlivých PLC tagů, kde je pro každý tag nezbytné, aby měl symbolický název spojený s konkrétní adresou a datovým typem. [1][2]

V TIA Portálu jsou dle typu tagů odlišovány adresní prostory, relevantní pro nás jsou konkrétně:

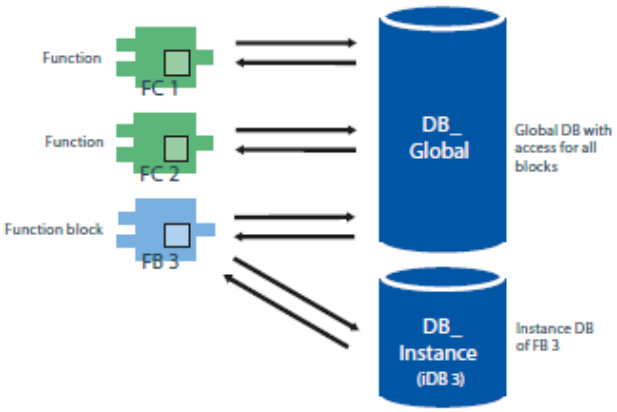
- I – paměť sloužící k uchování obrazu vstupních proměnných
- Q – paměť sloužící k přiřazení hodnot výstupních proměnných
- M – adresní prostor pro vnitřní paměť programu [1]

2.1.2. Datový blok

Datové bloky (DB) jsou využívány k ukládání dat ve speciální pamětní oblasti, ke kterým lze přistoupit z ostatních programových bloků (např. funkčních bloků, které definují něco jako třídu objektu, ze kterých lze vytvářet instance a které budou vysvětleny v další sekci). [2]

Existují dva typy datových bloků, a to globální a instanční. Hlavním rozdílem je to, že instanční datový blok je vždy vztažen k nějakému funkčnímu bloku. Díky vazbě na funkční blok je struktura instančního DB pevně dána a nelze individuálně změnit, kdežto u globálního datového bloku si proměnné a strukturu programátor vytváří sám pro každý datový blok. [1] [2]

Dělbou na globální a instanční bloky se v podstatě jedná o rozsah možného využití datových bloků v rámci programu. Instanční datové bloky pak vystupují pouze jako lokální k ukládání a poskytování dat dané instance provázaného funkčního bloku. Lze k nim sice stále přistupovat a zapisovat do nich i ze zbytku programu, ale drží pouze data týkající se příslušné instance, která je neustále modifikuje. Jednoduché schéma znázorňující rozsah využití typů datových bloků je na obrázku číslo 1.



Obrázek 1 Rozsah využití datových bloků v PLC [2]

2.1.3. Funkce

Funkce jsou programové bloky kódu bez vnitřní statické paměti. Všechny potřebné parametry tedy musí být definovány a předány funkci při jejím volání. Obsahují program, který se vykoná při zavolání funkce externím programovým blokem. Mohou se tedy používat pro opakované vykonávání stejné operace.[2]

2.1.4. Funkční blok

Funkční bloky (FB) jsou programové bloky s pamětí, u kterých jsou jejich data napříč cykly ukládána v instancích datových bloků. Funkční bloky na rozdíl od funkcí umožňují použití statických proměnných navrch k dočasným proměnným. Rozdíl mezi nimi je takový, že dočasné proměnné, jak název napovídá, mají platnost pouze jeden cyklus a je nutné zajistit před každým jejich čtením i jejich inicializaci. Jinak totiž nejde zajistit předvídatelnou funkci programu. Statické proměnné však na rozdíl od těch dočasných svou hodnotu napříč cykly zachovávají a jsou tedy i hlavním důvodem, proč jsou následně data funkčního bloku uchovávána v instančních datových blocích. [2][1]

Funkční bloky obsahují bloky kódu, které se vykonají při každém cyklu, a lze z nich vytvářet individuální instance funkčního bloku. Každá tato instance potřebuje vždy definovat i příslušný instanční datový blok, do kterého pak budou ukládány data náležící příslušné instanci funkčního

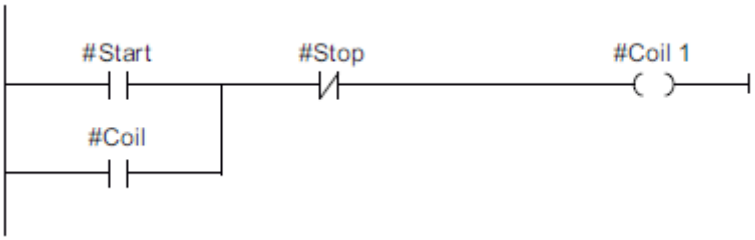
bloku. V jednom cyklu programu lze tedy ovládat nezávisle několik instancí funkčních bloků, kde jako příklad lze uvést obsluha pohonů.[2][1]

2.1.5. Programové jazyky funkcí

TIA Portal nabízí několik typů programovacích jazyků v závislosti na typu programovaného PLC. Pro nejčastěji používanou řadu PLC s7-1500 TIA Portal podporuje následující programovací jazyky: „LAD, FBD, STL, SCL, GRAPH, CEM“ [2]. Pro účely naší práce je důležité, aby čtenář měl alespoň základní představu o rozdílu mezi jazyky LAD a STL. Proto je stručně popíšeme v následujících sekcích, zatímco ostatní jazyky nebudeme rozebírat.

2.1.5.1. Programovací jazyk LAD

LAD (Ladder Logic) je grafický programovací jazyk, jehož reprezentace je založena na obvodových schématech. Program je zobrazován v jednom nebo více networcích, přičemž každý network obsahuje v levé části power-rail, odkud vycházejí jednotlivé větve programu. Binární signály jsou uspořádány ve formě kontaktů/spojů na programových větvích. Sériové zapojení prvků na větvi tvoří logickou funkci AND, zatímco paralelní zapojení vytvářejí logickou funkci OR. Složité funkce jsou znázorněny pomocí bloků kódu se vstupy a výstupy. [2] Příklad jednoduchého zapojení pomocí jazyka LAD je znázorněn na obrázku číslo 2.



Obrázek 2 Příklad jednoduchého kódu v jazyce LAD [2]

2.1.5.2. Programovací jazyk STL (Statement list)

STL (Statement List) je textově orientovaný programovací jazyk používaný k programování logických bloků. Program STL je rozdělen do networků, přičemž každý network může obsahovat jeden nebo více řádků, které se číslují od jedničky. Jednotlivé instrukce STL jsou programovány do řádků networku, kde v každém řádku může být uvedena pouze jedna instrukce STL. Každý příkaz je instrukcí pro CPU, které je vykonává shora dolů. [2] Příklad kódu v jazyce STL je uveden na obrázku číslo 3.

STL	Explanation
A"Tag_Input_1"	// Check whether the signal state of the operand is "1" and AND with current RLO
A "Tag_Input_2"	// Check whether the signal state of the operand is "1" and AND with current RLO
S "Tag_Output"	// Sets the operand to "1" if the RLO is "1"

Obrázek 3 Příklad jednoduchého kódu v jazyce STL [2]

2.1.6. Uživatelem definovaný typ (UDT)

Uživatelem definovaný datový typ (UDT) je komplexní datový typ, který uživatelé definují pro deklaraci tagů. Tento typ představuje datovou strukturu složenou z různých komponent, které mohou být odvozeny z jiných datových typů PLC, pole (ARRAY) nebo struktur (STRUCT). UDT lze deklarovat centrálně a pak ho přiřadit tagům, nebo obsahu datových bloků jako datový typ. Změny v této centrální deklaraci se pak automaticky projeví na všech místech, kde je tento typ použit.[2]

2.1.7. Vizualizační prostředí

Ačkoliv je práce primárně soustředěna na automatické generování instancí standardizovaných objektů, budeme se v její části věnovat i generování části standardu pro operátorské panely. Tato kapitola tedy slouží pouze ke stručnému vyjasnění používaných pojmů v této oblasti.

- **HMI** – Zkratka pro Human Machine Interface, což je rozhraní mezi člověkem a strojem. Může se jednat například o operátorský panel, který slouží k obsluze zařízení řízeného programem z PLC
- **Operátorský panel** (OP) – Vizualizační zařízení umožňující obsluze interagovat s programem ovládajícím technologické zařízení. Na operátorském panelu lze vizualizovat stavy vnitřních proměnných a umožnit zadávání dat nebo povelů do programu za provozu.
- **Operátorský panel Comfort** – Název starší řady operátorských panelů od společnosti Siemens
- **Operátorský panel Comfort Unified** – Nejnovější řada operátorských panelů od společnosti Siemens.
- **HMI tagy** – Jedná se o proměnné ve vizualizačním prostředí a mohou být externí nebo interní. Externí tagy spojují vizualizační prostředí s daty v automatizačním systému. Hodnoty externích tagů odpovídají procesním hodnotám v paměti automatizačního systému. Tyto hodnoty lze číst nebo přepisovat přímo v paměti systému. [2]
- **Textové listy** – Každý textový list má jedinečný název a obsahuje položky textu uložené pod unikátními hodnotami. Každá položka může mít různé obsahové varianty pro různé jazyky projektu. Pro úpravy, překlady nebo nahrání textů lze textové listy exportovat do formátu Office Open XML a následně je opět importovat zpět. [2]

2.2. Struktura firemního standardu

Tento oddíl poskytuje přehled o struktuře a implementaci firemního programátorského standardu, který se používá jak pro řízení procesů, tak pro factory automation. Standard rozděluje řízení procesů na jednotlivé technologické objekty, jejichž programová reprezentace je zajištěna již hotovými funkčními bloky.

Úvodem, než vysvětlíme samotnou strukturu firemního standardu, je třeba si vyjasnit význam jednotlivých označení použitých v souvislosti se standardem napříč touto prací.

- **Struktura firemního standardu** – Postup nasazení firemního standardu a jeho reprezentace v projektu. Zahrnuje prvky i nad rámec standardizovaných objektů a jejich instancí.
- **Standardizovaný objekt** – Zde se bavíme o konkrétní implementaci FB, která je interním know-how firmy.
- **Instance standardizovaného objektu** (zkráceně instance objektu) – Zde je řeč o konkrétní aplikaci standardizovaného objektu. Dle typu zavoláme před-vytvořené FB ve funkci volání, inicializujeme jeho vnitřní parametry ve funkci Init a definujeme instanční datový blok, do kterého se ukládají vnitřní proměnné pro danou instanci.

2.2.1. Typy standardizovaných bloků

Firemní standard je založen na principu rozložení automatizační úlohy (technologie, která se má řídit) na základní technologické objekty. Tyto objekty už mají své naprogramované funkční bloky, ve kterých je připravena jejich vnitřní logika. Programátor tedy nemusí v každé úloze znovu vytvářet základní ovládání typických technologických objektů a řešení automatizované úlohy je tak zjednodušeno na definování vlastností těchto objektů a jejich správné propojení mezi s sebou. Další výhodou tohoto postupu je, že funkční bloky jsou již otestovány na předchozích projektech, tudíž je minimalizována programová chyba uvnitř těchto bloků, a zároveň je dosaženo lepší čitelnosti a strukturovanosti kódu mezi programátory napříč firmou.

Níže následuje výčet typů standardizovaných bloků, které se budeme v této práci snažit automaticky generovat. Výčet je uveden i se stručným vysvětlením významu a použití daného objektu.

- **Motor** - Zajišťuje programové ovládání a vyhodnocování chyb technologických objektů, které zapínají a vypínají svůj chod na základě vstupních signálů (například motor, topení, filtr, apod.). Blok na základě vstupních signálů vyhodnocuje své stavy, umožňuje manuální ovládání a vytváří alarmová hlášení.
- **Ventil** - Zajišťuje logiku technologických objektů, které ovládají otevírání a zavírání cesty médiu pomocí elektrických povelů. Blok na základě vstupních signálů vyhodnocuje své stavy, umožňuje manuální ovládání a vytváří alarmová hlášení.
- **Analogové měření** - Zpracovává číselnou hodnotu na svém vstupu, kontroluje její stanovený rozsah a vyhodnocuje překročení definovaných mezí. Umí také přímo zpracovat analogový signál ze vstupní karty získaný z měření fyzikálních veličin.
- **Digitální měření** - Zpracovává binární signál na svém vstupu, umožňuje jeho inverzi či zpoždění a vytváří alarmové hlášení.

- **Klapka** - Zajišťuje logiku objektů, které se mohou přesouvat mezi svými dvěma krajními pozicemi a při tomto přesunu se mohou i zastavit. Nejčastěji se jedná o elektromotorické klapky. Blok na základě vstupních signálů vyhodnocuje své stavy, umožňuje manuální ovládání a vytváří alarmová hlášení.
- **Píst** – Zajišťuje logiku technologických objektů, které se na základě elektrických povelů přesouvají jen mezi dvěma pozicemi, nejčastěji se jedná o pneumatické písty. Blok na základě vstupních signálů vyhodnocuje své stavy, umožňuje manuální ovládání a vytváří alarmová hlášení.
- **Skupina** – Standardizovaný objekt sloužící ke sdružení několika instancí standardizovaných objektů do jedné logické skupiny, která je umožní ovládat jako celek. Usnadňuje tvorbu technologických návazností při spouštění či vypínání řízené technologie.
- **Cesta** – Standardizovaný objekt sloužící ke sdružení několika instancí standardizovaných objektů do společného celku, který ovlivňuje ovládání na základě aktivování jeho volby. Jako příklad si lze představit výběr cesty procesního toku materiálu při výrobě.
- **Navolení** – Standardizovaný objekt sloužící k poskytnutí přepínače volby na vizualizaci pro operátory technologického procesu za provozu.

2.2.2. Instanční datové bloky

Při vytvoření instance standardizovaného objektu je důležité nadefinování instančního datového bloku, který slouží k ukládání vnitřních proměnných a stavů dané instance. Jeho obsah je tvořený vnitřními proměnnými funkčního bloku, ze kterého tato instance vzniká.

Při tvorbě instančních datových bloků je potřeba stanovit název a číslo datového bloku tak, aby byly oba parametry unikátní. Tento název a číslo je pak vztažen na celou instanci standardizovaného objektu jak napříč kódem, tak i na vizualizačním rozhraní HMI pro informování obsluhy.

2.2.3. Funkce volání standardizovaných objektů (Call)

V této sekci nám bude trochu splývat terminologie ohledně významu pojmu volání. Z hlediska TIA Portalu je pod pojmem volání míněno zpracování kódu konkrétní instance funkčního bloku [2]. Z pohledu firemního standardu je pod pojmem volání standardizovaných objektů myšlena celá funkce, ve které jsou volány jednotlivé instance funkčních bloků.

Ve funkci je volání jedné instance standardizovaného objektu rozčleněno do několika předpřipravených networků psaných v jazyce LAD. Nejdříve jsou předpřipraveny networky pro budoucí navázání řídicí logiky a blokačních podmínek následované networkem s vlastním voláním instance funkčního bloku standardizovaného objektu.

Volané instanci funkčního bloku je vždy přiřazen konkrétní instanční datový blok, který slouží jako její vnitřní paměť. Dále jsou zde navázány veškeré vstupní a výstupní signály, které daná instance standardizovaného objektu používá. Tyto signály se vyhodnocují dle vnitřní logiky daného standardizovaného objektu, která kombinací dalších vstupů a vnitřních proměnných přiřadí hodnoty na svůj výstup.

2.2.4. Inicializační funkce instancí objektů (Init)

Jedná se o samostatnou funkci vztaženou vždy ke skupině instancí konkrétního typu standardizovaných objektů. Pro každou instanci objektu je ve funkci vytvořen jeden network v jazyce STL, který konfiguruje vnitřní proměnné příslušného instančního datového bloku tak, aby se dosáhlo požadovaného chování objektu.

Jde o proces definování použitých vstupních signálů, přiřazení potřebných parametrů, jejich rozsahů a požadovaných vlastností dané instance objektu. Pro snazší orientaci programátora mají inicializační networky výchozí strukturu, jejíž část se při změně parametru za komentuje a změněný parametr se uvede navrch funkce. K této informaci se vrátíme později v praktické části práce.

Aby byly zajištěny jejich definované vlastnosti po celou dobu vykonávání programu, je tato funkce volána v každém cyklu hlavního programu v PLC.

2.3. Rozhraní TIA Portal Openness

V této kapitole popíšeme rozhraní TIA Portal Openness vytvořené společností Siemens pro jejich integrované vývojové prostředí TIA Portal, které bylo stručně popsáno v předchozích kapitolách.

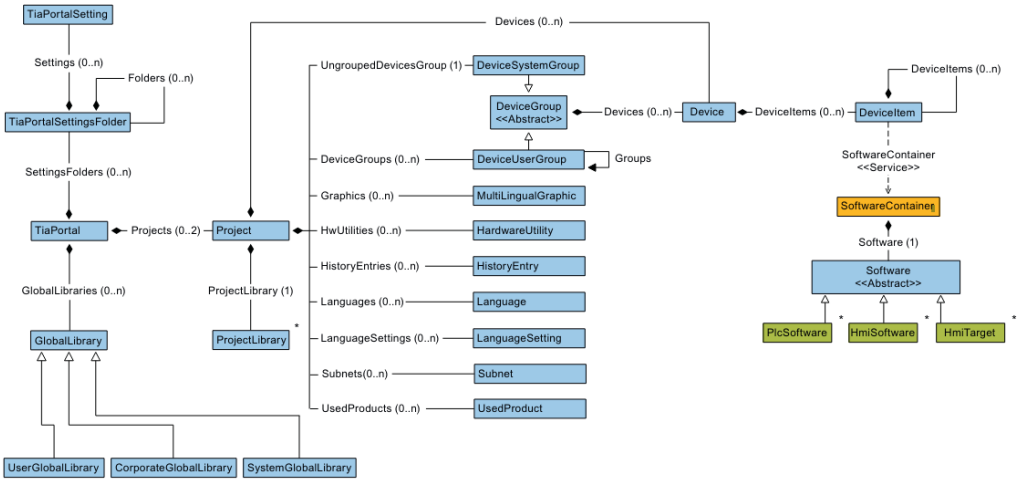
V následujících sekcích tedy popíšeme co vlastně toto rozhraní je, jak se s ním pracuje a co nám umožňuje dělat. Ještě je vhodné zmínit, že hlavním a téměř jediným relevantním dostupným zdrojem je hlavní dokumentace k rozhraní přímo od společnosti Siemenes. Budeme se odkazovat na dokumentaci V17, avšak v době psaní této práce už vyšla i V19. Užitečným zdrojem pro čtenáře mohou být i Demo aplikace, připravené společností Siemenes k demonstrování funkce TIA Openness, které jsou k dispozici na jejich fóru.

2.3.1. Obecný popis

TIA Portal Openness popisuje otevřené rozhraní pro práci s prostředím TIA Portal. Prostřednictvím tohoto rozhraní lze automatizovat vývojové procesy pomocí vlastního externího programu [3].

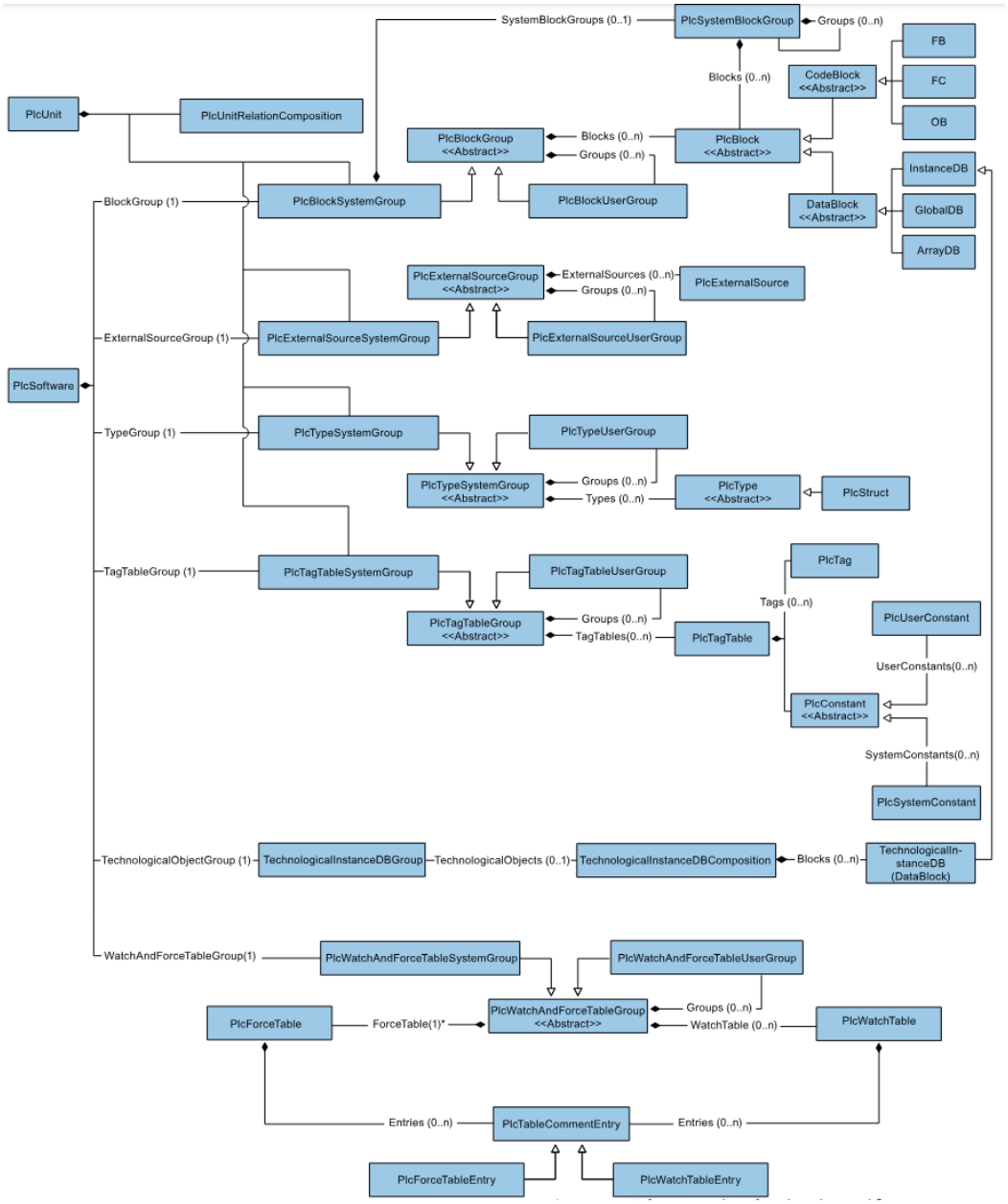
Externí program využívající rozhraní TIA Openness může zakládat nové projekty do prostředí TIA Portalu, nebo vytvářet, číst, mazat a editovat obsah a data v již existujících projektech [3].

Následující obrázek číslo 4 znázorňuje nejvyšší úroveň objektového modelu rozhraní TIA Openness. Pro účely této práce, se budeme soustředit převážně na větev PlcSoftware → DeviceItem → Device → Project → TiaPortal. Neboli posloupnost, která nám umožní čtení a modifikaci obsahu konkrétního PLC. Převážně, ale ne pouze, jeho programové struktury.



Obrázek 4 Nejvyšší úroveň objektového modelu rozhraní TIA Portal Openness [3]

Na obrázku číslo 5 je pak znázorněn objektový model rozhraní TIA Portal Openness přímo pro část PlcSoftware. Z tohoto obrázku jsou pro účely naší práce relevantní části vztažené k ‚PlcBlock', ‚PlcBlockGroup' a ‚PlcTypeSystemGroup'. Tyto větve nám umožní manipulovat s programovými bloky projektu (FB, FC, DB), strukturou adresářů v projektu, tak i s UDT obsažených v projektu.



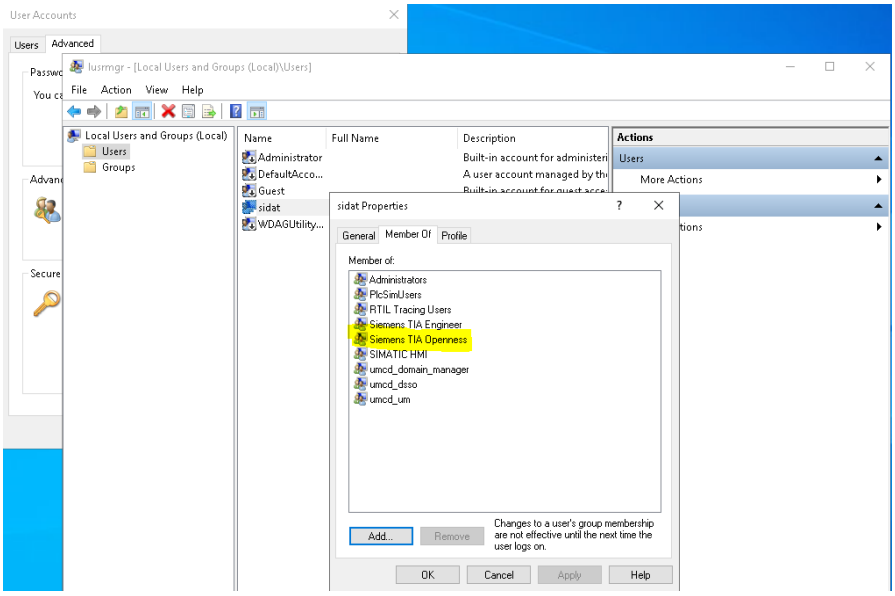
Obrázek 5 Objektový model struktury PlcSoftware rozhraní TIA Portal Openness [3]

2.3.2. Programové připojení k rozhraní

Rozhraní TIA Portal Openness lze na zařízení volitelně nainstalovat během instalačního procesu prostředí TIA Portal. Alternativním přístupem je získat rozhraní instalací add-on balíčku „TIA Portal Openness“ například do programového prostředí Visual Studio. V takovém případě však není na zařízení automaticky nastavená uživatelská role „Siemens TIA Openness“, která by uživatele opravňovala k využití tohoto rozhraní [3][4].

Zde je třeba zmínit, že tato role musí být nastavená na všech zařízeních, kde bude vyvinutá aplikace nasazena. Nestačí tedy mít tuto roli pouze na zařízení, kde dochází k vývoji této aplikace. V dokumentaci [3] je popsán postup, jak lze uživateli roli „Siemens TIA Openness“

přiřadit, avšak rychlejší je zadání příkazu „netplwiz“ do Windows vyhledávače a přiřazení této role vybraným uživatelským účtům, jak ukazuje obrázek číslo 6.



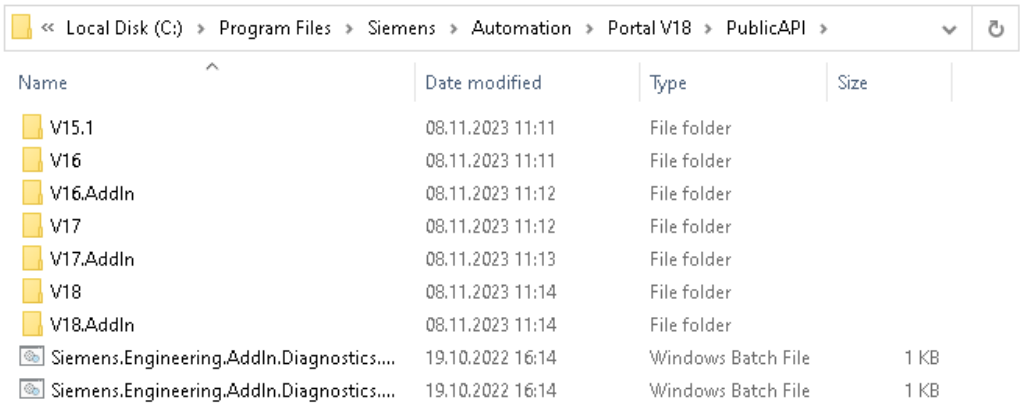
Obrázek 6 Znázornění přiřazení uživatelské role Siemens TIA Openness

Chceme-li k tomuto rozhraní přistoupit programově z naší vytvořené aplikace, je nezbytné zahrnout do projektu knihovny „Siemens.Engineering“, případně „Siemens.Engineering.Hmi“. Toho se dá docílit pomocí vytvoření reference v projektu k souboru „Siemens.Engineering.dll“, nebo jeho načtením v konfiguračním souboru aplikace [3].

V pozdější kapitole, využijeme pro kompatibilitu aplikace s různými verzemi prostředí TIA Portal skutečnosti, že od TIA Portal Openness V14 je součástí každé instalované verze registrový klíč s informacemi o aktuální verzi, a lze jej nalézt pod následující cestou [3]:

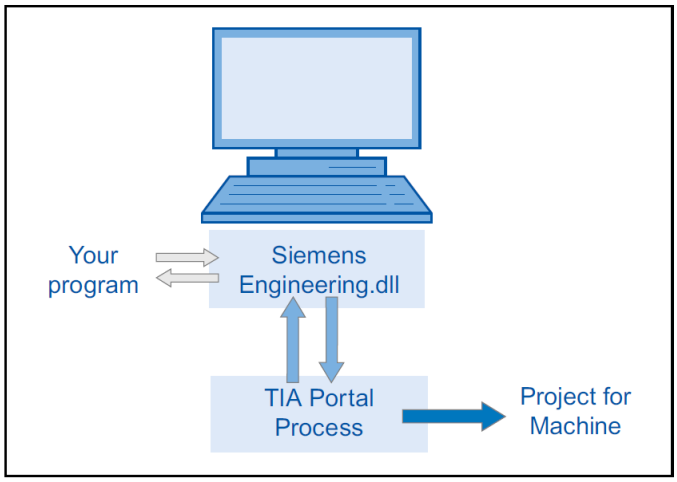
`HKEY_LOCAL_MACHINE\Software\Siemens\Automation\Openness\xx.x\PublicAPI`

Obrázek číslo 7 pak ukazuje obsah adresáře nacházejícího se na zmíněné cestě s tím, že každý uvedený adresář má v sobě uložené již dříve zmíněné .dll soubory a soubory typu XML schema [3].



Obrázek 7 Obsah adresáře na cestě z registrového klíče, který ve složkách obsahuje soubory Siemens.Engineering.dll

Po přidání reference souboru „Siemens.Engineering.dll“ do vyvíjeného projektu, lze jeho prostřednictvím přistupovat k procesům v prostředí TIA Portal pomocí externích aplikací [3]. Toto je demonstrováno na obrázku číslo 8 s tím, že se zároveň jedná i o princip využitý v naší práci.



Obrázek 8 Přehled přístupu k procesu TIA Portalu pomocí externí aplikace [3]

2.3.3. Vybrané podporované funkce

Pro splnění zadání této práce potřebujeme po rozhraní TIA Portal Openness možnost připojit se vyvíjenou aplikací k projektu v prostředí TIA Portal, iterovat hierarchií objektů, exportovat bloky kódu k získání předloh a následně importovat bloky kódu vytvořené na základě dat z uživatelského vstupu a zmíněných předloh.

V následujících podkapitolách budou stručně uvedeny snímky kódů z oficiální dokumentace a případný komentář ohledně jejich využití. Jedná se ale v podstatě o funkce , pomocí kterých budeme postupovat objektovým modelem TIA Portal Openness znázorněným na obrázcích 4 a 5 na začátku kapitoly 2.3.

2.3.3.1. Spuštění TIA Portalu

TIA Portal lze spustit dvěma způsoby: pomocí vyvíjené aplikace využívající rozhraní TIA Portal Openness, nebo klasickým způsobem, kdy uživatel spustí nainstalovanou aplikaci TIA Portal na svém zařízení [3].

Pokud přistoupíme ke spuštění procesu TIA Portalu pomocí vyvíjené aplikace, máme díky rozhraní TIA Portal Openness dvě možnosti [3]:

- **Spuštění s uživatelským rozhraním** – Využitím této možnosti je spuštěná instance TIA Portalu stejná, jako když uživatel ručně spustí nainstalovanou aplikaci na svém zařízení [3].
- **Spuštění bez uživatelského rozhraní** – V tomto případě, se spuštěná instance TIA Portalu vytvoří bez uživatelského rozhraní. To znamená, že proces běží na pozadí a uživatel s ním manipuluje exkluzivně přes rozhraní TIA Portal Openness [3]. Výhodou tohoto přístupu je především rychlejší import dat [3], který na základě našeho testování byl v některých případech i dvojnásobně rychlý.

Příklad úryvku kódu ke spuštění instance TIA Portalu pomocí rozhraní TIA Openness je uveden na obrázku 9. Pro spuštění bez uživatelského rozhraní stačí akorát přepsat „TiaPortalMode.WithUserInterface“ na „TiaPortalMode.WithoutUserInterface“. [3]

```
using (TiaPortal tiaPortal = new TiaPortal(TiaPortalMode.WithUserInterface))
{
    //begin of code for further implementation
    //...
    //end of code
}
```

Obrázek 9 Příklad programového spuštění TIA Portalu pomocí rozhraní TIA Portal Openness s uživatelským rozhraním [3]

2.3.3.2. Připojení k projektu

K využití funkcionalit rozhraní TIA Portal Openness vyvíjenou aplikací je nejprve potřeba stanovit programové připojení ke konkrétnímu projektu v prostředí TIA Portal, se kterým budeme chtít zacházet. Abychom toho docílili, lze postupovat tak, že program nejprve detekuje běžící procesy TIA Portalu, připojí se k jednomu z nich, zkontroluje, zda je otevřený projekt a následně se k němu připojí [3].

Když zanedbáme kontrolu úspěchu mezi-operací a možnost, že žádný proces a projekt nemusí existovat, lze k připojení k otevřenému projektu využít následující sekvenci příkazů [3].

```
IList<TiaPortalProcess> processes = TiaPortal.GetProcesses();
TiaPortal myTiaPortal = processes[0].Attach();
Project myProject = myTiaPortal.Projects[0];
```

2.3.3.3. Export souborů

Pro účely naší práce budeme potřebovat rozhraní TIA Portal Openness využít k exportování programových bloků a uživatelsky definovaných typů (UDT), abychom mohli získat předlohy k později generovaným XML souborům. Rozhraní nabízí funkce k exportu obou těchto typů s tím, že soubory exportované z projektu uloží jako XML soubory na poskytnutou cestu [3].

Export programových bloků

Rozhraní TIA Portal Openness umožňuje exportovat pouze konzistentní (zkompilované) programové bloky do formátu XML. Funkce pro export programových bloků podporuje následující typy bloků: „Funkční bloky (FB), funkce (FC), organizační bloky (OB a datové bloky (DB)“. A následující typy programových jazyků: „STL, FBD, LAD, SCL, GRAPH“ [3]. Export lze do XML formátu provést pomocí funkcí uvedených na obrázku 10.

```
private static void ExportRegularBlock(PlcSoftware plcSoftware)
{
    PlcBlock plcBlock = plcSoftware.BlockGroup.Blocks.Find("MyBlock");
    plcBlock.Export(new FileInfo(string.Format(@"D:\Samples\{0}.xml", plcBlock.Name)),
ExportOptions.WithDefaults);
}
```

Obrázek 10 Ukázka exportování programového bloku z projektu pomocí rozhraní TIA Portal Openness [3]

Tato funkce, stejně jako všechny další uvedené v této kapitole, předpokládá, že na svém vstupu obdrží objekt typu PlcSoftware [3]. K tomuto objektu lze programově přistoupit iterací přes objektový model rozhraní TIA Portal Openness, viz obrázek 4.

Export uživatelsky definovaných typů

Pro export UDT je potřeba přistoupit k jiné časti objektového modelu rozhraní než pro export programových bloků [3]. Příklad tohoto přístupu a exportu je uveden na obrázku 11.

```
private static void ExportUserDefinedType(PlcSoftware plcSoftware)
{
    string udtname = "udt name XYZ";
    PlcTypeComposition types = plcSoftware.TypeGroup.Types;
    PlcType udt = types.Find(udtname);
    udt.Export(new FileInfo(string.Format(@"C:\OpennessSamples\udts\{0}.xml", udt.Name)),
ExportOptions.WithDefaults);
}
```

Obrázek 11 Ukázka exportování UDT z projektu pomocí rozhraní TIA Portal Openness [3]

2.3.3.4. Import souborů

Další klíčovou funkcí rozhraní, kterou budeme v práci využívat, je obecně import XML souborů zpět do projektu. Zde nás opět bude nejvíce zajímat import programových bloků a UDT.

Import programových bloků

Přes rozhraní lze importovat XML soubory definující programové bloky dle typů uvedených v sekci „Export programových bloků“ [3]. Příklad kódu k importu programových bloků je uveden na následujícím obrázku číslo 12.

```
private static void ImportSystemBlocks(PlcSoftware plcSoftware)
{
    PlcBlockSystemGroup systemblockGroup = plcSoftware.BlockGroup;
    IList<PlcBlock> blocks = systemblockGroup.Blocks.Import(new FileInfo(@"D:\Blocks
\myBlock.xml"), ImportOptions.Override);
}
```

Obrázek 12 Ukázka importování programového bloku do projektu pomocí rozhraní TIA Portal Openness [3]

Import uživatelsky definovaných typů

Příklad funkce k importu XML souboru definujícího UDT je uveden na obrázku 13.

```
private static void ImportUserDataType(PlcSoftware plcSoftware)
{
    FileInfo fullFilePath = new FileInfo(@"C:\OpennessSamples\Import\ExportedPlcType.xml");
    PlcTypeComposition types = plcSoftware.TypeGroup.Types;
    IList<PlcType> importedTypes = types.Import(fullFilePath, ImportOptions.Override);
}
```

Obrázek 13 Funkce importu UDT do projektu pomocí rozhraní TIA Portal Openness [3]

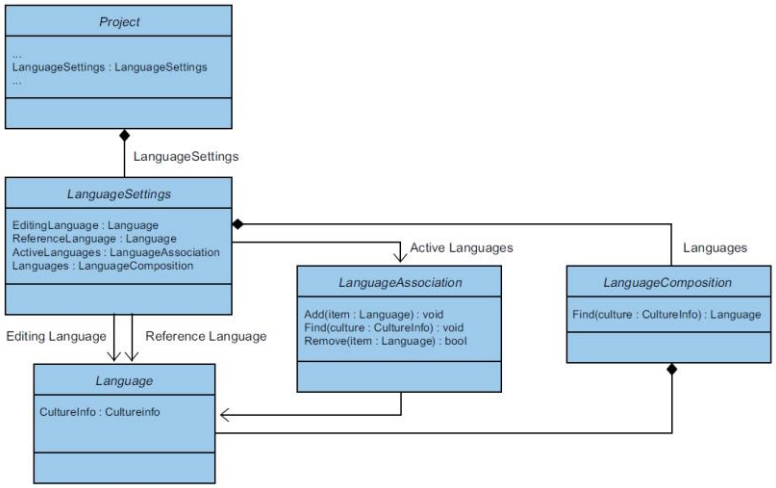
2.3.3.5. Nastavení jazyků projektu

Součástí XML souborů, které vzniknou exportem programových bloků, nebo jsou vytvořeny programově, mohou být i komentáře networků a proměnných. Tyto komentáře mohou být více jazyčné a při jejich importu do projektu je potřeba zajistit, že všechny jazyky definované v souboru XML jsou aktivované i v projektu [3].

Pomocí TIA Portal Openness lze zkontrolovat aktivní jazyky v otevřeném projektu, iterovat přes podporované jazyky a případně aktivovat všechny potřebné jazyky, které v projektu aktivní

nejsou. Tímto způsobem lze zajistit konzistenci při importu XML souborů do cílového projektu [3]. V naší práci budeme této funkcionality využívat prozatím pouze pro český a anglický jazyk, avšak je v plánu rozšíření na všechny typy podporovaných jazyků.

Přístup k jazykům projektu objektovým modelem TIA Portal Openness je znázorněn blokovým schématem uvedeným na obrázku 14 [3].



Obrázek 14 Model nastavení jazyků projektu rozhraní TIA Portal Openness [3]

Diagram na obrázku 14 znázorňuje vazby modelu mezi aktivními jazyky projektu, výčtem podporovaných jazyků a aktuálně nastaveného editačního jazyku [3]. Výpis funkcí a příklad jejich použití k vyhledání podporovaných a nastavení aktivních jazyků projektu je k vidění na obrázku číslo 15.

```
Project project = ...;

    LanguageSettings languageSettings = project.LanguageSettings;

    LanguageComposition supportedLanguages = languageSettings.Languages;
    LanguageAssociation activeLanguages = languageSettings.ActiveLanguages;

    Language supportedGermanLanguage =
supportedLanguages.Find(CultureInfo.GetCultureInfo("de-DE"));
    activeLanguages.Add(supportedGermanLanguage);

    languageSettings.EditingLanguage = supportedGermanLanguage;
    languageSettings.ReferenceLanguage = supportedGermanLanguage;
```

Obrázek 15 Příklad použití funkcí TIA Portal Openness k nastavení jazyků v připojeném projektu [3]

2.4. XML soubory

Vzhledem k tomu, že rozhraní TIA Portal Openness generuje XML soubory při exportu projektových objektů a při importu tyto soubory naopak používá ke specifikaci výsledných objektů, je užitečné krátce vysvětlit, co XML soubory vlastně jsou a jakou mají strukturu. V následujících sekcích tedy stručně zmíníme, co jsou XML soubory, jak jsou strukturované a jaké mají využití i z hlediska používaného rozhraní TIA Portal Openness.

2.4.1. Co jsou XML soubory

Extensible Markup Language (XML) je značkovací jazyk umožňující definovat a ukládat data ve sdílené podobě. XML podporuje výměnu informací mezi počítačovými systémy, jako jsou webové stránky, databáze a aplikace třetích stran. Předem definovaná pravidla usnadňují přenos dat ve formátu XML přes jakoukoliv síť, protože příjemce může tato pravidla použít k jednoznačnému a efektivnímu čtení dat. [5]

XML soubor umožňuje přenášet data spolu s jejich popisem, čímž se zamezuje ztrátě jejich integrity. Popisné informace v souboru pak lze využít k ověření platnosti dat, automatickému přizpůsobení prezentace dat různým uživatelům a konzistentnímu ukládání dat na různých platformách. [5]

2.4.2. Obecná struktura

- **XML tags**: Tagy (značky) se v XML používají k definování dat. Tyto tagy přinášejí sofistikované kódování dat pro integraci informačních toků mezi různými systémy. [5]
- **XML dokument**: Tagy <xml></xml> označují začátek a konec XML souboru. Obsah uvnitř těchto značek se nazývá XML dokument. Je to první tag, který je klíčový pro zpracování XML kódu jakýmkoliv softwarem. [5]
- **XML declaration**: XML dokument začíná uvedením informací o samotném XML souboru, například o použité XML verzi a kódování. Tato úvodní část se nazývá XML deklarace. Příklad: <?xml version="1.0" encoding="UTF-8"?>. [5]
- **XML elements**: Všechny ostatní tagy vytvořené v XML dokumentu se nazývají XML elementy. XML elementy mohou obsahovat text, atributy a další vnořené elementy. Všechny XML dokumenty začínají primární značkou, která se nazývá kořenový prvek. [5]
- **XML attributes**: XML elementy mohou mít další popisné prvky nazývané atributy. Název atributu a jeho hodnota jsou uvedeny v uvozovkách a lze si je volně definovat. Příklad: <person age="22">. [5]
- **XML content**: Data v XML souborech se také nazývají jako obsah XML. [5]

2.4.3. Použití

Soubory typu XML lze využít k přenosu dat mezi dvěma systémy, které například ukládají stejná data, avšak v odlišném formátu. Dále se dají využít pro zobrazování dat na webových stránkách, nebo jakožto datový typ v programech, které pracují s odlišnými programovacími jazyky [5].

V našem případě však bude hlavním využitím XML souborů definování programových bloků, které budeme chtít importovat do cílového projektu. V rámci TIA Portal Openness je struktura XML souboru programového bloku daná jasně definovanou množinou atributů a elementů.

Definice rozhraní exportovaného bloku je pokryta v elementu <Interface> v SimaticML bloku. Kořenovým prvkem je element <Sections>, který definuje parametry rozhraní exportovaného bloku. Níže je výčet elementů, které mohou být obsažené v rámci <Interface> [3].

- **Section** - představuje jeden typ parametrů nebo lokálních data programového bloku. Může se tedy například jednat o sekci specifikující parametry typu ‚Input' [3].
- **Member** - představuje tagy nebo konstanty použité v programovém bloku. V závislosti na datovém typu tagu mohou obsahovat další vnořené tagy typu member nebo obsahovat další strukturované sub-elementy [3].
- **AttributeList** - obsahuje všechny definované atributy člena [3].
- **StartValue** - je zapisována pouze tehdy, pokud uživatel nastaví výchozí hodnotu tagu nebo konstanty [3].
- **Comment** - je vyplněn, pokud jej uživatel nastavil. Komentáře tagu nebo konstanty jsou exportovány jako vícejazyčný text [3].

Na obrázku číslo 16 je znázorněná struktura XML souboru, který definuje instanční datový blok exportovaný z prostředí TIA Portal pomocí rozhraní TIA Portal Openness.

```
<?xml version="1.0" encoding="utf-8"?>
<Document>
  <Engineering version="V17" />
  <DocumentInfo>
    <Created>2024-02-26T15:03:15.1943403Z</Created>
    <ExportSetting>WithDefaults</ExportSetting>
    <InstalledProducts>...</InstalledProducts>
  </DocumentInfo>
  <SW.Blocks.InstanceDB ID="0">
    <AttributeList>
      <AutoNumber>false</AutoNumber>
      <DBAccessibleFromOPCUA>true</DBAccessibleFromOPCUA>
      <DBAccessibleFromWebserver>true</DBAccessibleFromWebserver>
      <HeaderAuthor>MaC</HeaderAuthor>
      <HeaderFamily>MODUL</HeaderFamily>
      <HeaderName>S7_DAMPER</HeaderName>
      <HeaderVersion>2.1</HeaderVersion>
      <InstanceOfName>S7_DAMPER</InstanceOfName>
      <InstanceOfType>FB</InstanceOfType>
      <Interface>
        <Sections xmlns="http://www.siemens.com/automation/Openness/SW/Interface/v5">
          <Section Name="Input">
            <Member Name="MODE_AUTO" Datatype="Bool" Remanence="NonRetain" Accessibility="Public">
              <AttributeList>
                <BooleanAttribute Name="ExternalAccessible" SystemDefined="true">true</BooleanAttribute>
                <BooleanAttribute Name="ExternalVisible" SystemDefined="true">false</BooleanAttribute>
                <BooleanAttribute Name="ExternalWritable" SystemDefined="true">false</BooleanAttribute>
              </AttributeList>
            </Member>
            ...
            <Member Name="EXCITATION_TIMEOUT" Datatype="TOF_TIME" Version="1.0" Remanence="NonRetain" Accessibility="Public">
              <AttributeList>...</AttributeList>
              <Sections>...</Sections>
            </Member>
          </Section>
        </Sections>
      </Interface>
      <Name>Instance_NAME</Name>
      <Number>17002</Number>
      <ProgrammingLanguage>DB</ProgrammingLanguage>
    </AttributeList>
    <ObjectList>
      <MultilingualText ID="1" CompositionName="Comment">
        <ObjectList>...</ObjectList>
      </MultilingualText>
      <MultilingualText ID="4" CompositionName="Title">
        <ObjectList>...</ObjectList>
      </MultilingualText>
    </ObjectList>
  </SW.Blocks.InstanceDB>
</Document>
```

Obrázek 16 XML soubor instančního datového bloku získaný exportem z TIA Portalu

2.4.4. XML Schema

XML Schema je dokument, který popisuje pravidla nebo omezení pro strukturu XML souboru. Tato omezení lze popsat různými způsoby a následující výčet slouží jako ilustrační příklad [5].

- Gramatická pravidla určující pořadí prvků
- Podmínky typu "ano" nebo "ne", které musí obsah splňovat
- Datové typy pro obsah XML souborů
- Omezení pro integritu dat [5]

V našem zmiňujeme dokumenty typu XML schema, jelikož se využívají na pozadí pro validaci XML souborů, které se importují přes rozhraní TIA Portal Openness do projektu v prostředí TIA Portal. Sice lze funkcí rozhraní k importu souborů využívat i bez znalosti existence souborů XML schema, ale v jistých případech se může jednat o užitečnou znalost.

Dokumenty typu XML schema lze nalézt na cestě instalované verze TIA Portalu, konkrétně: „C: \Program Files\Siemens\Automation\Portal Vxx\PublicAPI\Vxx\Schemas\“ [3]

Část souboru typu XML schema je pro demonstrativní účely uvedená na obrázku 17

```
    <xs:group name="PartSequence_G">
        <xs:sequence>
            <xs:element ref="TemplateValue" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="AutomaticTyped" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="Invisible" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="Negated" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="Comment" minOccurs="0"/>
        </xs:sequence>
    </xs:group>
    <xs:simpleType name="PinName_TE">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:element name="Wire" type="Wire_T"/>
    <xs:complexType name="Wire_T">
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="Powerrail"/>
            <xs:element ref="NameCon"/>
            <xs:element ref="IdentCon"/>
            <xs:element ref="Openbranch"/>
            <xs:element ref="OpenCon"/>
        </xs:choice>
        <xs:attribute name="UId" type="xs:int" use="required"/>
    </xs:complexType>
    <xs:element name="Wires" type="Wires_T"/>
    <xs:complexType name="Wires_T">
        <xs:sequence maxOccurs="unbounded">
            <xs:element ref="Wire"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Obrázek 17 Příklad části dokumentu typu XML schema pro TIA Portal Openness (SW.PlcBlocks.LADFBD_v4.xsd)

3. Praktická část

V této kapitole se zaměříme na návrh a realizaci procesu, jehož hlavním cílem je generovat instance standardizovaných objektů v prostředí TIA Portal pomocí rozhraní TIA Portal Openness na základě uživatelského vstupu v aplikaci Excel. Hlavní motivací pro tuto činnost je snížení celkového množství práce a času, který musí programátor věnovat přípravě firemní standardizované struktury u nových projektů.

V této kapitole budeme probírat především následující body:

- **Struktura navrženého řešení** – Tato sekce rozebereme rozdělení celkového řešení na jednotlivé části a jejich vzájemné interakce. Půjde hlavně o obecný přehled, který podrobněji rozvede následující kapitola.
- **Implementace generování instancí objektů pro PLC** – Tato sekce se zaměří na jednotlivé kroky potřebné k vygenerování instancí standardizovaných objektů od zadání uživatelského vstupu až po import vytvořených XML souborů do projektu. Popisuje se zde hlavní část této práce a způsob její implementace.
- **Implementace generování HMI struktur** – V této části se budeme zabývat způsobem dodatečného zpracování dat z uživatelského vstupu a jak tato data využijeme k tvorbě HMI tagů, alarmových hlášení a textových listů pro operátorské panely řady Comfort a Comfort Unified.

3.1. Struktura navrženého řešení

Jak již bylo zmíněno, cílem této kapitoly je popsat celkovou strukturu navrženého procesu generování instancí standardizovaných objektů do prostředí TIA Portal. Rozebereme způsob zadávání uživatelského vstupu, jeho zpracování a tvorbu XML souborů, které následně importujeme do projektu.

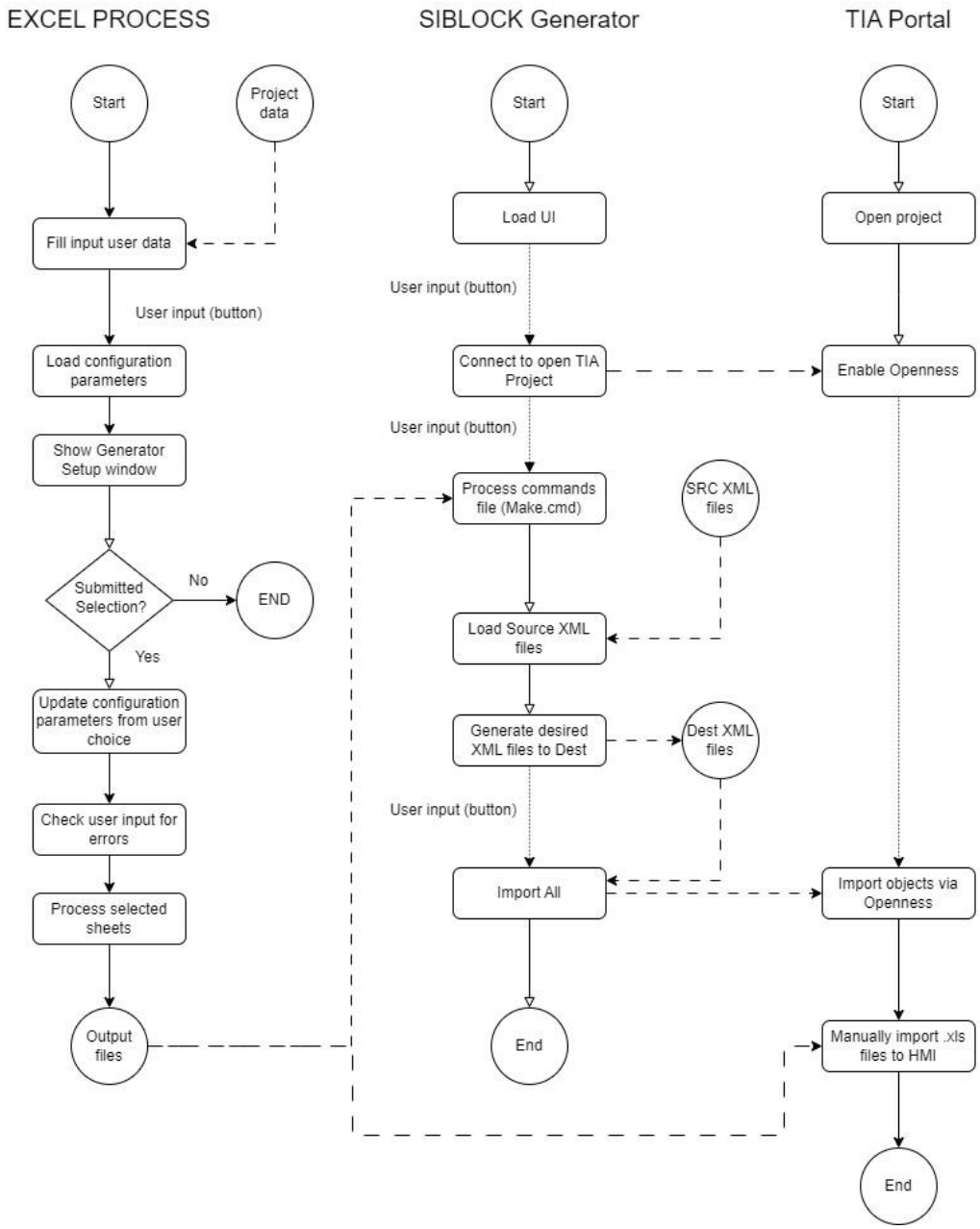
Je důležité zmínit, že vývoj projektu popisovaného v této práci nezačal zcela od začátku, ale navázal na prototyp dříve vyvíjený ve firmě Sidat. Z tohoto prototypu vycházejí některé fundamentální koncepce a nevyužité elementy, kterých si lze všimnout v pozdějších částech zprávy. Ačkoliv se z tohoto prototypu vycházelo, nejednalo se o plně funkční verzi, natož o stávající podobu prezentované práce.

Použitý způsob řešení vychází převážně z požadavků firmy Sidat, kde byl vzhledem k množství objektů a parametrů kladen důraz na jednoduchost zadání uživatelského vstupu v rámci aplikace Excel. Tento vstup je následně potřeba zpracovat a předat dalším procesům pro navazující operace.

Jelikož k zadání uživatelského vstupu nemusí docházet na stejném zařízení, kde dojde k importu vygenerovaných instancí objektů, byla struktura projektu rozdělena na část procesu Excelu a část aplikace generátoru psaného v jazyce C# a využívajícího rozhraní TIA Portal Openness.

Aplikace generátoru pak na základě zpracovaných vstupních dat a předpřipravených knihovních XML souborů vytvoří nové XML soubory, které při importu do projektu vytvoří požadované instance objektů. Hlavní části procesu automatického generování jsou znázorněny

na obrázku číslo 18. Význam těchto částí a jejich implementaci budeme popisovat napříč touto prací.



Obrázek 18: Zjednodušená logická struktura generování standardizovaných objektů do prostředí TIA Portal pomocí rozhraní TIA Openness

3.1.1. Zadání uživatelského vstupu

Úvodem této kapitoly byl zdůrazněn požadavek systematického a přehledného zadání uživatelské specifikace objektů, což je důležité zejména u rozsáhlejších projektů, kde množství instancí standardizovaných objektů může dosahovat stovek. Pro každou tvořenou instanci objektu musí být zároveň pečlivě definovány připojené vstupní/výstupní signály i jeho interní

parametry. Proto byla pro zadání uživatelského vstupu vybrána aplikace Excel, jejíž předností je efektivita a transparentnost při zadávání velkého množství dat.

Navržená specifikace objektů se provádí ve strukturované formě v Excelu, kde jsou podporované typy standardizovaných objektů kategorizovány do jednotlivých listů sešitu. Každý list reprezentuje konkrétní typ objektu, jeho řádky představujícími individuální instance a sloupce pak určují parametry a vlastnosti dané instance.

V kapitole 2.2 jsme zmínili, že standardizované objekty byly psány s ohledem na univerzálnost jejich nasazení pro různé typy projektů. Tato skutečnost znamená, že velká většina možných parametrů konkrétní instance standardizovaného objektu zůstane nevyužitá.

To nám z hlediska uživatelského vstupu dovoluje stanovit výchozí nastavení, kde nevyplněné buňky v tabulce automaticky přebírají výchozí nastavení parametrů. Tím se značně ulehčuje proces uživatelské specifikace, umožňuje efektivní nastavení a snižuje prostor pro uživatelské chyby, jelikož se uživatel nemusí zabývat nevyužitými parametry instance objektu.

3.1.2. Zpracování dat z uživatelského vstupu

Tato sekce popisuje metodu zpracování dat zadaných v uživatelském vstupu, což je nezbytný krok pro následné generování požadovaných instancí objektů v dalších fázích projektu. Jak již bylo uvedeno, nevyplněné parametry objektu jsou při zpracování automaticky doplněny výchozími hodnotami pro daný typ objektu. Výjimkou jsou parametry identifikující danou instanci a hodnoty konstant.

Proces zpracování dat probíhá dle části procesu Excel uvedené na obrázku číslo 18. V jednotlivých krocích využívá různé předpřipravené šablony a je strukturován následovně:

1. **Konfigurace** - Uživatel po stisknutí tlačítka ‚GENEROVAT' specifikuje na uživatelském rozhraní konfiguračního okna, které typy standardizovaných objektů chce generovat a jaké funkce si přeje při zpracování vstupních dat potlačit. Následně uvede cestu, na kterou se umístí vytvořené soubory
2. **Detekce chyb vyplněných dat** – Před zahájením zpracování dat vybraných typů objektů nejprve proběhne série předpřipravených testů, které se pokusí odhalit chybně vyplněné parametry a případné duplicity instancí objektů.
3. **Generování souboru pro HMI struktury** – Volitelný krok, který se provede pouze pokud uživatel zvolí generování standardu pro panely řady Comfort nebo Comfort Unified. Na základě uživatelem zvolených typů objektů a daného typu panelu se následně načtou šablony a vytvoří se soubory xlsx definující HMI Tagy, alarmové hlášení a textové listy.
4. **Generace příkazového souboru Make** - Posledním a hlavním krokem je zpracování uživatelského vstupu ke tvorbě programových instancí vybraných standardizovaných objektů. Toto zpracování probíhá na základě předpřipravené šablony a jeho výstupem je soupis příkazů v příkazovém souboru ‚Make'. Ten poskytuje podrobný návod, jak v navazujících krocích sestavit XML soubory, aby bylo možné vytvořit požadované instance objektů.

3.1.3. Zpracování Make souboru a tvorba XML souborů

Rozhraní TIA Portal Openness podporuje k importu do projektu v prostředí TIA Portal pouze soubory typu XML. To znamená, že potřebujeme na základě dat z uživatelského vstupu vytvořit sadu XML souborů, které budou definovat požadované instance standardizovaných objektů. Vytvořené XML soubory navíc musí dodržovat formát dle programovacího jazyka a typu programového bloku, který je kompatibilní s importem do prostředí TIA Portal.

Jelikož mají XML soubory striktně definovanou strukturu, využijeme tento fakt k naší výhodě, jelikož známe výslednou podobu generovaných programových bloků. Vytvářené programové bloky jsou dané firemní standardizovanou strukturou a mají v zásadě neměnnou strukturu, která se liší pouze v napojených signálech a interních parametrech. Z prostředí TIA Portal nám tedy stačí získat jejich podobu ve formátu XML a tyto předlohy pak pouze kopírovat a upravovat při tvorbě XML souborů specifikujících generované instance. Tyto vzorové XML funkce standardizovaných struktur budeme dále nazývat jako **knihovní soubory**.

Při generování instancí standardizovaných objektů kombinujeme data z příkazového souboru získaného z uživatelského vstupu s knihovními soubory získanými z TIA Portalu. Jedná se o část procesu, který náleží větvi SIBLOCK generátor z obrázku číslo 18.

Proces začíná otevřením knihovního XML souboru pro zvolený typ objektu, z něhož se do cílového XML souboru zkopírují sekce (networky) potřebné pro požadovanou instanci. Následně se klíčová slova v těchto sekcích nahradí údaji získanými z uživatelského vstupu. Tento postup se opakuje pro všechny instance požadovaných typů objektů, čímž získáme kompletní sadu potřebných XML souborů, které jsou připraveny k importu do projektu.

3.1.4. Využití rozhraní TIA Portal Openness

Pro práci s rozhraním TIA Portal Openness je nezbytné využít knihovnu Siemens.Engineering.dll, jak bylo podrobněji objasněno v kapitole 2.3.2 teoretické části této práce. Pro manipulaci s touto knihovnou byl zvolen programovací jazyk C#, protože příklady funkcí v dokumentaci Siemens a vzorové projekty demonstrující využití rozhraní jsou psány právě v tomto jazyce. Tato volba nám tedy umožňuje snadněji využívat rozsáhlou dokumentaci a snadno integrovat existující vzorové kódy do naší aplikace

V C# jsme vyvinuli aplikaci s uživatelským rozhraním, která umožňuje uživatelům využívat vybrané funkce rozhraní TIA Portal Openness. Hlavními součástmi podporovaných funkcí jsou programové spouštění prostředí TIA Portal, manipulace s jeho projekty, export programových bloků a UDT z projektu do formátu XML a import vybraných XML souborů zpět do projektu.

Vyvinutá aplikace pak nabízí další funkcionality rozhraní, jako nastavení potřebných jazyků projektu a automatická tvorba grup (adresářů) v projektu dle hierarchie adresáře, ze kterého jsou aktuálně importovány XML soubory.

3.2. Implementace generování instancí standardizovaných objektů pro PLC

Tato kapitola popisuje implementaci hlavního cíle práce - automatického generování instancí standardizovaných objektů v prostředí TIA Portal s využitím rozhraní TIA Portal Openness. Celý proces zahrnuje použití několika vývojových prostředí a souborů, což zajišťuje přechod od vstupních dat až k finálním programovým blokům specifikující instance objektů vytvořených v projektu v prostředí TIA Portal.

Klíčové komponenty v procesu generování jsou aplikace Excel pro zadání uživatelského vstupu, aplikace generátoru napsaná v jazyce C# pro tvorbu XML souborů a jejich import do projektu, a TIA Portal pro přípravu knihovních souborů.

Struktura vzájemných vazeb mezi zmíněnými komponentami je zobrazena na obrázku číslo 18 z kapitoly 3.1. - Struktura navrženého řešení. V této kapitole však bude již detailněji rozebrána struktura jednotlivých segmentů zmíněného obrázku a jejich implementace. Zmíníme také postup tvorby knihovních souborů a jejich následné využití při vytváření výsledných XML souborů.

3.2.1. Tvorba knihovních souborů

Jak již bylo zmíněno, knihovní soubory v kontextu našeho projektu popisují soubory, ze kterých se vychází při tvorbě XML souborů, které popisují požadované instance standardizovaných objektů. Jedná se o soubory ve formátu XML získané exportem předpřipravených programových bloků dle standardizované struktury z projektu v prostředí TIA Portal.

Následující kapitoly tedy čtenáře seznámí se specifikací instancí standardizovaných objektů ve vzorovém projektu, s procesem extrakce souborů v XML formátu, a ukážeme formát získaných XML souborů, které budeme využívat jako knihovní soubory. Výstupem těchto kapitol budou XML soubory označené zkratkou Src na obrázku číslo 18, který znázorňuje celkovou strukturu řešení.

3.2.1.1. Příprava předloh v projektu TIA Portalu

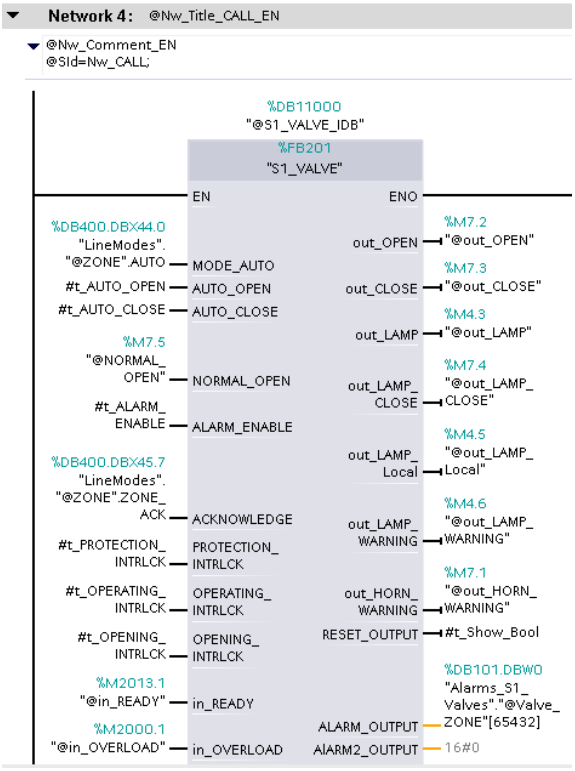
Abychom mohli začít generovat instance objektů, musíme nejprve vědět, jak má vypadat jejich výsledná XML podoba, kterou budeme následně importovat do projektu. Tuto podobu sice ukážeme později, ale lze říci, že ke změně názvů parametrů a signálů v daném programovém bloku stačí pouze upravit textový obsah ve výsledných XML souborech.

Vytvořili jsme tedy projekt, ve kterém mají vzorové instance standardizovaných objektů navázané parametry na zástupné symboly, které určuje identifikátor „@“ v prefixu jejich názvu. Na obrázku 19 je vidět příklad využití identifikátoru k definování zástupných parametrů v instanci funkce volání standardizovaného objektu ventilu. Tyto parametry nám později umožňují u generovaných XML souborů dynamicky měnit jak název networku, jeho komentář, tak i název datového bloku, na který je tato instance napojená, a názvy příslušných vstupních a výstupních signálů, dle potřeb instance objektu.

V komentáři každého networku je pak navíc uveden parametr „@SId=“, který definuje jednoznačný identifikátor vzorového networku v příslušné funkci. Tato funkcionalita nám

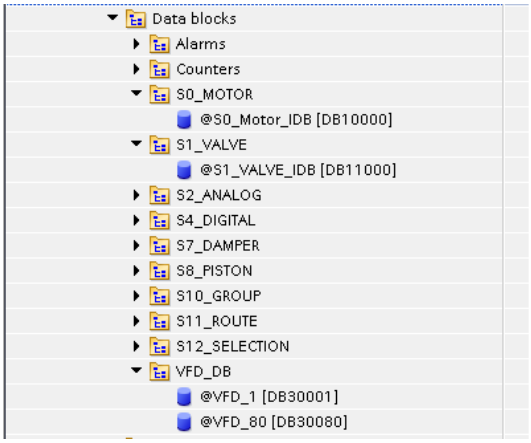
umožňuje v pozdějších krocích generování určit, které operace se aplikují na který vzorový network, a kterou část z předdefinovaných knihovních souborů má program při tvorbě nových XML souborů využít.

Příkladem využití identifikátoru networku může být generování frekvenčních měničů u instancí objektu motoru. Generátor v takovémto případě potřebuje vycházet z jiného vzorového networku než normálně, aby do programu umístil jak instanci motoru, tak i požadovaný typ frekvenčního měniče. Parametr „@SId=“ a napojení zástupných symbolů na volanou instanci funkčního bloku ventilu je znázorněno na obrázku číslo 19.

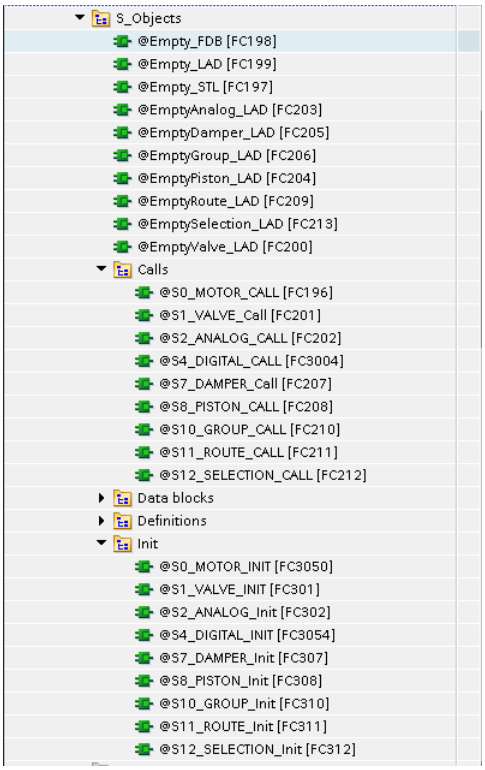


Obrázek 19 Příklad části Networku uvnitř funkce volání instance objektu ventilu sloužící jako knihovní soubor

Podobný princip využití identifikátoru je aplikován i na názvy funkcí a datových bloků, jelikož se jedná o nastavitelný parametr v XML souboru. Na obrázcích číslo 20 a 21 je vidět formát specifikace datových bloků a funkcí v adresářových strukturách.



Obrázek 20 Příklad využití zástupných symbolů v názvu instančních datových bloků v TIA Portalu



Obrázek 21 Zástupný identifikátor aplikovaný v názvech funkcí v TIA Portalu

Na obrázku 21 je vidět struktura projektu, ze kterého se získávají knihovní soubory exportem do formátu XML. Funkce s prefixem „@Empty“ slouží v pozdějších krocích generování k vykopírování nosné XML struktury a vnitřních proměnných příslušných funkcí do nového, generovaného souboru. Tento generovaný soubor se pak dynamicky rozšiřuje kopírováním networků z knihovních souborů, u kterých se rovnou za zástupné symboly dosazují požadované hodnoty z uživatelského vstupu.

Knihovní soubory pro funkce Init instancí standardizovaných objektů mají odlišnou strukturu. Hlavním důvodem je, že inicializační funkce jsou pro lepší čitelnost sepsané v jazyce STL místo jazyka LAD. Struktura je definována po samostatných řádcích a lze je takto odstraňovat. Z hlediska standardu mají funkce Init vždy výchozí strukturu, která při změně

zůstane pouze v komentáři a změněný parametr se zapíše na vrch daného networku Init funkce. Návrh tedy počítá s tím, že vytvoříme oba možné scénáře pro každý parametr a při generování jeden z nich odstraníme. Struktura takovéto připravené funkce je k vidění na obrázcích číslo 22 a 23, kde první z nich ukazuje část výchozí a druhý část modifikovanou.

```
// ----------------------------------DEFAULTS-------------------------
//Signal definitions
      SET
      R     "@Sl_VALVE_IDB".Definitions.Use_Ready                    %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Ready
      R     "@Sl_VALVE_IDB".Definitions.Use_Overload                 %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Overload
      R     "@Sl_VALVE_IDB".Definitions.Use_Feedback_Open            %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Feedback_Open
      R     "@Sl_VALVE_IDB".Definitions.Use_Feedback_Close           %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Feedback_Close
      R     "@Sl_VALVE_IDB".Definitions.Use_LOTO_Lock                %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_LOTO_Lock
      R     "@Sl_VALVE_IDB".Definitions.Use_Close                    %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Close
      R     "@Sl_VALVE_IDB".Definitions.Use_Air                      %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Air
      R     "@Sl_VALVE_IDB".Definitions.Use_Local                    %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Local
      R     "@Sl_VALVE_IDB".Definitions.Use_Positioner               %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Positioner
      R     "@Sl_VALVE_IDB".Definitions.Use_Horn                     %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Horn
      R     "@Sl_VALVE_IDB".Definitions.Use_Remote                   %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Use_Remote

// Object definitions
      SET
      R     "@Sl_VALVE_IDB".Definitions.Mode_Change_Intern           %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.Mode_Change_Intern
      R     "@Sl_VALVE_IDB".Definitions.SP_Visible                   %DB11000.…
//      R     "@Sl_VALVE_IDB".Definitions.SP_Visible
```

Obrázek 22 Výchozí část inicializační funkce knihovního objektu ventilu v prostředí TIA Portal

```
// Object definitions
      SET
      S     "@Sl_VALVE_IDB".Definitions.Mode_Change_Intern           %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.SP_Visible                   %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.Position_Visible             %DB11000.…
      R     "@Sl_VALVE_IDB".Definitions.Restart_After_Fault          %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.LocalCMD_InManual            %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.Manual_InLocal               %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.Close_InManual               %DB11000.…
      S     "@Sl_VALVE_IDB".Definitions.Reset_OUT_inFeedback         %DB11000.…

// Setpoint definitions
      L     -154321.1                                                -154321.1
      T     "@Sl_VALVE_IDB".Definitions.Setpoint_Low                 %DB11000.…

      L     -154321.2                                                -154321.2
      T     "@Sl_VALVE_IDB".Definitions.Setpoint_High                %DB11000.…

      L     -154321.3                                                -154321.3
      T     "@Sl_VALVE_IDB".Definitions.Position_Hysterese           %DB11000.…

// Parameters (Timers)
      L     T#-24d_20h_26m_26s_001ms                                 T#-24d_20…
      T     "@Sl_VALVE_IDB".Parameters.FeedbackOpenDelay             %DB11000.…

      L     T#-24d_20h_26m_26s_002ms                                 T#-24d_20…
      T     "@Sl_VALVE_IDB".Parameters.FeedbackCloseDelay            %DB11000.…

      L     T#-24d_20h_26m_26s_003ms                                 T#-24d_20…
      T     "@Sl_VALVE_IDB".Parameters.LostPositionDelay             %DB11000.…

      L     T#-24d_20h_26m_26s_004ms                                 T#-24d_20…
      T     "@Sl_VALVE_IDB".Parameters.NotInPositionDelay            %DB11000.…

      L     T#-24d_20h_26m_26s_005ms                                 T#-24d_20…
      T     "@Sl_VALVE_IDB".Parameters.OpenDelay                     %DB11000.…
```

Obrázek 23 Pozměněná část inicializační funkce knihovního objektu ventilu v prostředí TIA Portal

Na obrázku číslo 23 lze vidět, že se pro časové a numerické parametry využívají velice specifické konstantní hodnoty. Tyto hodnoty jsou navrženy tak, aby za běžných okolností nikdy v projektu nenastaly, ale stále zachovávaly datový typ příslušného parametru a byly unikátní.

Funkce těchto konstantních hodnot, v podobě absurdních čísel, je podobná funkci zástupného identifikátoru "@". Konkrétně jde o jednoznačné odlišení parametrů definovaných v knihovních souborech od parametrů generovaných objektů.

3.2.1.2. Exportování souborů z TIA Portalu do XML formátu

V této kapitole se budeme zabývat převedením programových bloků ze vzorového projektu v prostředí TIA Portal, do formátu souborů XML. Tyto soubory totiž později využijeme při tvorbě XML souborů, které budou definovat instance automaticky generovaných objektů.

Pro převod souborů z projektu využijeme aplikaci vyvinutou v rámci této práce, která pomocí rozhraní TIA Openness exportuje programové bloky z projektu do formátu XML a uloží je na definované cestě.

Export pomocí TIA Openness

Jak bylo zmíněno v teoretické části této práce, rozhraní TIA Portal Openness poskytuje možnost exportu jak bloků a UDT z připojeného projektu do formátu XML. Abychom toho dosáhli, musíme nejprve propojit naší aplikaci přes toto rozhraní s cílovým projektem a následně postupovat napříč objektovým modelem TIA Portal Openness až na úroveň požadovaných bloků. Zmíněný objektový model byl uveden na obrázku 4 v kapitole 2.3.1.

Po propojení aplikace s cílovým projektem pomocí rozhraní TIA Portal Openness bude naším cílem exportovat bloky UDT a všechny programové bloky tak, abychom zachovali jejich adresářovou hierarchii. Vzhledem k tomu, že v projektu může být předem neznámý počet vnořených grup (adresářů), použijeme pro tento účel rekurzivní algoritmus.

Princip fungování algoritmu

Aplikace v připojeném projektu iteruje přes všechny instance Device a následně i jejich instance objektu DeviceItem. Ty pomocí služby GetService<SoftwareContainer>() převede a zkontroluje, zda náleží do skupiny PlcSoftware. Zde se jedná o postup napříč nejvyšší úrovní objektového modelu, který je k nalezení na obrázku 4.

Jakmile program najde platnou instanci objektu PlcSoftware, vyvolá dvě rekurzivní funkce. První postupuje napříč objekty spadajícími pod BlockGroup náležící instanci PlcSoftware, a druhá postupuje napříč adresáři spadajícími pod TypeGroup. V prvním případě se jedná o programové bloky a v druhém o uživatelsky definované typy.

Popsaný algoritmus je pro lepší názornost uveden pomocí pseudokódu jako Algoritmus 1 této práce.

Algoritmus 1: Export všech programových bloků z projektu

```
    Input: exportPath
    Output: XML files generated in directory exportPath
1   FOR EACH Device device in ConnectedProject.Devices DO
2       blockPath ← PathCombine(actPath, actBlock.Name)
3       Handle already existing file at blockPath
4       FOR EACH DeviceItem deviceItem in device.DeviceItems
5           swTarget ← GetPlcSoftware(deviceItem)
6           IF controllerTarget is PlcSoftware THEN
7               targetPath ← PathCombine(exportPath, device.Name)
8               ExportPLCBlocks(swTarget.BlockGroup, targetPath)
9               ExportUDT(swTarget.TypeGroup, targetPath)
10          END IF
11      NEXT deviceItem
12  NEXT device
13  END
```

Jelikož je u obou zmíněných rekurzivních funkcí princip velmi podobný, pouze iterují přes objekty jiného typu přes rozhraní TIA Portal Openness, popíšeme stručně pouze funkci zpracovávající objekt typu BlockGroup.

Tato rekurzivní funkce dostává jako argument parametr PlcBlockGroup, což je objekt reprezentující projektový adresář v programových blocích, a cestu, na kterou má exportované soubory ukládat. V každé iteraci funkce rozšíří tuto cestu o název aktuálně obdržené grupy. Pokud adresář dle názvu grupy ještě neexistuje na požadované cestě, funkce ho vytvoří a následně iteruje přes všechny instance objektu PlcBlock náležící do objektu PlcBlockGroup.

Každý z těchto bloků se pokusí pomocí funkce exportu zmíněné v kapitole 2.3.3.3 teoretické části exportovat do formátu XML a uložit ho na aktuální cestu prodlouženou o název daného bloku. Po dokončení iterace přes všechny bloky postoupí funkce k dalšímu cyklu, kdy iteruje přes všechny instance typu PlcBlockGroup (adresáře v připojeném projektu) patřící do aktuálně zpracovávaného objektu PlcBlockGroup a v každém kroku vyvolá opět sama sebe s upravenými argumenty. Tato funkce je stručně popsána pomocí pseudokódu uvedeného jako algoritmus 2, pro lepší orientaci čtenáře.

```
Algoritmus 2: Rekurzivní funkce ExportPLCBlocks
    Input: PlcBlockGroup actualBlockGroup, string actualPath
    Output: XML files generated in directory actualPath
1   actualPath ← PathCombine(actualPath, actGroup.Name)
2   Create new directory at actualPath
3   FOR PlcBlock actBlock in actualBlockGroup.Blocks DO
4   |  blockPath ← PathCombine(actPath, actBlock.Name)
5   |  Handle already existing file at blockPath
6   |  OpennessExport(actBlock, blockPath)
7   NEXT actBlock
8   FOR PlcBlockGroup actualGroup in actualBlockGroup.Groups DO
9   |  ExportPLCBlocks(actualGroup, actPath) //Recursive call
10  NEXT actualGroup
11  END
```

Adresářová struktura knihovních souborů

XML soubory získané exportem z projektu jsou organizovány na cílové cestě pomocí adresářové struktury, která reflektuje organizaci programových bloků v projektu. Abychom mohli tyto soubory efektivně využít jako předlohu pro automatické generování, provedeme manuální dodatečné dotřídění.

Rozdělíme adresář s XML soubory bloků z projektu do dvou samostatných adresářů, jejichž obsah se později použije při dvou odlišných operacích procesu generování objektů. Tyto adresáře budou označeny následovně:

- **Src** – Tento adresář obsahuje pouze XML soubory získané ze specificky připraveného projektu, které slouží jako předloha pro dynamické generování instancí standardizovaných objektů. Jedná se o knihovní soubory.
- **Standard** – Tento adresář obsahuje XML soubory, které stačí do projektu importovat pouze jednou a není nutné je dynamicky upravovat. Obsah adresáře může být prakticky libovolný, ale v současné verzi obsahuje objekty potřebné k vytvoření celkové struktury firemního standardu v projektu. XML soubory se dle organizace v tomto adresáři importují roztříděné do složek v projektu.

3.2.1.3. Formát XML souborů

V této sekci ukážeme formát XML souborů získaných exportem z dříve vytvořených knihovních souborů. Formát bude demonstrován pomocí obrázků a popisů, které objasní, ke které části programu daný obrázek patří. Nebudeme popisovat všechny aspekty těchto XML souborů, ale zaměříme se pouze na klíčové části.

Demonstraci formátu předvedeme na funkci pro volání instance standardizovaného objektu motoru. Důležité pro nás bude identifikovat, jak je v XML formátu definována samotná funkce, následně její networky a jak jsou v těchto networcích definovány signály napojené na vstupy a výstupy volané instance funkčního bloku.

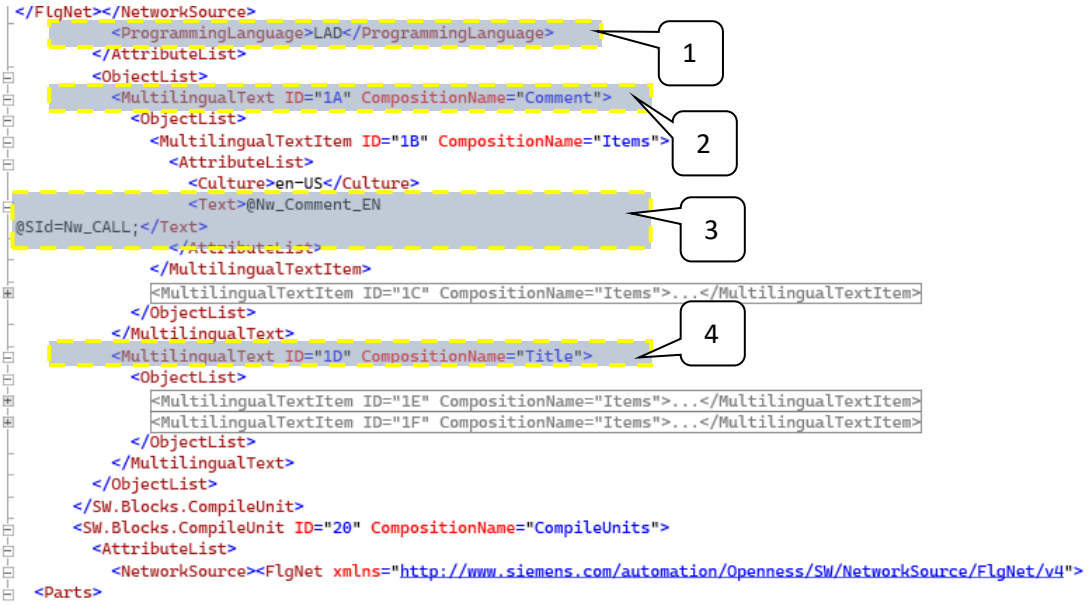
Následující popsaný obrázek číslo 24 znázorňuje hlavičku XML dokumentu, interface (rozhraní) specifikující vnitřní proměnné funkce a unikátní identifikátor funkce v podobě názvu a čísla. V Tabulce číslo 1 jsou pak vysvětlivky k popsaným prvkům obrázku.

Obrázek 24 Popsaná část XML souboru specifikující interface a identifikátor funkce

Tabulka 1 Význam číselných zkratek v popsaném obrázku číslo 24

Položka	Význam
1	Deklarace XML souboru
2	Informace o instalovaných produktech z exportovaného prostředí
3	Interface funkce. Deklarace jejích vstupů, výstupů a dalších typů proměnných
4	Identifikátory funkce. Její název a číslo
5	List objektů ve funkci. Obsahuje všechny networky a kód v exportované funkci

Obrázek 25 označuje specifikaci networku obsaženého ve funkci spolu s jeho programovým jazykem, komentářem a názvem. Tabulka 2 uvádí vysvětlivky k vyznačeným prvkům obrázku. Tato část je uvedena hlavně kvůli tomu, že později při tvorbě XML souborů specifikujících instance generovaných objektů budeme mluvit o kopírování konkrétních networků z předloh dle jejich identifikátoru „@SId“.



Obrázek 25 Popsaná část XML souboru specifikující network funkce

Tabulka 2 Význam číselných zkratek v popsaném obrázku číslo 25

Položka	Význam
1	Programový jazyk daného networku
2	Specifikace komentářů daného networku. Podporuje vícejazyčné texty
3	Zde využíváme speciální anglický komentář @SiD k pozdějšímu cílenému zkopírování daného networku do tvořeného XML souboru.
4	Specifikace názvu/titulku daného networku. Podporuje vícejazyčné texty

Následující obrázky (26, 27 a 28) postupně ukazují specifikaci volaného funkčního bloku v rámci networku. Z programového hlediska nás bude zajímat především parametr specifikující název instančního datového bloku. Obrázky dále popisují „UId“ jednotlivých proměnných použitých v networku a jejich scope, určující, zda se jedná o tag nebo například konstantu. Proměnné se symbolem „@“ jsou připraveny k budoucímu nahrazení za parametry z uživatelského vstupu. Poslední obrázek znázorňuje napojení proměnných na vstupy volaného funkčního bloku dle jejich parametru „UId“.

```
<Call UId="105">
  <CallInfo Name="S0_MOTOR" BlockType="FB">
    <Instance Scope="GlobalVariable" UId="106">
      <Component Name="@S0_Motor_IDB" />
    </Instance>
    <Parameter Name="MODE_AUTO" Section="Input" Type="Bool" />
    <Parameter Name="AUTO_ON" Section="Input" Type="Bool" />
    <Parameter Name="AUTO_ON_REV" Section="Input" Type="Bool" />
    <Parameter Name="AUTO_SLOW" Section="Input" Type="Bool" />
    <Parameter Name="AUTO_EXCITATION" Section="Input" Type="Bool" />
    <Parameter Name="IN_Reserve_0_5" Section="Input" Type="Bool" />
    <Parameter Name="ALARM_ENABLE" Section="Input" Type="Bool" />
    <Parameter Name="ACKNOWLEDGE" Section="Input" Type="Bool" />
    <Parameter Name="PROTECTION_INTRLCK" Section="Input" Type="Bool" />
    <Parameter Name="PROTECTION_INTRLCK_REV" Section="Input" Type="Bool" />
    <Parameter Name="OPERATING_INTRLCK" Section="Input" Type="Bool" />
    <Parameter Name="OPERATING_INTRLCK_REV" Section="Input" Type="Bool" />
    <Parameter Name="START_INTRLCK" Section="Input" Type="Bool" />
    <Parameter Name="START_INTRLCK_REV" Section="Input" Type="Bool" />
    <Parameter Name="SLOW_INTRLCK" Section="Input" Type="Bool" />
    <Parameter Name="SLOW_INTRLCK_REV" Section="Input" Type="Bool" />
    <Parameter Name="in_READY" Section="Input" Type="Bool" />
    <Parameter Name="in_OVERLOAD" Section="Input" Type="Bool" />
    <Parameter Name="in_FEEDBACK" Section="Input" Type="Bool" />
    <Parameter Name="in_FEEDBACK_REV" Section="Input" Type="Bool" />
    <Parameter Name="in_LOTO_LOCK" Section="Input" Type="Bool" />
    <Parameter Name="in_MOTOR_TEMP" Section="Input" Type="Bool" />
    <Parameter Name="in_BRAKE_PROT" Section="Input" Type="Bool" />
    <Parameter Name="in_HW_SPEED_MON" Section="Input" Type="Bool" />
    <Parameter Name="in_SW_SPEED_MON" Section="Input" Type="Bool" />
    <Parameter Name="in_SIMO_PROT" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_START" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_START_REV" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_SLOW" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_STOP" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_SP_PLUS" Section="Input" Type="Bool" />
    <Parameter Name="in_LOCAL_SP_MINUS" Section="Input" Type="Bool" />
    <Parameter Name="in_VFD_ERROR" Section="Input" Type="Bool" />
    <Parameter Name="in_VFD_COMMERROR" Section="Input" Type="Bool" />
    <Parameter Name="in_CONTACT_PROT" Section="Input" Type="Bool" />
```

Obrázek 26 Část XML souboru specifikující instanci funkčního bloku obsaženého v networku funkce

```
<Access Scope="GlobalVariable" UId="36">
  <Symbol>
    <Component Name="@in_READY" />
  </Symbol>
</Access>
<Access Scope="GlobalVariable" UId="37">
  <Symbol>
    <Component Name="@in_OVERLOAD" />
  </Symbol>
</Access>
<Access Scope="GlobalVariable" UId="38">
  <Symbol>
    <Component Name="@VFD_1" />
    <Component Name="RUNNING" />
  </Symbol>
</Access>
<Access Scope="GlobalVariable" UId="39">
  <Symbol>
    <Component Name="@VFD_1" />
    <Component Name="RUNNING_REV" />
  </Symbol>
</Access>
<Access Scope="GlobalVariable" UId="40">
  <Symbol>
    <Component Name="@in_LOTO_LOCK" />
  </Symbol>
</Access>
<Access Scope="GlobalVariable" UId="41">
  <Symbol>
    <Component Name="@in_MOTOR_TEMP" />
  </Symbol>
</Access>
```

Obrázek 27 Část XML souboru specifikující navázání zástupných symbolů tagů dle parametru UId

```
<Wire UId="114">
  <IdentCon UId="36" />
  <NameCon UId="74" Name="in_READY" />
</Wire>
<Wire UId="115">
  <IdentCon UId="37" />
  <NameCon UId="74" Name="in_OVERLOAD" />
</Wire>
<Wire UId="116">
  <IdentCon UId="38" />
  <NameCon UId="74" Name="in_FEEDBACK" />
</Wire>
<Wire UId="117">
  <IdentCon UId="39" />
  <NameCon UId="74" Name="in_FEEDBACK_REV" />
</Wire>
<Wire UId="118">
  <IdentCon UId="40" />
  <NameCon UId="74" Name="in_LOTO_LOCK" />
</Wire>
<Wire UId="119">
  <IdentCon UId="41" />
  <NameCon UId="74" Name="in_MOTOR_TEMP" />
</Wire>
```

Obrázek 28 Část XML souboru specifikující navázání tagů na konkrétní vstupy funkčního bloku

3.2.2. Implementace a zpracování uživatelského vstupu

Tato kapitola se zabývá zadáváním a zpracováním uživatelského vstupu do prostředí Excel. Zmíníme podporované typy standardizovaných objektů a nezbytné kroky k definování jedné jejich instance. Poté popíšeme algoritmus, který tento vstup zpracuje a následně z něj vytvoří příkazy do příkazového souboru. Popisujeme zde větev procesu Excel z obrázku číslo 18 uvedeného v kapitole 3.1.

3.2.2.1. Formát vyplnění uživatelského vstupu

V této části práce se zaměříme na formát uživatelského vstupu používaného v projektu, kde je hlavním nástrojem aplikace Microsoft Excel. Tato aplikace je strukturována do několika listů, z nichž jeden slouží jako úvodní strana a ostatní, pro uživatele viditelné, jsou pojmenovány podle typu standardizovaného objektu, který podporují. Seznam listů je uveden v tabulce 3.

Tabulka 3 Listy specifikované v Excelu pro uživatelský vstup

Název Excel listu	Project	TAG_List	S0_Motors	S1_Valves	S2_Analogs
	S4_Digitals	S7_Dampers	S8_Pistons	S10_Groups	S12_Selections

Každý list je specializován na jeden typ standardizovaného objektu. Každý řádek na listu potom definuje jednu instanci daného objektu s jeho specifickými parametry a připojenými signály. Tyto parametry jsou rozděleny do sloupců, které odpovídají identifikátorům z knihovních souborů.

Pro názornost se podíváme na detailní rozdělení listu pro objekt motoru, který je z hlediska množství parametrů jeden z nejkomplexnějších typů podporovaných objektů. Z důvodu lepší přehlednosti jsou parametry rozčleněny do několika logických skupin, které budeme postupně probírat. Zmíněný Excelový list s tabulkou zde není možné vložit ve vizuální formě, avšak jeho úryvek je na obrázku číslo 29.

	A	B	BN	BO	BP	BQ	BR
1	**Object**		**Parameters**				
2	**Name**	**DB number**	**Slow_Speed**	**Setpoint_Local_Increment**	**UnitCode**		**FeedbackDelay**
3	16M3	10001	20.95	5.0	1342		T#20s
4	17M4	10002	20.95	5.0	1342		T#20s
5	18M1	10003	20.95	5.0	1342		T#20s
6	19M3	10004	20.95	5.0	1342		T#20s
7	19M4	10005	20.95	5.0	1342		T#20s
8	20M1	10006	20.95	5.0	1342		T#20s
9	21M1	10007	20.95	5.0	1342		T#20s
10	21M2	10008	20.95	5.0	1342		T#20s

Obrázek 29 Příklad vyplnění části tabulky uživatelského vstupu pro instance objektu motoru

Identifikátor instance objektu

První skupina parametrů je určena pro identifikaci instance objektu a určuje jeho unikátní název, číslo datového bloku a komentáře, které jsou využity jak pro popis kódu, tak i na

operátorských vizualizačních obrazovkách v různých jazycích[1]. Jejich výčet a příklad vyplnění je uveden v tabulce číslo 4.

Tabulka 4 Příklad vyplnění identifikačních údajů objektu motoru

Vstupní parametr objektu motoru	Příklad vyplnění
Name	0904MA01
DB Number	10003
Comment EN	Filter fan
Comment CZ	Ventilátor filtru

Parametry technologické skupiny

Další sekce specifikuje, do jaké technologické skupiny objekt patří, a tedy i do jaké adresářové větve programových bloků projektu se následně vygeneruje. Jedná se primárně o určení vytvářené adresářové struktury. Toto členění usnadňuje orientaci v projektu a zajišťuje, že jsou instance objektů přehledně tříděné podle příslušných technologických sekcí. Výčet je k vidění v tabulce 5.

Tabulka 5 Příklad vyplnění údajů skupiny objektu motoru

Vstupní parametr	Příklad vyplnění	Vysvětlení
Group	Zone 0 (Common)	Určuje cílový adresář generování struktur
Grp short	Zone0	Přípona názvu funkce pro její unikátnost
Grp No	0	Údaj pro navázání na část externího DB
Index in Grp	1	Index v poli navázané části externího DB

Parametry frekvenčních měničů

Další skupinou jsou informace se specifikací frekvenčních měničů. Projekt aktuálně podporuje dva typy frekvenčních měničů, ale seznam lze kdykoliv jednoduše rozšířit. Pokud instance nevyužívá frekvenční měnič, je VFD_Type nastaven na 0. Tato skupina parametrů je uvedena v tabulce 6.

Tabulka 6 Příklad vyplnění údajů o frekvenčním měniči objektu motoru

Vstupní parametr	Příklad vyplnění	Vysvětlení
VFD_Type	1	Objekt se vytvoří s připojeným VFD typu G120
VFD_DB	20003	Číslo datového bloku pro tvořené VFD

Parametry napojení vstupních signálů

Následující logická skupina se věnuje napojení vstupních signálů na objekt při jeho volání. Většina těchto parametrů zůstane při definování instance nevyplněná, jelikož slouží zejména k zajištění univerzálnosti standardizovaného objektu motoru. Uživatel vyplní pouze požadované signály a zbytek nechá prázdný. Nevyužité signály se následně nahradí identifikátorem nevyužitého signálu, který po importu stačí v projektu odstranit pomocí funkce najít a nahradit. V tabulce 7 uvádíme pouze výčet těchto parametrů, bez příkladů a dodatečných vysvětlivek .

[1] Zde se počítá s rozšířením podpory na až 3 dodatečné jazyky dle výběru uživatele u budoucí verze projektu

Tabulka 7 Výčet možných vstupních signálů objektu motoru

SKUPINA SIGNÁLŮ NA VSTUPU	In_READY	In_OVERLOAD	In_FEEDBACK	In_FEEDBACK_REV
	In_LOTO_LOCK	In_MOTOR_TEMP	In_BRAKE_PROT	In_HW_SPEED_MON
	In_SW_SPEED_MON	In_SIMO_PROT	In_LOCAL	In_LOCAL_START
	In_LOCAL_START_REV	In_LOVAL_SLOW	In_LOCAL_STOP	In_LOCAL_SP_PLUS
	In_LOCAL_SP_MINUS	In_REMOTE	IN_CONTACT_PROT	In_EXCITATION
	OS_CONTROL	OP_CONTROL		

Parametry napojení výstupních signálů

Pro skupinu napojení výstupních signálů objektu platí stejné informace jako pro skupinu napojení vstupních signálů. Níže v tabulce 8 je opět jejich stručný výčet

Tabulka 8 Výčet možných výstupních signálů objektu motoru

SKUPINA SIGNÁLŮ NA VÝSTUPU	Out_ON	Out_ON_REV	Out_SLOW
	Out_LAMP	Out_LAMP_REV	Out_LAMP_Local
	Out_LAMP_WARNING	Out_HORN_WARNING	Out_EXCITATION

Parametry nastavení objektu

Další skupina parametrů se týká nastavení objektu. Jedná se o část inicializační funkce, která pomocí příkazů S (SET) a R (RESET) specifikuje, jaké funkce vytvářený objekt podporuje a které vstupní signály ve výsledku reálně čte.

Ke snížení konfiguračních parametrů je zde použit předpoklad, že pokud uživatel na instanci naváže některý ze vstupních signálů, tak jde o žádaný parametr a automaticky se tak nakonfiguruje i v rámci inicializační funkce. V této skupině tedy nejsou uvedeny všechny parametry, které objekt reálně využívá. Další funkcionalitou je, že uživatel může nechat buňky prázdné, pokud si přeje využít výchozí hodnoty. Výčet možných parametrů nastavení (bez využití vstupních signálů) je uveden v tabulce 9.

Tabulka 9 Výčet možných vnitřních nastavení objektu motoru (S/R)

SKUPINA NASTAVENÍ OBJEKTU	Use_Slow	Use_Reverse	Use_Horn
	Mode_Change_intern	SP_Visible	VFD_Data
	Value_Visible	Reset_After_Fault	LocalCMD_inManual
	Manual_inLocal	Stop_InManual	Flow_InManual

Vnitřní numerické parametry objektu

Další logickou skupinu tvoří vnitřní numerické parametry objektů, které jsou opět vztaženy k jeho inicializační funkci. Parametr UnitCode určuje klíč jednotky z textového listu, která se u objektu zobrazí na vizualizační obrazovce při jeho chodu. Tabulka s výčtem je v tabulce 10.

Tabulka 10 Výčet možných vnitřních nastavení parametrů objektu motoru (numerické)

SKUPINA SIGNÁLŮ NA VÝSTUPU	Setpoint_Low	Setpoint_High	Slow_Speed
	Setpoint_Local_Increment	UnitCode	

Časové konstanty objektu

Poslední skupinou v tabulce uživatelského vstupu je sekce časových konstant objektu (např. ve tvaru T#10s). Jedná se opět o vnitřní parametry objektu napojené v jeho inicializační funkci. Výčet možných parametrů je uveden v tabulce 11.

Tabulka 11 Výčet možných vnitřních nastavení časových konstant objektu motoru

SKUPINA ČASOVÝCH KONSTANT OBJEKTU	FeedbackDelay	FeedbackStopDelay	OnDelay
	OffDelay	Excitation_Timeout	ChangeDirDelay
	SpeedMonDelay	SpeedMonTact	FlowDelay
	Horn_Startup_Time	Waiting_Startup_Time	

3.2.2.2. Programové zpracování uživatelského vstupu

Při konfiguraci standardizovaných objektů je potřeba specifikovat velké množství parametrů, které se mezi typy objektů liší nejen názvy, ale i pozicemi ve sloupcích tabulky. Vyčítání dat z absolutních pozic by bylo nepraktické a neudržitelné, zejména pokud by došlo k častým nebo větším změnám ve struktuře dat.

Abychom tento problém vyřešili, implementovali jsme systém pro zpracování vstupních dat pomocí samostatného konfiguračního listu v Excelu, který je skryt před uživatelem. Tento list definuje operace a mapování dat pro každý typ objektu individuálně, což zahrnuje dynamické načítání identifikátorů a využití zástupných symbolů..

Specifika využití dynamického načítání identifikátorů:

- **Specifikace zpracování**: Pro každý typ objektu existuje definice v konfiguračním listu, která určuje, jak se mají data zpracovat.
- **Kolekce názvů sloupců**: Po načtení konfigurace se vytvoří kolekce názvů sloupců, k nimž se přistupuje pomocí zástupných symbolů, což umožňuje flexibilitu v případě změn ve struktuře tabulky.
- **Dynamický přístup k datům**: Místo odkazů na pevné pozice sloupců parametrů, systém využívá zástupné symboly ve formátu „{NÁZEV SLOUPCE}“ pro dynamický přístup k datům podle názvu parametru.
- **Výhoda**: Flexibilita tohoto přístupu umožňuje snadnou adaptaci na změny bez potřeby znovu konfigurovat absolutní pozice parametrů.

- **Nevýhoda**: Každá změna názvu sloupce vyžaduje aktualizaci všech odpovídajících zástupných symbolů v konfiguračním listu.

Tento přístup je základem pro efektivní a udržitelné zpracování uživatelského vstupu, což je klíčové pro automatickou generaci instancí objektů v našem systému. Jedná se navíc o přístup opakovaně využívaný napříč projektem a v dokumentu bude ještě několikrát zmíněn.

Formát konfiguračního listu

Konfigurační list slouží ke specifikaci pravidel zpracování uživatelského vstupu a tvorby příkazů do výstupní souboru ‚Make'. Je členěn na sekce pomocí struktury odsazovaní ve sloupcích, kde první čtyři sloupce hierarchicky specifikují typ objektu, jeho vztažený knihovní soubor a konkrétní Network, kterých se následující instrukce týkají. Příklad specifikace je uveden na následujících dvou obrázcích 30 a 31. Ty vznikly rozdělením stejných řádků konfiguračního listu na dva obrázky a v nerozdělené podobě je lze nalézt v dodatku A.

	A	B	C	D	E
3	**List**	**Destination file**			Kind
4			**Source**	Destination	Cmd
59	**S0_Motors**				
60		{@Dst}{Group}\S_Objects\Data Blocks\S0_MOTOR\{Name}.xml			DB
61		{@Dst}{Group}\S_Objects\Data Blocks\VFD_DB\{Name}_VFD.xml			DB
62		{@Dst}{Group}\S_Objects\Calls\S0_MOTOR_{Grp short}.xml			LAD
63			Nw_AUTO_ON	{Name}_AUTO_ON	
64					TXT
65					TXT
73			Nw_CALL	{Name}_CALL	
74					TXT
75					TXT
76					TXT
77					TXT
78					ALL
79					ALL
83					ALL
84					ALL
85					ALL
86					ALL
87					ALL
88					ALL

Obrázek 30 Příklad specifikace konfiguračního listu pro zahájení zpracování vstupních dat listu S0_Motors

E	F	G	H	I	J
Kind	Block No	Source file			
Cmd	Condition			TIA Symbol	New text
DB	{DB number}				
DB	{VFD_DB}				
LAD	270	{@Src}S_Objects\Calls\@S0_MOTOR_CALL.xml			
TXT				@Nw_Title_AUTO_ON_EN	{Name} - {Comment EN} (AUTO_ON)
TXT				@Nw_Title_AUTO_ON_CZ	{Name} - {Comment CZ} (AUTO_ON)
TXT				@Nw_Title_CALL_EN	{Name} - {Comment EN} (CALL)
TXT				@Nw_Title_CALL_CZ	{Name} - {Comment CZ} (CALL)
TXT				@Nw_Comment_CZ	Automaticky generované volání
TXT				@Nw_Comment_EN	Auto generated call
ALL				Alarms_S0_Motor.@Motor_ZONE[65432]	Alarms_S0_Motor.Motor_ZONE{Grp No}{{Inc
ALL				Counters_S0_Motor.@Counter_ZONE[654	Counters_S0_Motor.Counter_ZONE{Grp No}{
ALL	{in_READY}	<>		@in_READY	{in_READY}
ALL	{in_READY}	=		@in_READY	c_UNUSED
ALL	{in_OVERLOAD}	<>		@in_OVERLOAD	{in_OVERLOAD}
ALL	{in_OVERLOAD}	=		@in_OVERLOAD	c_UNUSED
ALL	{in_FEEDBACK}	<>		@in_FEEDBACK	{in_FEEDBACK}
ALL	{in_FEEDBACK}	=		@in_FEEDBACK	c_UNUSED

Obrázek 31 Příklad příkazů navazující na obrázek číslo 30

Vysvětlivky k položkám na obrázcích jsou uvedeny v tabulce číslo 12:

Tabulka 12 Popis funkce sloupců v konfiguračním souboru zpracování uživatelského vstupu

Sloupec	Funkce
A	Typ generovaného objektu a zároveň název listu pro čtení vstupních dat
B	Oddělení sekcí cyklicky aplikovaných na vyplněné instance objektů; zároveň udává cestu, do které se objekty vytvoří v cílovém adresáři
C	Popis Networku (SId) knihovního souboru, ze kterého se vychází
D	Popis Networku (SId), na který se přepíše generovaný objekt
E	Identifikátor příkazu/aktuální prováděné operace
F	Propsaná data aktuálního objektu, pro která se operace vyhodnocuje (levá strana podmínky)
G	Srovnávací podmínka
H	Pravá strana podmínky (na obrázcích odpovídá prázdnému textu„“)
I	Identifikátor ze vzorových symbolů, který se má nahradit při splnění podmínky
J	Data, kterými se nahradí identifikátor při splnění podmínky

Styl zpracování této šablony bude záhy popsán, ale prozatím je důležitá informace, že po detekování textu ve sloupci ‚B‘ načte algoritmus všechny řádky až do dalšího řádku s vyplněnou položkou v tomto sloupci a cyklicky aplikuje všechny jejich příkazy na každou instanci generovaného typu objektu. Z řádku, kde je sloupec ‚B‘ vyplněn vyčte informace uvedené v tabulce 13.

Tabulka 13 Specifikace podkladů ke generování objektu ze sloupců konfiguračního souboru

Sloupec	Funkce
B	Cesta, do které se objekty vygenerují v cílovém adresáři; objekty se umístí do individuálních adresářů podle dynamicky propsané skupiny přiřazené objektu na uživatelském vstupu
E	Typ generovaného kódu. DB, LAD nebo STL.
F	Číslo přiřazené generovanému objektu. Buď dynamicky propisuje číslo datového bloku z uživatelského vstupu, nebo přiřadí číslo tvořené funkci a inkrementuje ho dle čísla skupiny (kvůli předcházení duplicitám)
G	Cesta ke knihovnímu souboru, ze kterého se mají kopírovat Networky do nově vytvářených objektů
M	Cesta ke knihovnímu souboru, ze kterého má nově vytvářený soubor vycházet. Kopíruje se jako celek a využívá se zde buď předloha datového bloku, nebo předloha prázdné funkce volání.

Funkce sloupce ‚E‘ je lehce odlišná v závislosti na vyplnění sloupce ‚B‘. Pokud jsou vyplněné oba, udává jazyk generovaného kódu, tedy zda se jedná o DB, LAD, nebo STL. Na základě toho se načte informace, na které knihovní soubory má aplikace SIBLOCK Generatoru při dalším zpracování aplikovat příkazy z nyní tvořeného příkazového souboru. Pokud je sloupec 'B' prázdný, definuje sloupec 'E' příkazy vypsané v tabulce 14.

Tabulka 14 Výčet možných příkazu v konfiguračním souboru

Příkaz	Funkce
TXT	Bezpodmínečné nahrazení textu v generovaném objektu (komentář/titul)
ALL	Podmíněné nahrazení textu/signálu za data z uživatelského vstupu
DEL	Podmíněné odstranění řádku z generovaného objektu (používá se zejména pro inicializační funkce)
VAL	Podmíněné nahrazení zástupného symbolu specificky za reálná čísla

Podmínky srovnávání jsou stanoveny ve sloupci ‚G' a jejich výčet je uveden v tabulce 15.

Tabulka 15 Výčet možných podmínek v konfiguračním souboru

Podmínka	Funkce
=	Porovnání zda jsou dva texty identické (není case sensitive)
<>	Porovnání zda jsou dva texty odlišné (není case sensitive)
!=	Negovaná rovnost. Stejná funkce jako ‚<>', ale využívá se pro hodnoty
S	Porovnání zda načtený uživatelský vstup patří do {„S",“SET",“1"}
R	Porovnání zda načtený uživatelský vstup patří do {„R",“RESET",“0"}, nebo je nevyplněn
RI	Porovnání zda načtený uživatelský vstup patří do {„R",“RESET",“0"}
SI	Porovnání zda načtený uživatelský vstup patří do {„S",“SET",“1"}, nebo je nevyplněn
<	Porovnání zda je levá strana menší než pravá (nevyužívá se)
>	Porovnání zda je levá strana větší než pravá (nevyužívá se)
<=	Porovnání zda je levá strana menší nebo rovna pravé (nevyužívá se)
>=	Porovnání zda je levá strana větší nebo rovna pravé (nevyužívá se)

Tvorba dat do příkazového souboru

Pro každý generovaný typ standardizovaného objektu se specifikují knihovní soubory, které se použijí k pozdější tvorbě výsledných XML souborů, a určí se místo pro ukládání těchto výsledků. Při zpracování dat z uživatelského vstupu se potě vyhodnocuje sada podmínek, která následně určí, jaký text se ve vygenerované instanci použije k nahrazení zástupných symbolů. Tyto údaje jsou uvedeny ve výstupním souboru pod názvem ‚Make'.

Je třeba zdůraznit, že k nahrazování textu ani k využití knihovních souborů nedochází v kroku zpracování uživatelského vstupu. Zde se pouze vytvoří předpis pro tyto operace, které se následně provedou až v aplikaci SIBLOCK Generatoru, kde se pak vytvoří výsledné XML soubory.

Tento předpis, nebo lépe řečeno příkazy, mají jasně daný formát. Než ho vysvětlíme, je na obrázku číslo 32 k vidění část výsledného příkazového ‚Make' souboru. Jsou zde vidět části příkazů ke generování instancí standardizovaných objektů Digitálu, kde se na prvním řádku specifikují cílové adresáře a poté každý řádek definuje individuální příkaz ke generování.

Obrázek 32 Část příkazového souboru ‚Make' vytvořeného na základě zpracování uživatelského vstupu

Vzhledem k velikosti tohoto souboru není možné jednoduše vizualizovat všechny jeho části, a proto provedeme výčet složek hlavičky příkazů v tabulce 16 i s jejich vysvětlením. Tato hlavička identifikuje každý příkaz a v některých případech je i sama o sobě dostačující. Každá část příkazu je vždy oddělena symbolem středníku.

Tabulka 16 Popsaná struktura hlavičky příkazu zpracovaného z uživatelského vstupu

Segment	Funkce
Typ příkazu	Identifikátor typu dat obsažených v příkazu (DIRS, FILE, MOD)
Typ souboru	Struktura kódu generovaného souboru (DB, LAD, STL)
Zdroj (Src)	Cesta knihovního XML souboru pro vyčítání sekcí dat
Sektor	Identifikátor k určení sekce (Node) XML souboru
Destinace	Cesta kam umístit vygenerované objekty
SubSektor	Identifikátor k určení podsekce (Node) XML souboru
Alternativní zdroj	Zdroj pro zkopírování celého XML souboru
Number	Číslo přiřazené generovanému souboru. Tedy číslo funkce nebo DB

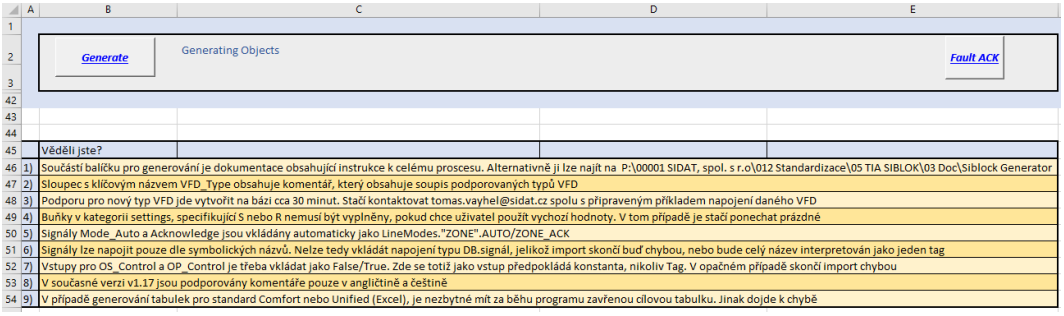
Po hlavičce následuje výčet jednotlivých příkazů k nahrazení textů a signálů označených zástupným symbolem za data určená podmíněným zpracováním z uživatelského vstupu. Jejich struktura je velmi jednoduchá, avšak opakující se pro každý individuální příkaz, což místy vede k velice dlouhým příkazům jakožto celku. Příklad struktury těchto příkazů je uveden v tabulce 17.

Tabulka 17 Popsaná struktura opakující se části příkazu specifikující konkrétní operace nahrazování textu

Segment	Funkce
Typ příkazu	Typ aktuálního příkazu daný konfiguračním souborem
Oddělovač	Následuje oddělení pomocí symbolů „|||" (jedná se o nevyužité segmenty)
Identifikátor	Identifikátor zástupného textu z knihovních souborů
Oddělovač	Oddělení symbolem „|"
Text k nahrazení	Text získaný na základě uživatelského vstupu, který má nahradit identifikátor textu ve výsledném souboru

Projektový list a spuštění zpracování uživatelského vstupu

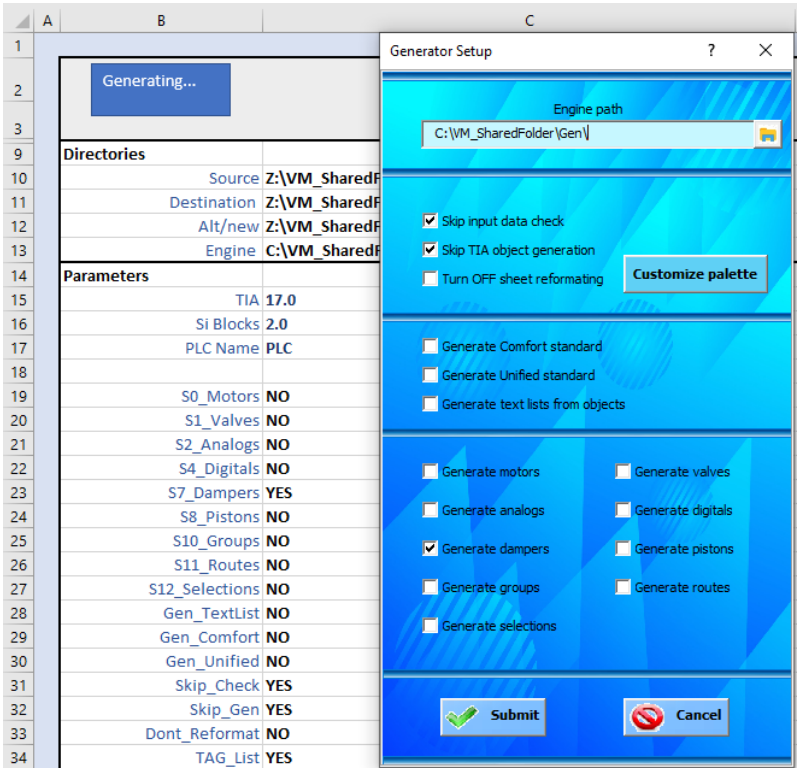
Součástí aplikace Excel sloužící k zadávání a zpracování uživatelského vstupu je úvodní list s názvem ‚Project'. Viz obrázek číslo 33.



Obrázek 33 Náhled úvodního listu Excelu s tlačítkem k zahájení zpracování uživatelského vstupu

Na tomto listu může uživatel stisknutím tlačítka 'Generate' otevřít konfigurační okno, pomocí kterého si navolí požadované funkcionality a zahájí generování. Aby uživatel nemusel při každém individuálním generování opakovat všechny volby, jsou na listu skryté konfigurační parametry, ze kterých se při spuštění okno inicializuje.

Jakmile je generování schváleno uživatel, propíše okno aktualizované parametry zpátky do konfigurace, aby byly uchovány pro budoucí použití. Na obrázku 34 lze vidět tuto vnitřní paměť a vyskakovací konfigurační okno, které uživatel využívá.



Obrázek 34 Náhled konfiguračního okna zpracování uživatelského vstupu a skryté paměti parametrů

Popis algoritmu zpracování uživatelského vstupu

V této sekci popíšeme algoritmus, který zpracovává data z uživatelského vstupu a konfiguračního souboru a vytváří na jejich základě výstupní příkazový soubor s názvem ‚Make'. Tento soubor slouží v pozdějších krocích ke generování výsledných XML souborů, které po importu do projektu vytvoří požadované instance standardizovaných objektů.

Algoritmus funguje tak, že po spuštění nejprve načte konfigurační data zvolená uživatelem a uloží je do paměti. Následně prochází list ‚Generator' po řádcích, dokud nenarazí na klíčové slovo v prvním sloupci. Pokud se toto klíčové slovo shoduje s názvem některého z listů Excelu a zároveň je v paměti povoleno generování stejnojmenného standardizovaného objektu, začne algoritmus zpracovávat řádky náležící tomuto klíči. V opačném případě je přeskočí a hledá další klíčové slovo v prvním sloupci.

Pokud je generování daného typu objektu povoleno, projede algoritmus list dedikovaný tomuto objektu a načte si do objektu kolekce názvy všech sloupců a jejich odpovídající indexy. Tato informace bude využita později při nahrazování zástupných symbolů dynamickým doplňováním textu.

Poté začne algoritmus hledat v rámci dané sekce vyplněnou buňku ve druhém sloupci. Jakmile ji najde, načte celý tento řádek a vztáhne k němu všechny řádky až do doby než narazí na další vyplněnou položku buď v prvním, nebo druhém sloupci. Načtený řádek vztahující se k položce vyplněné ve druhém sloupci udává specifikaci, jaká část standardizovaného objektu se nyní tvoří (datový blok, funkce volání nebo funkce inicializace). Dále rovněž udává, jaké knihovní soubory má navazující program využívat ke tvorbě objektů a kam má výsledek uložit.

Sekci příkazů vztahující se k vyplněné položce ve druhém sloupci projíždí algoritmus opakovaně po řádcích tak, že na každou instanci generovaného objektu aktuálního typu daného předchozí specifikací z prvního sloupce, aplikuje všechny uvedené příkazy a operace.

Při zpracovávání příkazů se data opět dělí na další podskupiny na základě údaje ve třetím sloupci. Tento údaj určuje, který network funkcí má navazující aplikace zkopírovat ze knihovního XML souboru do generovaného souboru. Příkazy uvedené od pátého sloupce pak řeší nahrazování konkrétních částí textů a dat z onoho networku tak, aby na výstupu byl již požadovaný objekt.

Při vyčítání dat z konfiguračního souboru dochází rovnou ke zpracování a tvorbě příkazů, které se následně zapíší do příkazového souboru. Formát příkazů jsme již vysvětlili dříve, takže zde představíme jen logickou strukturu jejich tvorby. V podstatě se jedná o podmíněné zpracovávání dat a řetězení těchto údajů za sebe v jisté posloupnosti.

Popsaný proces zpracování uživatelského vstupu na základě šablony a následné tvorby příkazů do souboru ‚Make' je znázorněný pseudokódy, které popisují Algoritmus 3, Algoritmus 4 a Algoritmus 5.

Algoritmus 3: Načtení šablony dle typu objektu ke zpracování vstupu

```
    Input: Configuration selected by user, Generator sheet
    Output: Make file with processed user input.
1   FOR each row in generator sheet DO
2     IF genSheet.cell(row, 1) contains objectKeyword THEN //Col`A`
3       genPermitted ← FALSE
4       IF objectKeyword was selected by user THEN
5         dataCollection ← Load table headers of objectKeyword
6         genPermitted ← TRUE
7       End IF
8     Else IF genSheet.cell(row, 2) NOT Empty AND genPermitted THEN
9       ProcessCMDS(genSheet, objectKeyword, dataCollection, row)
10    End IF
11  NEXT row
12  END
```

Algoritmus 4: Funkce **ProcessCMDS**

Input: genSheet, objectKeyword, dataCollection, genRow

Output: Processed user input written in Make file as commands

```
1  segmentSize ← number of rows until next element in column A or B
2  Segment ← genSheet rows from genRow to genRow + segmentSize
3  FOR each objectRow in objectKeyword sheet DO
4      FOR each segRow in Segment DO
5          IF segRow (2) is not Empty THEN //Column ‚B‘
6              Create and write FILE cmd from loaded row
7          ELSE
8              IF segRow (3) is not Empty THEN //Column ‚C‘
9                  IF modCMD is not empty THEN
10                     WRITE modCMD to Make file
11                     CLEAR modCMD
12                 END IF
13                 modCMD ← Create new header for MOD cmd
14             ELSE
15                 tmpCMD ← EvalCMD(objectRow, dataCollection, segRow)
16                 modCMD ← modCMD appended by tmpCMD
17             END IF
18         END IF
19     NEXT segRow
20 NEXT objectRow
21 Save updated Make file and RETURN
```

Algoritmus 5: Funkce **EvalCMD**

```
    Input: objectRow, dataCollection, segmentRow
    Output: Processed part of MOD cmd related to larger structure
1   cmdKind ← segmentRow(5)  //Column ‚E'
2   condition ← segmentRow(7) //Column ‚G'
3   tmpCmd ← „"
4   IF cmdKind = „DEL" THEN
5   |  cmdCell ← segmentRow(6) //Column ‚F'
6   |  cmdData ← getInputData(cmdCell, dataCollection, objectRow)
7   |  IF evaluateSRCond(condition, cmdData) THEN
8   |  |  rowTarget ← segmentRow(11) //Column ‚K'
9   |  |  tmpCmd ← constructDelCMD(rowTarget)
10  |  END IF
11  ELSE
12  |  leftCell ← segmentRow(6) //Column ‚F'
13  |  leftData ← getInputData(leftCell, dataCollection, objectRow)
14  |  rightCell ← segmentRow(8) //Column ‚H'
15  |  rightData ← getInputData(rightCell, dataCollection, objectRow)
16  |  IF evaluateCond(condition, leftData, rightData) THEN
17  |  |  replacedSym ← segmentRow(9) //Column ‚I'
18  |  |  newSym ← segmentRow(10) //Column ‚J'
19  |  |  tmpCmd ← constructNewCMD(cmdKind, replacedSym, newSym)
20  |  END IF
21  END IF
22  RETURN tmpCmd
```

3.2.3. Implementace generátoru objektů

V této kapitole popíšeme způsob implementace funkcí aplikace generátoru objektů, které jsou potřebné v procesu tvorby a importu XML souborů. Na závěr kapitoly se zmíníme i o použitém řešení kompatibility použité aplikace a rozhraní TIA Portal Openness s různými verzemi prostředí TIA Portal.

3.2.3.1. Tvorba XML souborů dle příkazového souboru

V předchozích krocích jsme ukázali, jak se zadává a zpracovává uživatelský vstup v prostředí Excel. Zmíněný proces vytvoří příkazový soubor ‚Make', který v této části převedeme na XML soubory připravené k importu do projektu v prostředí TIA Portal.

Tento převod, nebo lépe řečeno zpracování příkazového souboru, probíhá pomocí aplikace napsané v jazyce C#, nazvané SIBLOCK Generátor. Tato aplikace funguje zcela nezávisle na tabulce uživatelského vstupu v prostředí Excel. Jediným společným prvkem je zmíněný příkazový soubor ‚Make'.

Aplikace tedy může běžet i na zcela jiném zařízení, než na kterém byl vyplněn uživatelský vstup. Stačí, když se jí poskytne přístup k souboru ‚Make' ať už například pomocí sdíleného adresáře, nebo jeho přetažením.

Vyčítání dat z příkazového souboru

V předchozích kapitolách jsme popsali formát, ve kterém jsou ukládány příkazy zpracované z uživatelského vstupu do souboru ‚Make'. Zmínili jsme, že jde o formát podobný CSV, který využívá středník jako oddělovač segmentů hlavičky příkazu. Za posledním středníkem se nachází hlavní část příkazu, která specifikuje jednotlivé operace nahrazení nebo smazání textu. V této části se jako oddělovač využívá symbol „|". Každý příkaz je ve zmíněném souboru definován právě na jednom řádku.

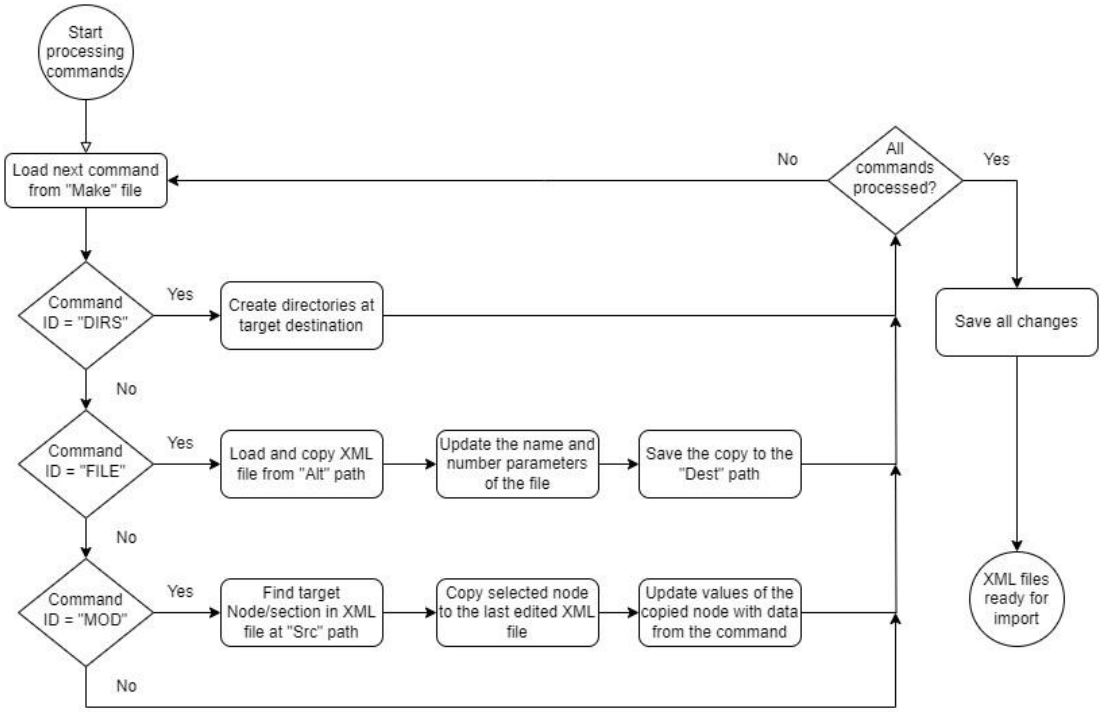
Tyto skutečnosti využijeme při vyčítání dat z příkazového souboru. Pomocí funkcí jazyka C# vyčteme data ze souboru po řádcích a rozdělíme je na části definované středníkem. Tímto způsobem získáme identifikátor příkazu, který uložíme do objektu instance třídy, dle struktury popsané v části kapitoly 3.2.2.2 Tvorba dat do příkazového souboru. K tomuto objektu nadále přiřadíme všechny operace nahrazení nebo smazání textu.

Tvorba XML souborů

Po vyčtení dat z příkazového souboru a jejich organizaci do předpřipravených struktur program pokračuje krokem zpracování těchto dat. Během této operace vyčítá jednotlivé příkazy a dle jejich identifikátoru přizpůsobuje svou funkcionalitu.

Nejdůležitějšími kroky jsou příkazy s identifikátorem „FILE", které po zpracování vytvoří nový XML soubor v cílovém adresáři kopií již existující předlohy. U tohoto nového souboru se následně upraví identifikátor programového bloku TIA Portal ve formě jeho názvu a čísla bloku. Na příkaz „FILE" může navazovat série příkazů s identifikátorem „MOD", které rozšiřují vytvořený XML soubor o další sekce (nodes) zkopírované z knihovního XML souboru. Obsah zkopírovaných sekcí se následně modifikuje o data z uživatelského vstupu, která jsou specifikována v rámci zpracovávaného příkazu.

Princip zpracování příkazů je graficky znázorněn na obrázku 35. Zjednodušeně řečeno, popsaná funkce pouze kopíruje data z XML souborů sloužících jako knihovní předlohy a následně vyhledá a modifikuje text, který je ve zkopírovaných částech obsažený.



Obrázek 35 Vývojový diagram procesu tvorby XML souborů na základě příkazů ze souboru ‚Make'

V případě jazyka LAD provádíme pouze operace dosazení textu za zástupné symboly, což vede k situaci, kdy i v případě nevyužitého signálu musíme kvůli dodržení formátu doplnit nějaký text. Pro tento krok se využívá text „C_Unused", který značí nevyužité signály napojené na instanci funkčního bloku. Tohoto zástupného textu se v projektu pak lze hromadně zbavit pomocí funkce najít a nahradit.

Tohoto zástupného textu se dá zbavit lepší manipulací s XML soubory, kdy by se kromě nahrazování textu odstraňovaly i konkrétní segmenty XML souboru. Tento aspekt práce zatím nebyl opraven, protože nebyl prioritou, ale je v plánu ho v budoucích verzích předělat.

Výsledek zpracování

Vytvořené XML soubory definují programové bloky kódu, které reprezentují instance standardizovaných objektů definované na základě uživatelského vstupu z Excelu. Pro lepší organizaci a přehlednost projektu, do kterého budeme tyto soubory importovat, jsou vytvořené soubory rozděleny v rámci předdefinované adresářové hierarchie. Tato struktura se pak při procesu importu přes rozhraní TIA Portal Openness překlopí i do prostředí TIA Portal.

Zmíněná struktura je znázorněná pomocí tabulky číslo 18, kde míra odsazení značí hloubku zanoření v adresářové struktuře. Názvy adresářů jsou znázorněné tučně a text uvedený mezi symboly složených závorek „{}“ značí dynamicky doplňovaný text dle údajů daného objektu.

Tabulka 18 Adresářová hierarchie vytvořených XML souborů

PLC
Program blocks
{Group_NAME}
S_Objects
Calls
S1_VALVES_CALL_{Group_SHORT}.xml
Data_Blocks
S1_VALVE
{NAME}.xml
Init
S1_VALVES_INIT_{Group_SHORT}.xml

3.2.3.2. Importování souborů XML formátu do TIA Portalu

Tato kapitola popisuje etapu práce zaměřenou na importování vytvořených XML souborů, které specifikují instance standardizovaných objektů a všech dalších náležitostí do cílového projektu. K tomu využijeme funkci implementovanou do naší aplikace SIBLOCK Generator, která pomocí rozhraní TIA Portal Openness importuje všechny vygenerované XML soubory do projektu.

Funkci importu rozdělíme na dva kroky: import vygenerovaných XML souborů z adresáře ‚Dest‘ a import statických souborů z adresáře ‚Standard‘. Z implementačního hlediska se jedná o stejnou funkci s různým vstupním argumentem složky, ze které má program čerpat. Z uživatelského hlediska jsou k dispozici dvě samostatná tlačítka.

Pro správný průběh importu je nutné dodržet pořadí operací, aby například datový blok využívající UDT nebyl importován před tím, než se v projektu nachází příslušné UDT. I když při importu nedojde k chybě, nekonzistence se automaticky nevyřeší po dodatečném doplnění UDT do projektu a uživatel by musel tyto nekonzistence ručně aktualizovat.

Technologický postup operací při importu je následující: program nejprve zkontroluje, zda jsou v projektu všechny potřebné jazyky, a případně je automaticky nastaví. Poté projde poskytnutý adresář a pokusí se nalézt složku s názvem ‚UserTypes‘. Pokud ji najde, vytvoří její podadresáře jako grupy (složky) v sekci ‚UserTypes‘ připojeného projektu. Následně začne importovat všechny uživatelské datové typy definované XML soubory z poskytnutého adresáře tak, aby byla v projektu stejná hierarchie jako v daném adresáři. Stejný postup se opakuje i pro ‚ProgramBlocks‘, což znamená sekci programových bloků v cílovém projektu.

Stejně jako v případě implementované funkce pro export programových bloků (kapitola 3.2.1), musíme i zde projít objektovým modelem TIA Portal Openness. Drobným rozdílem však je, že v případě importu chceme cílit pouze na jedno zařízení v projektu, které je dané jako uživatelský vstup. Tato iterace je popsána pseudokódem Algoritmu 6.

Algoritmus 6: Procedura Importu XML souborů do TIA Portal projektu

```
    Input: source directory path: srcDir, targetDevice
    Output: Generated TIA Portal objects based on XML files
1   SET configured languages in connected project
2   FOR EACH Device device in ConnectedProject.Devices DO
3       FOR EACH DeviceItem deviceItem in device.DeviceItems
4           swTarget ← GetPlcSoftware(deviceItem)
5           IF swTarget is PlcSoftware AND device is targetDevice THEN
6               targetPath ← PathCombine(srcDir, swTarget.TypeGroup.Name)
7               CreateUdtGroups(swTarget.TypeGroup.Groups, targetPath)
8               ImportUserTypes(swTarget.TypeGroup, targetPath)
9               targetPath ← PathCombine(srcDir,swTarget.BlockGroup.Name)
10              CreateBlockGroups(swTarget.BlockGroup.Groups,targetPath)
11              ImportProgramBlocks(swTarget.BlockGroup, targetPath)
12          END IF
13      NEXT deviceItem
14  NEXT device
15  END
```

Uvedený pseudokód dále vyvolává zmíněné funkce na tvorbu adresářů v projektu a funkce k importu vytvořených XML souborů. Stejně jako funkce pro export programových bloků z projektu, jsou i tyto funkce rekurzivní, protože program dopředu nezná úroveň zanoření adresářů v cílové složce.

Princip operace všech těchto funkcí je velmi podobný, a proto popíšeme pouze funkci „ImportProgramBlocks“. Tato funkce předpokládá, že v cílovém projektu již existuje kompletní struktura grup (složek), která je identickým obrazem adresářové struktury na cestě, ze které importujeme XML soubory. Funkce pak jako vstupní argument obdrží objekt aktuální grupy programových bloků získaný pomocí TIA Portal Openness z připojeného projektu a cestu ke složce v počítači, ze které se objekty importují.

Algoritmus následně rozšíří cestu ke složce s XML soubory o název aktuální grupy programových bloků. Z této cesty pak vyčteme všechny soubory typu XML, zkontrolujeme, zda již v cílovém projektu nejsou obsaženy a případně se je pokusíme importovat přes rozhraní TIA Portal Openness. Uživatel si může zvolit typ chování programu v případě detekované kolize, kdy je generovaný soubor již přítomný v cílovém projektu.

Po dokončení iterace přes všechny XML soubory na aktuální cestě algoritmus pokračuje iterací přes všechny grupy obsažené v aktuální grupě programových bloků a rekurzivně se vyvolá znovu s aktualizovanými vstupními parametry. Pseudokód je popsaný jako Algoritmus 7.

```
Algoritmus 7: Rekurzivní funkce ImportProgramBlocks
    Input: PlcBlockGroup actualBlockGroup, string actualPath
    Output: Program code imported into TIA Portal project
 1  actualPath ← PathCombine(actualPath, actualBlockGroup.Name)
 2  dirInfo ← DirectoryInfo(actPath).GetFileSystemInfos()
 3  FOR FileSystemInfo fileInfo in dirInfo DO
 4  |  fileName ←  fileInfo.FullName
 5  |  Handle cases where fileName already exist in target Project
 6  |  actBlock ← actualBlockGroup.Blocks
 7  |  OpennessImport(actBlock, blockPath)
 8  NEXT actBlock
 9  FOR PlcBlockGroup actualGroup in actualBlockGroup.Groups DO
10  |  ImportProgramBlocks(actualGroup, actPath) //Recursive call
11  NEXT actualGroup
12  END
```

3.2.3.3. Podpora kompatibility rozhraní s verzemi TIA Portalu

Jak již bylo uvedeno v teoretické části, vlastnosti rozhraní TIA Portal Openness jsou stanoveny verzí knihovny Siemens.Engineering.dll. Tato knihovna určuje verzi rozhraní a tím i schopnost aplikace připojit se k určité verzi TIA Portalu. Je podporována zpětná kompatibilita, takže s verzí souboru V17 je aplikace schopná komunikovat i s projekty vytvořenými ve verzích V15, V15.1, V16 a V17 [6].

Hlavní otázkou této kapitoly je zjistit, jakým způsobem lze naši vyvinutou aplikaci případně modifikovat tak, aby byla kompatibilní i s budoucími verzemi TIA Portalu, které ještě neexistují. Jaké kroky je potřeba podniknout a jak reálná je tato ambice? Je nutné vždy zasahovat do kódu? Bude každá verze potřebovat svou vlastní aplikaci?

Po výzkumu se ukázalo, že existuje způsob, který teoreticky nevyžaduje každoroční zásahy do kódu s každou novou verzí TIA Portalu (a tím i TIA Openness). Jak však uvedeme později, toto je pouze částečná pravda.

Podle dokumentu [6] „TIA Portal Openness: Referencing the Siemens.Engineering.dlls and Assembly Resolve“ lze budoucí kompatibilitu zajistit pomocí tzv. Assembly Resolve, tedy způsobu řešení použitých/načtených verzí dll souborů. Principem je, že během vývoje používáme zástupný dll soubor, který při spuštění programu nahradíme aktuálním souborem detekovaným na počítači v rámci instalačních cest.

Dle zmíněného dokumentu můžeme použít buď NuGet Package s názvem „Siemens.Collaboration.Net.TiaPortal.Openness.Resolver“, který po nainstalování stačí pouze

inicializovat a potřebné linkování dll souborů řeší automaticky, nebo můžeme postupovat ručně vlastním kódem.

Dodatečné poznatky

Níže uvedené informace slouží zejména pro sdílení zkušeností, protože k tomuto tématu patrně neexistují prakticky žádné informace na fórech ani v dokumentacích k datu psaní této práce.

Na základě poznatků získaných již během vývoje je vhodné zmínit, že řešení pomocí nainstalovaného balíčku dle dokumentu [6] skutečně umožnilo připojení aplikace jak k verzi TIA v18, tak i k v17, na které probíhal vývoj. Problém však nastal v situaci, kdy bylo na zařízení nainstalováno více verzí prostředí TIA Portal. K otevřeným instancím prostředí se aplikace dokázala připojit vždy správně, ale když chtěl uživatel spustit instanci TIA Portalu přímo z aplikace (například kvůli možnosti spuštění bez UI), spustila se vždy pouze verze 17.

Aby se tomuto chování předešlo, byl zaveden postup detekce nainstalovaných verzí TIA Portal na zařízení z registrů, ze kterých si uživatel při spouštění aplikace jednu vybere. Myšlenkou bylo na základě této vybrané verze vnutit Assembly Resolveru z balíčku pouze jedinou verzi dll souborů. Tento přístup však nefungoval, dokud se neaktualizovala verze knihovny v projektu na alespoň stejnou verzi jako má TIA Portal, který chce uživatel spustit z aplikace. Pro tuto funkcionalitu je tedy nutné s každou novou podporovanou verzí TIA Portalu aktualizovat i aplikační soubory.

3.3. Generování standardizovaných struktur pro HMI

V této kapitole se odchýlíme od primárního tématu práce, kterým je automatické generování instancí standardizovaných objektů prostřednictvím rozhraní TIA Portal Openness. Zaměříme se místo toho na další metody generování datových struktur do projektu, konkrétně na tvorbu standardizovaných struktur pro operátorské panely, které zprostředkovávají rozhraní mez člověk a strojem (HMI).

Generování objektů pro HMI bylo iniciováno myšlenkou: „Pokud uživatelé zadávají data do tabulky, je možné z těchto vyplněných dat získat více informací?“ Odpověď je „Ano“. Prostředí TIA Portal totiž umožňuje export a import specifických datových struktur ve formátu „.xlsx“ (Excel). V prostředí Excelu lze na základě předlohy jednoduše rozkopírovat řádky reprezentující jednotlivé datové objekty a vyplnit několik údajů k vytvoření nové validní struktury. Všechna potřebná data jsou navíc již obsažena v tabulce uživatelského vstupu generátoru, což eliminuje potřebu dalších úprav.

Cílem této kapitoly je generovat datové struktury pro operátorské panely řady Comfort a Comfort Unified v platném formátu souboru xlsx, které se následně dají importovat do TIA Portalu. To zahrnuje:

- Vytváření HMI tagů, které mohou být interní nebo navázány na specifické PLC signály.
- Vytváření položek v textových listech dle standardu.
- Generování alarmových hlášení, která budou propojena s vytvořenými HMI tagy i textovými listy.

Na závěr této úvodní stojí za zmínku, že hlavním důvodem použití xlsx souborů k definování požadovaných datových struktur na operátorské panely je jednoduchost implementace. I když dokumentace naznačuje, že generování by mohlo být provedeno pomocí rozhraní TIA Portal Openness, předběžná analýza a návrh kódové struktury naznačují, že by se jednalo o komplexnější proces s vyšší pravděpodobností chyb. Museli bychom totiž i v tomto případě opakovat celý proces generování XML souborů.

V navazujících sekcích se tedy podíváme na to, jak jsou požadované datové struktury reprezentovány ve formátu souboru „.xlsx“ a poté vysvětlíme algoritmus automatického generování těchto souborů na základě dat uvedených na uživatelském vstupu.

3.3.1. Exportovaná struktura HMI Tagu

V této části se věnujeme popisu struktury HMI Tagu ve formátu souboru xlsx. Ukážeme konkrétní příklad pro HMI Tag, který se využívá k vizualizaci parametrů instance standardizovaného objektu motoru, a který je použitelný pro řadu operátorských panelů Comfort.

Struktura HMI tagu v souboru .xlsx se liší podle typu operátorského panelu, na kterém bude tento tag definován. Příklad definice HMI tagu v souboru .xlsx uvedeme pouze pro zmíněnou řadu panelů Comfort, jelikož mají kratší a přehlednější hlavičku specifikující jednotlivé parametry.

Tabulka 19: Příklad definice konkrétního HMI tagu na panely řady Comfort (záložka Hmi Tags)

Hlavička sloupce	Příklad vyplněných dat
Name	S0_MOTOR_DEF_007
Path	S_Object_Specific\S_Specific_0_Motor\S0_Motor_Definition
Connection	HMI_Connection_1
PLC tag	<No Value>
DataType	DInt
Length	4
Coding	Binary
Access Method	Absolute access
Address	%DB[DB_Motor_007].DBD136
Indirect addressing	False
Index tag	<No Value>
Start value	<No Value>
ID tag	0
Display name [en-US]	<No Value>
Comment [en-US]	<No Value>
Acquisition mode	Cyclic in operation
Acquisition cycle	1 s
Limit Upper 2 Type	None
Limit Upper 2	<No Value>
Limit Upper 1 Type	None
Limit Upper 1	<No Value>
Limit Lower 1 Type	None
Limit Lower 1	<No Value>
Limit Lower 2 Type	None
Limit Lower 2	<No Value>
Linear scaling	False
End value PLC	10
Start value PLC	0
End value HMI	100
Start value HMI	0
Gmp relevant	False
Confirmation Type	None
Mandatory Commenting	False

Parametry tagu získané z předlohy vyžadují pro vytvoření nového HMI tagu pouze minimální úpravy. Je dostačující změnit pouze údaje v polích Name, Address a Start value, abychom definovali nový unikátní tag, který je přizpůsobený konkrétním požadavkům projektu. Další informace o této struktuře a jejím praktickém využití v projektu budou podrobněji rozebrány v následujících sekcích při rozboru implementace algoritmů tvorby .xlsx souborů.

V podobném stylu jako HMI tag rozebraný v tabulce 19 získáme předlohu pro všechny typy podporovaných tagů exportem předpřipravených tagů, které jsou definovány dle vizualizačního standardu firmy. Na základě takto získaných předloh sestavíme šablonu, jejímž využitím se budeme zabývat v dalších částech této kapitoly.

3.3.2. Exportovaná struktura Textového listu

Tato část kapitoly je zaměřena na popis struktury Textového listu, který se využívá pro standardizované popisy objektů, jako jsou motory, ve formátu souboru .xlsx. Tato struktura je identická pro oba typy operátorských panelů řad Comfort a Comfort Unified.

Textový list slouží jako zdroj textových dat pro popisy a názvy, které jsou aplikovány na HMI elementy v projektu. Každý list obsahuje záznamy, které jsou přiřazeny k jednotlivým objektům a mají vícejazyčné varianty textů zobrazených na vizualizačním rozhraní.

- **TextList** - Definuje existenci textového listu. Každý list reprezentuje jednu skupinu textů, jako například popis (ve standardu „Description“) motorů. Příklad parametrů je uveden v tabulce číslo 20.
- **TextListEntry** - Definuje jednotlivé položky v TextListu. Každá položka odpovídá jednomu textovému popisu nebo názvu konkrétního objektu, který je použit v projektu. Příklad lze vidět v tabulce 21.

Tabulka 20 Příklad definice konkrétního HMI textového listu (záložka TextList)

Hlavička sloupce	Příklad vyplněných dat
Name	S0_MOTORS_DESCRIPTIONS
ListRange	Decimal
Comment [en-US]	<No value>
Comment [cs-CZ]	<No Value>

Tabulka 21 Příklad definice konkrétní položky HMI textového listu (záložka TextListEntry)

Hlavička sloupce	Příklad vyplněných dat
Name	Text_list_entry_2
Parent	S0_MOTORS_DESCRIPTIONS
DefaultEntry	<No value>
Value	10002
Text [en-US]	Inner conveyor DP01
Text [cs-CZ]	Vnitřní dopravník DP01
FieldInfos	

Pro vytvoření nového textového listu je nutné založit nový list v souboru, definovat jeho hlavičku a výchozí hodnoty. Přidání nových položek do listu vyžaduje inkrementaci vstupní položky Name a nastavení patřičné Value, která v rámci použitého standardu odpovídá datovému bloku instance objektu, ke kterému je text navázán. Texty jsou dále definovány podle požadovaných jazyků, což umožňuje flexibilní lokalizaci projektu.

3.3.3. Exportovaná struktura alarmového hlášení

Tato část popisuje, jak je struktura alarmu definována ve formátu souboru .xlsx, který se používá pro konfiguraci alarmových hlášení v aplikacích HMI. Struktura se liší v závislosti na řadě operátorských panelů, přičemž zde se zaměřujeme na formát používaný u řady Comfort.

Alarmová hlášení v HMI systémech slouží k upozornění operátorů na výjimečné nebo kritické stavy zařízení. Každý alarm má specifické parametry, které určují jeho chování a způsob, jakým je operátorovi prezentován.

Alarmy jsou konfigurovány pomocí tabulky, kde každý řádek představuje jeden alarm. Níže je v tabulce 22 uvedena struktura typického alarmu pro technologii Comfort.

Tabulka 22 Příklad definice konkrétního HMI Alarmu objektu motoru (záložka DiscreteAlarms)

Hlavička sloupce	Příklad vyplněných dat
ID	1006
Name	S0_MOTOR_001_5
Alarm text [en-US], Alarm text	<field ref="1" /> - <field ref="2" /> - SIMOCODE protection
FieldInfo [Alarm text]	"<ref id = 1; type = CommonTextList; TextList = S0_MOTORS_NAMES; Tag = DB_Motor_001; Length = 5;> <ref id = 2; type = CommonTextList; TextList = S0_MOTORS_DESCRIPTIONS; Tag = DB_Motor_001; Length = 5;> "
Class	Errors
Trigger tag	Alarms_S0_MOTOR
Trigger bit	13
Acknowledgement tag	<No value>
Acknowledgement bit	0
PLC acknowledgement tag	<No value>
PLC acknowledgement bit	0
Group	<No value>
Report	False
Info text [en-US], Info text	<No value>

Konfigurace alarmu vyžaduje specifikaci jeho aktivace (Trigger Tag a Trigger Bit), způsobu potvrzení a skupiny, do které alarm patří. Text alarmu je dynamicky generován s odkazy na textové položky, které umožňují lokalizaci a přizpůsobení textů pro různé jazyky a regiony, ale především relativně snadné navázání na velké množství objektů.

3.3.3.1. Dynamizace zobrazeného textu

Struktura je taková, že za položku „<field ref = „1“/>“ ve sloupci „Alarm text“ se dosadí údaje ze sloupce „FieldInfo“. V uvedeném případě v tabulce 22 konkrétně „<ref id = 1; type = CommonTextList; TextList = S0_MOTORS_NAMES; Tag = DB_Motor_001; Length = 5; >“. Tato reference říká, že za dříve zmíněnou položku v textu alarmu dosadí program údaje z textového listu S0_MOTORS_NAMES, ze kterého vyčte textovou položku pod hodnotou danou HMI Tagem DB_Motor_001. Druhá reference funguje stejně, akorát místo názvu dosadí popis daného objektu.

Co se týče následného generování, bude naším cílem pro každý objekt modifikovat sloupce „ID“, „Name“, „FieldInfo“ a „TriggerBit“. Jednotlivé operace budou opět vysvětleny později.

3.3.4. Generování importovatelných Excel souborů

Tato sekce se zabývá metodou generování .xlsx souborů připravených k importu do softwaru operátorských panelů v prostředí TIA Portal. Hlavním cílem je zjednodušit práci programátorů efektivním využitím dat již vyplněných v tabulkách uživatelského vstupu pro automatické generování instancí standardizovaných objektů. Popíšeme tedy strukturu, funkcionalitu a implementaci algoritmu, který tato data převede do formátu vhodného pro snadný import do projektu v prostředí TIA Portal.

3.3.4.1. Navržená metoda a její funkcionalita

Metoda generování .xlsx souborů pro tento projekt využívá dynamické šablony, které korespondují s datovými strukturami požadovanými ve vizualizačním prostředí TIA Portal. Tento přístup zajišťuje vysokou míru flexibility, umožňuje adaptaci na změny nebo rozšíření generovaných struktur bez nutnosti zásahu do kódu. Úpravy se provádějí přímo na úrovni šablony, což minimalizuje riziko chyb a zvyšuje produktivitu při údržbě aplikace.

3.3.4.2. Hlavní kroky algoritmu

- **Inicializace šablony** - Na základě typu generovaného .xlsx souboru se nastaví šablona obsahující potřebné struktury a konfigurace. Příkladem jsou data uvedená v předchozích kapitolách, potřebná ke specifikaci HMI tagů, alarmů a textových listů. V šabloně se modifikují potřebné kolonky pomocí zástupných symbolů, které indikují dynamicky propisovaná data z aktuálně generovaného objektu. Příklad šablony je vidět na obrázku 36.
- **Načtení dat** - Algoritmus načítá data z konkrétní konfigurované šablony postupně po sekcích oddělených klíčovými slovy, jako jsou „Config“, „Header“, „Rules“ a specifické položky pro daný typ standardizovaného objektu (např. „S0_Motors“, „S1_Valves“). Zástupné symboly jsou označeny textem ve složených závorkách, např. „{ID}“.
- **Generování výstupních dat** - Výstupní data jsou generována kopírováním dat ze šablony. Data z šablony jsou kopírována pro každou instanci generovaného objektu nebo jednou pro každý generovaný typ objektu. Při kopírování dat se zástupné symboly nahrazují daty z

uživatelského vstupu pro aktuální instanci objektu. Klíčová slova z načtených dat „Rules“ definují operace, které modifikují zpracování příslušných sloupců šablony. Příkladem může být vynechání sloupců s <No Value> pro redukci velikosti výsledného souboru a zrychlení generování, nebo pro explicitní zapsání dat jako textu pomocí prefixu „‘“ (Excel totiž při zpracování může interpretovat některé vstupy, například True/False jako proměnou typu Boolean a zapsat je do nového souboru v tomto formátu, což by vyvolalo chyby, jelikož je zapíše v jazyce, ve kterém je nastavené prostředí Excel. Tedy například jako Pravda/Nepravda).

	A	B	C	D
1	**Config**			
2		**Sheet Names**	DiscreteAlarms	
3		**Write to**	DiscreteAlarms	
4		**Repeat**	YES	
5	**Rules**			
6		Increment		
7	**Header**			
8		ID	Name	Alarm text [&Language&], Alarm text
9	**S0_Motors**	cs-CZ		
10		1001	S0_MOTOR_{DB NUMBER}&MOD&_0	<field ref="1" /> - <field ref="2" /> - Není připraven
11		1002	S0_MOTOR_{DB NUMBER}&MOD&_1	<field ref="1" /> - <field ref="2" /> - Přetížení
12		1003	S0_MOTOR_{DB NUMBER}&MOD&_2	<field ref="1" /> - <field ref="2" /> - Chyba zpětného hlášení
13		1004	S0_MOTOR_{DB NUMBER}&MOD&_3	<field ref="1" /> - <field ref="2" /> - Chyba zpětného hlášení vzad
14		1005	S0_MOTOR_{DB NUMBER}&MOD&_4	<field ref="1" /> - <field ref="2" /> - Nezastaveno
15		1006	S0_MOTOR_{DB NUMBER}&MOD&_5	<field ref="1" /> - <field ref="2" /> - Jištění SIMOCODE
16		1007	S0_MOTOR_{DB NUMBER}&MOD&_6	<field ref="1" /> - <field ref="2" /> - Zámek LOTO
17		1008	S0_MOTOR_{DB NUMBER}&MOD&_7	<field ref="1" /> - <field ref="2" /> - Mimo provoz
18		1009	S0_MOTOR_{DB NUMBER}&MOD&_8	<field ref="1" /> - <field ref="2" /> - VFD Chyba
19		1010	S0_MOTOR_{DB NUMBER}&MOD&_9	<field ref="1" /> - <field ref="2" /> - VFD chyba komunikace
20		1011	S0_MOTOR_{DB NUMBER}&MOD&_10	<field ref="1" /> - <field ref="2" /> - Chyba sledování otáček
21		1012	S0_MOTOR_{DB NUMBER}&MOD&_11	<field ref="1" /> - <field ref="2" /> - Jištění stykače
22		1013	S0_MOTOR_{DB NUMBER}&MOD&_12	<field ref="1" /> - <field ref="2" /> - Teplota motoru
23		1014	S0_MOTOR_{DB NUMBER}&MOD&_13	<field ref="1" /> - <field ref="2" /> - Jištění brzdy
24		1015	S0_MOTOR_{DB NUMBER}&MOD&_14	<field ref="1" /> - <field ref="2" /> - Rezerva
25		1016	S0_MOTOR_{DB NUMBER}&MOD&_15	<field ref="1" /> - <field ref="2" /> - Rezerva

Obrázek 36: Příklad struktury vzorového listu pro generování souborů typu xlsx pro import alarmů

3.3.4.3. Implementace

Algoritmus čte a interpretuje šablony specifikované na samostatném listu Excelu uživatelského vstupu. Podle dat vyčtených ze šablony modifikuje svojí funkci, dosazuje data za zástupné symboly a provádí dynamické generování obsahu podle zadaných pravidel. Algoritmus vytvoří nový soubor typu xlsx, do kterého vyplní zpracovaná data. Tento soubor obsahuje listy dle specifikace v sekci Config uvedené v šabloně.

Místy jsou v šabloně symboly ampersand, které indikují speciální operaci zavedenou přímo v kódu. Například „&MOD&“, usekne z dosazovaného čísla prefix indikující řadu standardizovaného objektu (tedy místo 8003 zapíše 003). Jedná se o řešení, které indikuje atypickou operaci při zpracování šablony.

Princip funkce je takový, že klíčová slova standardizovaných objektů se vyhledávají podle již existujících listů přímo v souboru uživatelského vstupu. Pokud jsou nalezeny a uživatel provedl při konfiguraci volbu tohoto typu, načte data z listu až k dalšímu klíčovému slovu, zpracuje je a uloží do generovaného souboru. Pro snazší orientaci v tomto postupu je níže uvedený pseudokód popsaný Algoritmem 8 a Algoritmem 9, znázorňující strukturu čtení vzorového listu a ještě o něco níže je další pseudokód vysvětlující zpracování dat.

Algoritmus 8: Generování Excel souborů k importu do TIA Portalu

```
    Input: Template sheet name and target language
    Output: Excel file filed with data according to the template
1   rowCount ← Detect range of used rows in template sheet
2   columnCount ← Detect range of used columns in template sheet
3   Create new workbook
4   FOR each row in target sheet DO
5   |  IF templateSheet.cell(row, 1) contains „Config“ THEN
6   |  |  SET: Names of generated sheets in new workbook
7   |  |  SET: Sheet to generate into in the new workbook
8   |  |  repeatConfig ← Apply template segment for every object
9   |  End IF
10  |  IF templateSheet.cell(row, 1) contains „Rules“ THEN
11  |  |  rules ← load keywords for each column from next row
12  |  End IF
13  |  IF templateSheet.cell(row, 1) contains „Header“ THEN
14  |  |  Write the following row as a header to configured sheet
15  |  End IF
16  |  IF templateSheet.cell(row, 1) contains objectKeyword THEN
17  |  |  IF objectKeyword was selected by user THEN
18  |  |  |  segmentSize ← get number of rows until next keyword
19  |  |  |  templateData ← Load all data in segment to memory
20  |  |  |  ProcessData(templateData,objectKeyword,rules,config)
21  |  |  End IF
22  |  End IF
23  Next row
24  Save and close created workbook
```

Algoritmus 9: Algoritmus zpracování vzorových dat (ProcessData)

```
    Input: templateData, objectKeyword, rules array, configuration
    Output: Excel file filed with data according to the template
1   FOR each object in objectKeyword sheet DO
2     FOR each element in templateData DO
3       IF element contains „{“ and „}“ THEN
4         Detect keyword between the brackets
5         Lookout the keyword in objectKeyword sheet as column header
6         Get the keyword value from the object row and header column
7         Suplement the keyword and brackets with the value
8       END IF
9       IF element contains „&“ twice THEN
10        Perform additional operations based on the identifier
11        Remove the identifier
12      END IF
13      Process the current element data according to the rules
14      Store the processed data to a buffer
15      IF buffer.size > bufferMaxSize THEN
16        Write buffer data to the configured file
17        Reset buffer
18      END IF
19    Next element
20    IF not useRepeat THEN
21      BREAK FOR
22    END IF
23  Next row
24  Write unwritten buffer data to the configured file
25  Return configured file
```

V současné podobě projekt podporuje generování Tagových, textových a Alarmových listů jak pro operátorské panely Comfort, tak panely Unified pro všechny typy generovaných standardizovaných objektů. Tato podpora se navíc vztahuje na český a anglický jazyk, s velice jednoduchou možností rozšíření pro další jazyky. To je dáno tím, že stačí akorát zkopírovat celou strukturu a nahradit pouze statické alarmové hlášky.

3.3.4.4. Práce s daty (čtení/zápis)

Při generování výstupních souborů je klíčové zajistit efektivní manipulaci s daty, aby se minimalizoval vliv na výkon a dobu běhu programu. Program upřednostňuje práci s celými bloky dat místo jednotlivých buněk, což výrazně zkracuje čas potřebný pro operace s pamětí. Při dosažení limitní velikosti načítaného segmentu program data zapíše zpět do paměti a proces se iterativně opakuje.

Empirické testy stanovily optimální velikost segmentu na 100 řádků, přičemž každý obsahuje počet položek dle velikosti šablony. Tento přístup snížil čas potřebný pro generování souboru o velikosti 1 MB ze zhruba 4,5 minuty na přibližně 8 vteřin, což je zrychlení o 3375 %. Tyto údaje jsou orientační, ale ilustrují význam efektivní manipulace s daty pro celkovou dobu běhu programu, což je kritické při opakovaném spouštění programu pro opravu drobných chyb ve vstupních datech.

3.4. Kontrola vstupních dat

V tomto projektu je hlavním úkolem uživatele zadávat vstupní data, která specifikují a parametrizují potřebné instance standardizovaných objektů dle potřeb projektu, což často vyžaduje manipulaci s velkým objemem informací. Některá vstupní data musí být vyplněna ve specifickém formátu kvůli kompatibilitě s datovými typy prvků v knihovních XML souborech. Abychom uživatelům usnadnili práci a minimalizovali množství chyb, byl zaveden kontrolní mechanismus vyplněných vstupních dat, jehož popisu se věnuje tato kapitola.

Níže popíšeme typy automatických kontrol při spuštění algoritmu, mechanismus určující na jaká vstupní data se kontroly aplikují, a jak funguje samotný algoritmus prováděné kontroly.

3.4.1. Typy chybových kontrol

- **Kontrola Duplicit** - Unikátnost generované instance standardizovaného objektu je dána jeho názvem a číslem datového bloku. Tento test načítá vstupní data z těchto sloupců a kontroluje, zda se každý údaj vyskytuje právě jednou.
- **Kontrola prázdných buněk** - Pro ulehčení uživatelské obsluhy je tabulka vstupních dat navržená tak, aby bylo potřeba vyplňovat co nejméně údajů. V případě nevyužitých vstupních/výstupních signálů, či definic použitých funkcionalit stačí ponechat prázdné kolonky a program je interpretuje jako hodnoty ve výchozím nastavení. Toto ovšem neplatí z implementačních důvodů pro hodnoty konstant, názvy objektů, čísla datových bloků a konfigurace skupin. Pro tyto vstupní datové položky je nutné provést kontrolu vyplnění, a vyhlásit chybu pokud jsou prázdné.
- **Kontrola zakázaných symbolů** - Na základě sloupců Name, Group a Group short se v mezi-procesu generování vytvoří soubory XML a Adresářové struktury, které mají názvy složené z těchto dat. Obsahují-li proto tyto sloupce zakázané symboly v prostředí Windows, dojde při funkci aplikace k chybě. Tento test má tyto symboly předem odhalit a upozornit na ně uživatele.
- **Kontrola konzistence skupin** - Abychom se vyhnuli duplicitám z hlediska importování funkcí do projektu, je nezbytné zajistit konzistenci vstupních dat mezi sloupci Group a Grp short. To znamená, že pro jeden typ identifikátoru sloupce Group, existuje právě jeden identifikátor Grp short a vice versa.
- **Kontrola mezery na začátku** („leading space“) - V průběhu testování se ukázalo, že někdy vyvolá chyby, když je na začátku názvu objektu mezera před samotným textem. Příčina a rozsah této chyby se důkladně netestoval a místo toho vznikl automatický test, který detekuje přítomnost mezery na začátku pro všechny sloupce, které specifikují název a umístění generovaného objektu.
- **Kontrola časových konstant** - Definice časových konstant objektu, tedy například doba houkání varovné sirény, musí být ve specifickém formátu, který pro ně TIA Portal definuje. Příkladem může být „T#20s“. Formát je testován programově pomocí regulárního výrazu. Ten specifikuje, že údaj začíná prefixem „T#“, je následován číslicí, a zakončen údajem o jednotkách a bere v potaz i různé možné validní kombinace těchto jednotek (regulární výraz vypadá konkrétně jako: "^T#(?:\d+h)?(?:\d+m)?(?:\d+s)?(?:\d+ms)?$").
- **Kontrola celočíselného vstupu** - Některé vstupní položky, jako číslo datového bloku, index prvku ve skupině, či identifikátor položky v textovém listu udávající fyzikální jednotku, jsou povolené pouze v celočíselném formátu. Tato kontrola má tedy za úkol zkontrolovat, zda

buňky obsahují pouze numerické hodnoty, či nikoliv. Desetinná čísla rovněž vyhodnocuje jako chybu.

- **Kontrola desetinného vstupu** - Jisté parametry, jako například set-pointy a limity měřených signálů, vyžadují, aby byl formát vstupních dat ve formátu Real, tedy desetinného čísla. Později generované XML soubory zmíněný formát vyžadují. V průběhu generování, se tedy kontroluje, zda je vstupní položka celočíselná, a pokud ano, tak se jí přidá dodatečně koncovka s desetinou tečkou a nulou. Pokud není celočíselná a obsahuje právě jednu desetinou tečku, nebo právě jednu desetinou čárku, vyhodnotí test jako úspěšný a čárku nahradí tečkou.
- **Kontrola Set/Reset buněk** - Buňky specifikující definice inicializační funkce v TIA Portalu jsou navrženy tak, že v případě prázdné buňky vyplní výchozí hodnotu objektu. Krom toho, přijímají vstupy jako „SET, RESET, S, R, 1, 0“. Ve výsledku se všechny tyto vstupní hodnoty převedou pouze na „S“ a „R“. Náplní tohoto testu je kontrolovat, zdali buňky neobsahují něco jiného než povolený výčet symbolů

3.4.2. Algoritmus kontroly vstupních dat

Navržený algoritmus pro kontrolu vyplněných vstupních dat kontroluje správnost dat zadaných uživatelem do tabulek Excelu. Poskytuje rychlou a intuitivní zpětnou vazbu o případných chybách ve vstupních datech. Tato zpětná vazba je dvojího typu:

- **Interaktivní Dialogové Okno** - Po spuštění programu se v případě detekování chyb zobrazí uživateli okno informující o celkovém počtu nalezených chyb. Okno uživatele vyzve, jestli chce pokračovat v generování navzdory chybám, či nikoliv.
- **Detailní Report Soubor** - Pro hlubší analýzu poskytuje algoritmus na svém výstupu soubor s podrobným popisem a lokalizací chyb podle individuálních listů a objektů, což umožňuje efektivnější revizi a opravu dat.

Proces kontroly automaticky probíhá při zahájení zpracování dat z uživatelského vstupu a začíná načtením specifických konfiguračních parametrů ze vzorového listu, což umožňuje algoritmu dynamicky aplikovat testy relevantní pro každý typ objektu a cílit pouze na jeho konkrétní sloupce, jak je ukázáno ve vizuálním příkladu na obrázku 37.

	A	B	C	D	E	F
1	Config					
2		Output file	Report			
3		Colors	Target	Even	Odd	
4			Fault			
5			Rows			
6			{Name}			
7			{DB number}			
8	S0_Motors					
9		Duplicity_Check				
10			{Name}	{DB number}		
11		Include_Empty_Check				
12			{Name}	{Group}	{Grp short}	{DB number}
13			{FeedbackDelay}	{FeedbackStopDelay}	{OnDelay}	{OffDelay}
14		Symbols_Check				
15			Symbols	<>:"/|?*☺☻♥♦♣♠•◘○♂♀♪♫☼►◄↕‼¶§▬↨↑↓→←∟↔▲▼		
16			{Name}	{Group}	{Grp short}	
17		Groups_Check				
18			{Group}	{Grp short}		
19		Leading_space				
20			{Name}	{Group}	{Grp short}	
21		Times_check				
22			{FeedbackDelay}	{FeedbackStopDelay}	{OnDelay}	{OffDelay}
23		Values_check				
24			{DB number}	{Grp No}	{Index in Grp}	{VFD_Type}
25		Floats_check				
26			{Setpoint_Low}	{Setpoint_High}	{Slow_Speed}	{Setpoint_Local_Increment}

Obrázek 37 Příklad šablony pro spuštění testů k detekci chyb vstupních dat

Algoritmus vykonává následující posloupnost operací:

- Načtení specifických konfiguračních parametrů ze vzorového listu, což umožňuje dynamicky aplikovat testy relevantní pro každý typ objektu a cílit pouze na jeho konkrétní sloupce.
- Vyhledávání klíčových slov standardizovaných objektů v prvním sloupci vzorového listu.
- Kontrola, zda uživatel povolil generování daného typu objektu.
- Načtení dat z šablony do paměti až do dalšího klíčového slova.
- Identifikace typů testů pomocí klíčových slov ve druhém sloupci šablony a dynamické získávání seznamu cílových sloupců pro daný test.
- Aplikace testů na všechny uvedené sloupce a logování zjištěných chyb s přesnými souřadnicemi problémové buňky.
- Zvýraznění chybných buněk pomocí předem nastaveného barevného schématu a vytvoření strukturovaného reportu výsledků.

Tato metoda kontroly je pro názornost popsána i Algoritmem 10 a Algoritmem 11. Metoda poskytuje robustní rámec pro zajištění integrity a správnosti vstupních dat, což je nezbytné pro bezproblémovou funkčnost celé aplikace, kde předností uvedeného postupu je jednoduchá rozšiřitelnost pro další typy standardizovaných objektů a možnost snadných úprav.

Algoritmus 10: Smyčka kontroly vstupních dat zadaných uživatelem

```
    Input: Target (rules) sheet name
    Output: Excel file filed with test results and highlighted cells
1   rowCount ← Detect range of used rows in template sheet
2   columnCount ← Detect range of used columns in template sheet
3   FOR each row in target sheet DO
4       IF templateSheet.cell(row, 1) contains „Config“ THEN
5           SET: Output file name
6           Load: Color scheme for reformating and its target columns
7       End IF
8       IF templateSheet.cell(row, 1) contains objectKeyword THEN
9           IF objectKeyword was selected by user THEN
10              Reformat objectKeyword sheet using loaded color scheme
11              segmentSize ← get number of rows until next keyword
12              templateData ← Load all data in segment to memory
13              Errors ← RunTests(templateData,objectKeyword, config)
14          End IF
15      End IF
16  Next row
17  Create new workbook using output file name
18  Fill Errors to the workbook
24  Save and close created workbook
```

Algoritmus 11: Běh testů pro konkrétní typ objektů

```
    Input: templateData, objectKeyword, configuration
    Output: Data structure loging encountered errors
1   identifierIdx ← 2
2   FOR each dataRow in templateData DO
3   |  IF dataRow(identifierIdx) is not empty DO
4   |  |  testName ← dataRow(identifierIdx)
5   |  |  NEXT dataRow
6   |  END IF
7   |  FOR each element in dataRow DO
8   |  |  IF element contains „{" and „}" THEN
9   |  |  |  Detect keyword between the brackets
10  |  |  |  Lookout the keyword in objectKeyword sheet as column header
11  |  |  |  Get the keyword column index
12  |  |  |  Run testName for whole target column of objectKeyword sheet
13  |  |  |  Store each fault into errorStructure with cell coordinates
14  |  |  END IF
15  |  Next element
16  Next dataRow
17  Highlight faulty cells with Fault color from configuration
18  Return errorStructure
```

4. Ověření funkcionality

Funkčnost navrženého a implementovaného řešení byla průběžně ověřována během vývoje v rámci testovacích etap, přičemž se kladl důraz na odstranění co největšího počtu implementačních chyb. Aplikace automatického generátoru standardizovaných objektů byla zároveň v různých fázích vývoje prakticky nasazena během procesu přípravy tří firemních projektů, z nichž jeden je již aplikován v provozu, a zbylé dva stále ve fázi vývoje.

V následujících kapitolách demonstrujeme proces nasazení aplikace z pohledu uživatele, se zaměřením na kroky potřebné k použití aplikace, nikoli na podrobný postup vyplnění uživatelských dat. Uvedeme také orientační údaje o množství času ušetřeného nasazením a vývojem aplikace, založené na interních zkušenostech s manuální tvorbou instancí objektů a měřených dobách vykonávání jednotlivých programových částí v závislosti na počtu generovaných instancí objektů.

4.1. Proces generování z pohledu uživatele

V předchozích kapitolách jsme se zaměřili na teoretické základy a popis implementace našeho projektu. Tato kapitola se tedy bude věnovat představení celého projektu z pohledu uživatele, se zaměřením na praktické použití aplikace.

Následující podkapitoly se nejprve zabývají potřebnými soubory, které musí uživatel přetáhnout na své zařízení. Na to navazujeme popisem uživatelské obsluhy při zpracování uživatelského vstupu, bez detailního popisu jeho samotného vyplnění. Nakonec předvedeme použití aplikace SIBLOCK Generator a příklad projektu s vygenerovanými objekty.

V následujících kapitolách budeme pro demonstrativní účely uvažovat scénář, kdy jsme nasazovali vyvinutou aplikaci na nový projekt pro bioplynovou stanici. Nebudeme zde zabíhat do podrobností, ale pro kontext se jedná o soupis objektů dle tabulky 23.

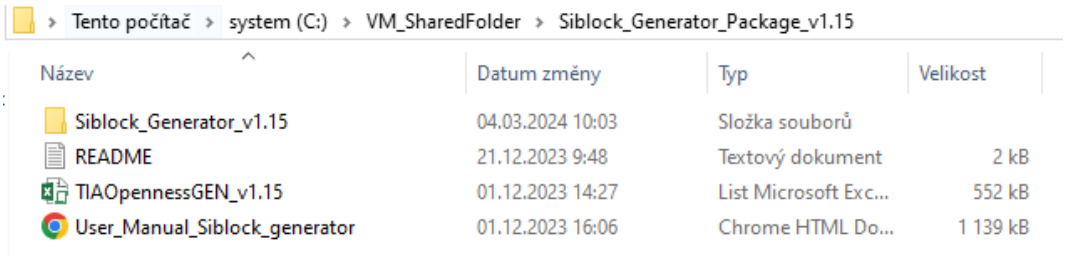
Tabulka 23 Soupis objektů generovaných v popisovaném projektu

Typ standardizovaného objektu	Počet generovaných instancí
S0_Motors	64
S1_Valves	64
S2_Analogs	147
S4_Digitals	92
S7_Dampers	1

4.1.1. Rozbalení generátoru

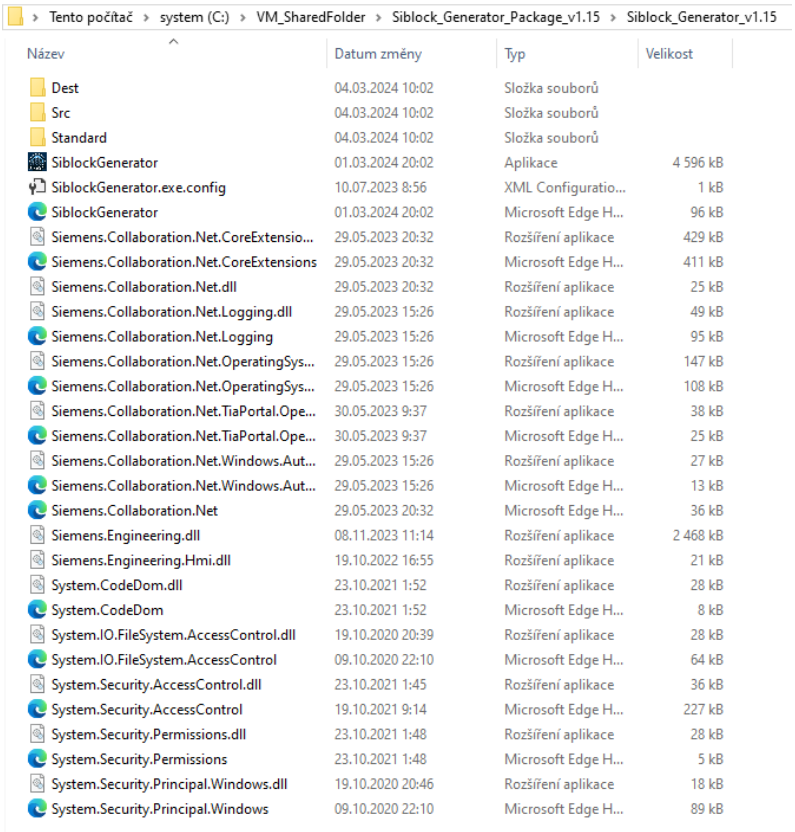
Prvním krokem uživatele k použití naší aplikace je stažení předpřipraveného archivu z firemního síťového disku. Tam je na definované cestě k dispozici adresář „Siblock_Generator_Package_v1.15“, který obsahuje veškeré potřebné soubory, textový dokument README s potřebnými instrukcemi a v neposlední řadě i uživatelský manuál. Tento manuál vysvětluje všechny potřebné a možné operace, které může uživatel udělat a zmiňuje nejčastější chyby v procesu generování, spolu s jejich možným řešením.

Obsah tohoto adresáře je k vidění na obrázku číslo 38. Instrukce pro uživatele jsou takové, že po rozbalení celého balíčku si umístí adresář „Siblock_Generator_v1.15“ na zařízení s prostředím TIA Portal, do kterého bude chtít provést automatické generování. Soubor Excelu s názvem „TIAOpennessGEN_v1.15“ si následně umístí libovolně tak, aby se mu pohodlně vyplňovala specifikace generovaných souborů. Praxe je totiž taková, že zařízení s prostředím TIA Portalu nemusejí mít nainstalované prostředí Excelu a tak je pohodlnější vyplnit si tuto tabulku nezávisle bokem.



Obrázek 38 Obsah archivu Siblock generator package na firemním síťovém disku

Obsah adresáře „Siblock_Generator_v1.15“ je k nahlédnutí na obrázku 39. V adresáři se primárně nachází spustitelná aplikace SIBLOCK Generator a tři adresáře obsahující buď knihovní XML soubory, nebo v případě složky „Dest“ později vygenerované XML soubory. Avšak tím se uživatel nemusí zabývat, ani s nimi ručně manipulovat. Zbylé soubory v adresáři vznikly instalací balíčku „Siemens.Collaboration.Net.TiaPortal.Openness.Resolver“ a pro uživatele nejsou relevantní.

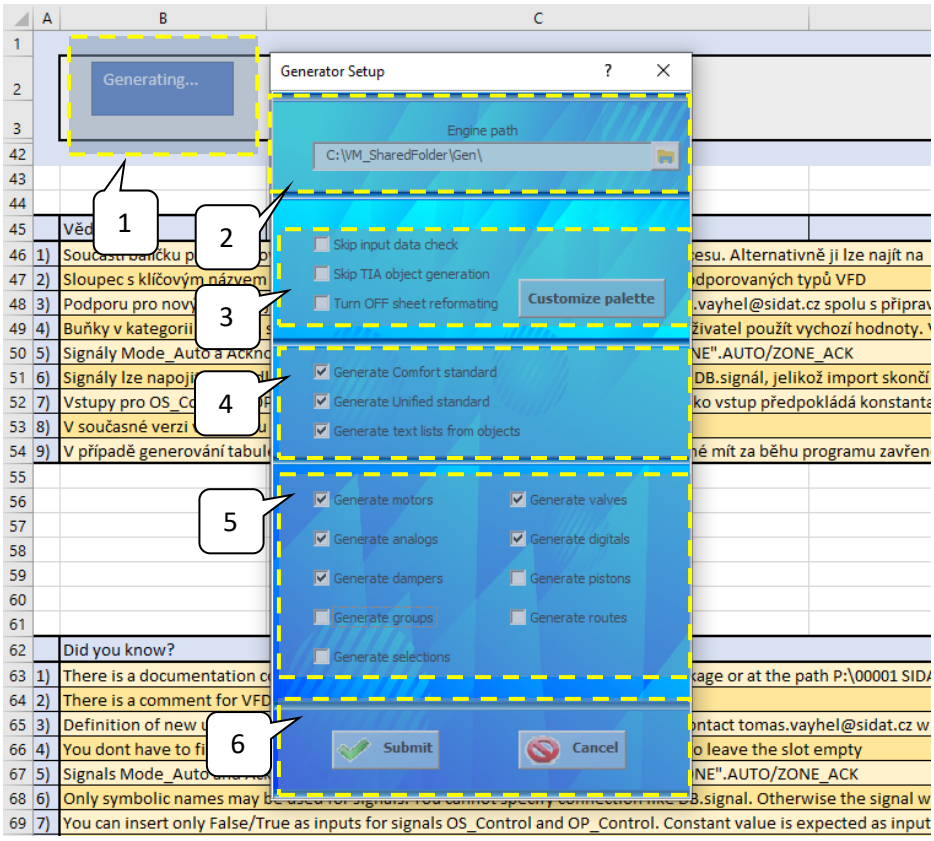


Obrázek 39 Obsah adresáře aplikace SIBLOCK Generátoru

4.1.2. Pokyn zpracování uživatelského vstupu

V kapitole 3.2.2.1 jsme popsali, jak se předpokládá vyplnění uživatelského vstupu do předpřipravené tabulky v Excelu, takže to zde nebudeme zmiňovat znovu. Uživatel tato data vyplňuje na základě specifikace uvedené v elektrických schématech, kterou si převede na jednotlivé instance firemně standardizovaných objektů.

Po vyplnění všech potřebných parametrů zadá uživatel v prostředí Excel povel ke zpracování pomocí tlačítka „Generate“, na základě čehož mu vyskočí konfigurační okno, které je vidět na obrázku 40 s výrazněnými sekcemi, jejichž význam je uveden v tabulce 24 pod obrázkem. Jeho cílem je poskytnout uživateli větší svobodu a možnosti při používání tohoto nástroje.

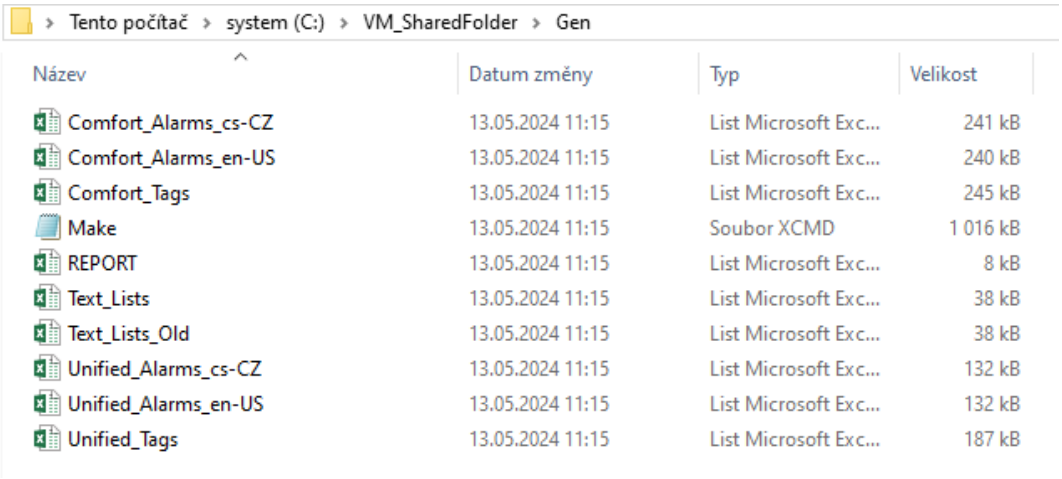


Obrázek 40 Popsané konfigurační okno zpracování uživatelského vstupu

Tabulka 24 Význam popisků na obrázku znázorňující konfigurační okno zpracování uživatelského vstupu

Číslo	Vysvětlení obsahu
1	Původně tlačítko ‚Generate‘. Po stisku popisuje stav procesu zpracování dat
2	Uživatelsky vybraná cesta, kam se vygenerují zpracované soubory
3	Možnost vynechání vybraných kroků zpracování uživatelského vstupu.
4	Možnost vygenerovat importovatelné xlsx soubory pro HMI standard
5	Výběr objektů (a záložek), pro které je zpracován uživatelský vstup v tabulce
6	Tlačítko k povolení zpracování dat a tlačítko zrušení celého procesu

V závislosti na navolených možnostech se do navoleného cílového adresáře vygenerují všechny potřebné soubory. Pro demonstrativní účely jsou na obrázku 41 znázorněny všechny možné vygenerované typy souborů. Tedy soubory jak pro vytváření instancí standardizovaných objektů, tak pro vizualizační standardy pro HMI panely řady Comfort a Comfort Unified. Při běžném použití by tedy souborů bylo méně.



Obrázek 41 Obsah adresáře s vygenerovanými soubory po zpracování uživatelského vstupu

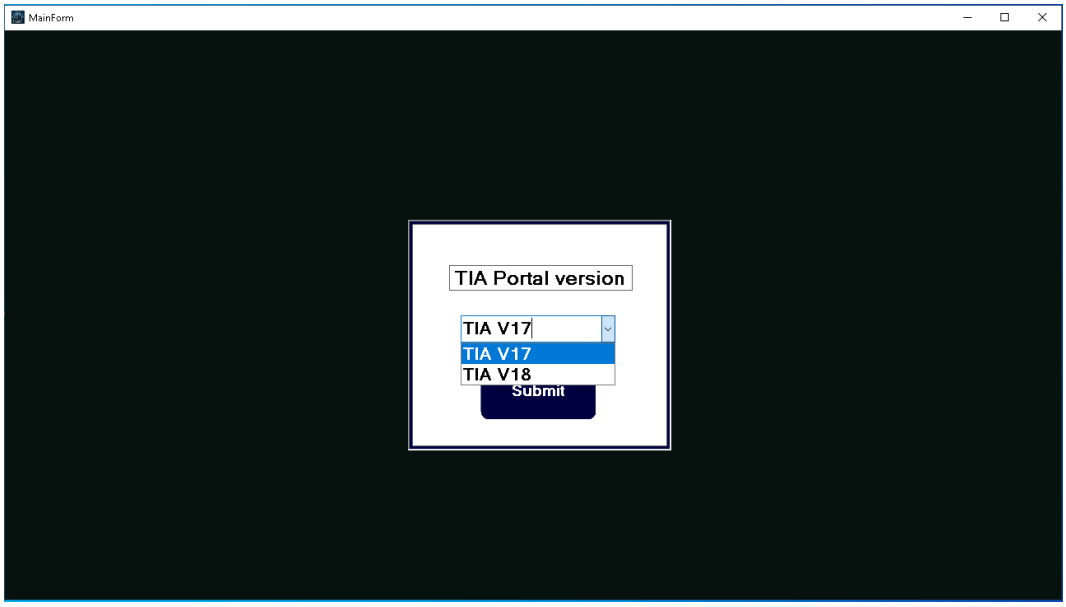
Pro lepší přehled významu všech těchto souborů, jejichž tvorbu jsme popisovali napříč celou naší prací, slouží tabulka 25. Vygenerované soubory, které si uživatel přeje využít je následně potřeba přesunout na cílové zařízení obsahující TIA Portal.

Tabulka 25 Vysvětlivky k jednotlivým typům souborů generovaných zpracováním uživatelského vstupu

Název položky	Význam
Comfort_Alarms	Unikátní xlsx soubor pro každý generovaný jazyk. Soubory jsou připraveny ve formátu, kdy je stačí manuální importovat do projektu a vytvoří standardizovanou strukturu alarmového hlášení pro Comfort HMI
Unified_Alarms	Stejné jako v případě ‚Comfort_Alarms', avšak ve formátu podporovaném na panelech řady Comfort Unified
Text_Lists	Excel soubor k importu textových listů vztažených k instancím standardizovaných objektů, stejný pro oba podporované vizualizační standardy
Comfort_Tags	Excel soubor k importu HMI Tagů v rámci firemního standardu pro řadu panelů Comfort.
Unified_Tags	Stejné jako v případě ‚Comfort_Tags' avšak pro standard firemního standardu pro řadu panelů Comfort Unified
REPORT	Excel soubor obsahující soupis nalezených chyb a jejich lokaci
Make	Příkazový soubor pro tvorbu XML souborů instancí standardizovaných objektů
_Old	Identifikátor za souborem, sloužící k záloze poslední vygenerované varianty daného souboru.

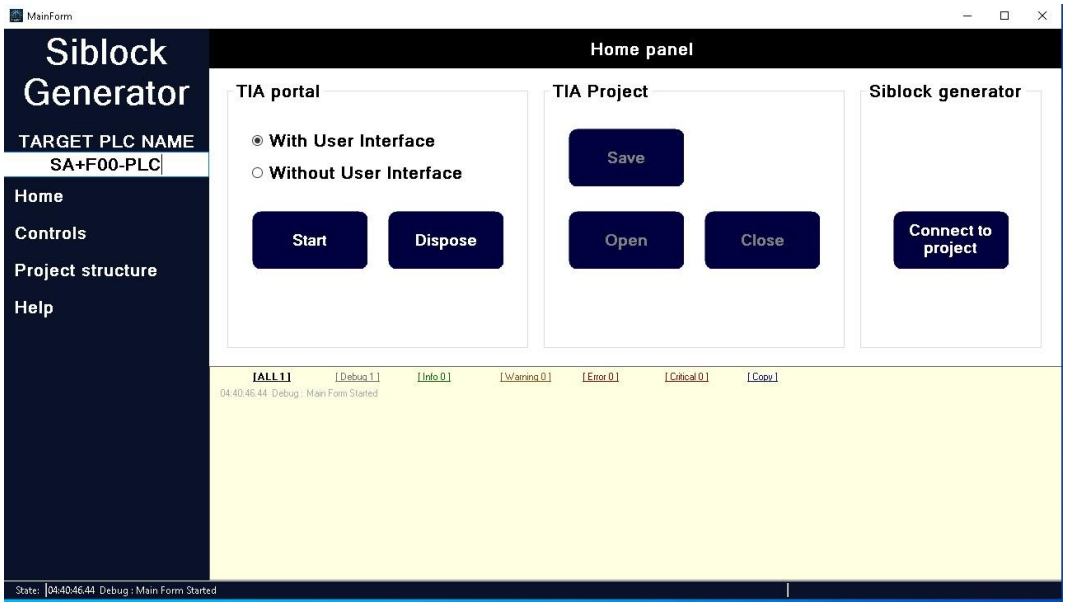
4.1.3. Obsluha aplikace SIBLOCK Generátor

Po přesunutí vygenerovaných souborů a aplikace na cílové zařízení může uživatel postoupit k dalším krokům v procesu generování. Po spuštění aplikace SIBLOCK generátor na zařízení s vývojovým prostředím TIA Portal si uživatel vybere cílovou verzi TIA Portalu z listu verzí, jejichž instalace byla detekována na daném zařízení. Přehled tohoto výběru je vidět na obrázku 42.



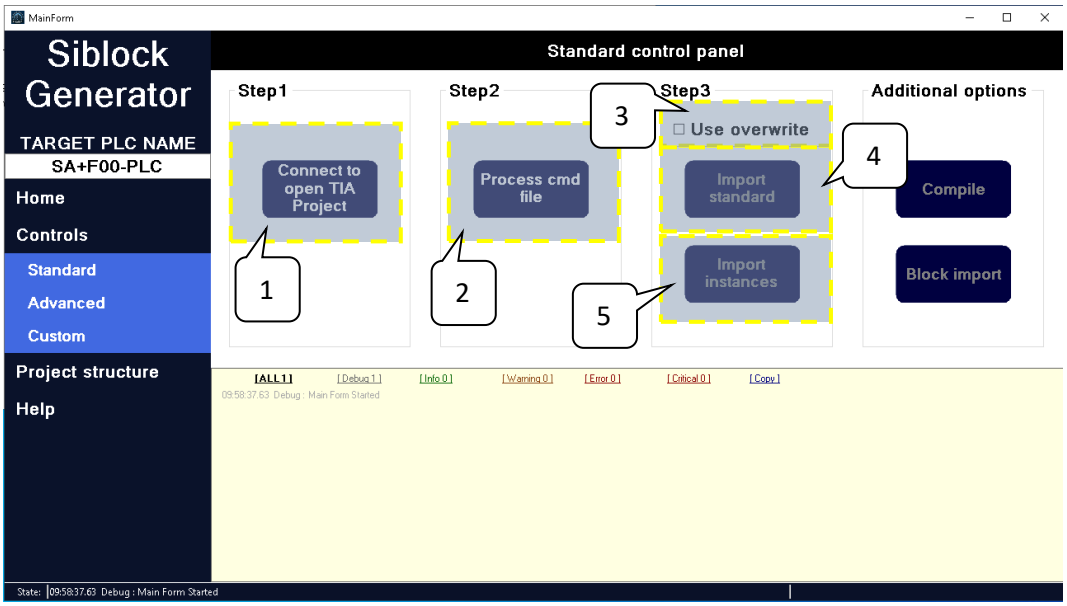
Obrázek 42 Startovací okno aplikace SIBLOCK Generator s výběrem cílové verze prostředí TIA Portal

Po výběru cílové verze TIA Portalu, se na uživatelském rozhraní zobrazí domovská záložka aplikace. Ta poskytuje uživateli nástroje ke spuštění nového procesu TIA Portalu a k manipulaci s projekty. Domovské okno aplikace je znázorněno na následujícím obrázku číslo 43.



Obrázek 43 Okno Home aplikace SIBLOCK Generátor

K využití aplikace k procesu automatického generování slouží okno Standard, které je k dispozici v rozšířené navigační nabídce pod záložkou Controls. Na tomto okně jsou intuitivně organizovány tlačítka dle posloupností operací potřebných k automatickému vygenerování objektů. Náhled uživatelského rozhraní je k vidění na obrázku 44.



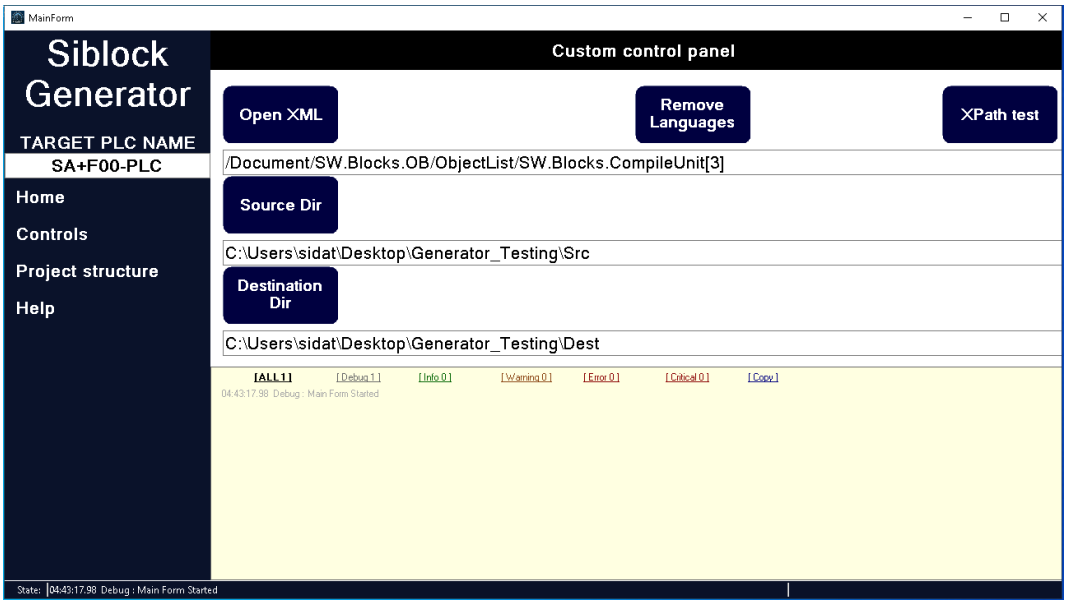
Obrázek 44 Okno Standard aplikace SIBLOCK generátor

Význam jednotlivých stěžejních kroků, a tedy i tlačítek na panelu ‚Standard' je uveden v tabulce číslo 26:

Tabulka 26 Vysvětlení vybraných tlačítek aplikace z obrázku 44

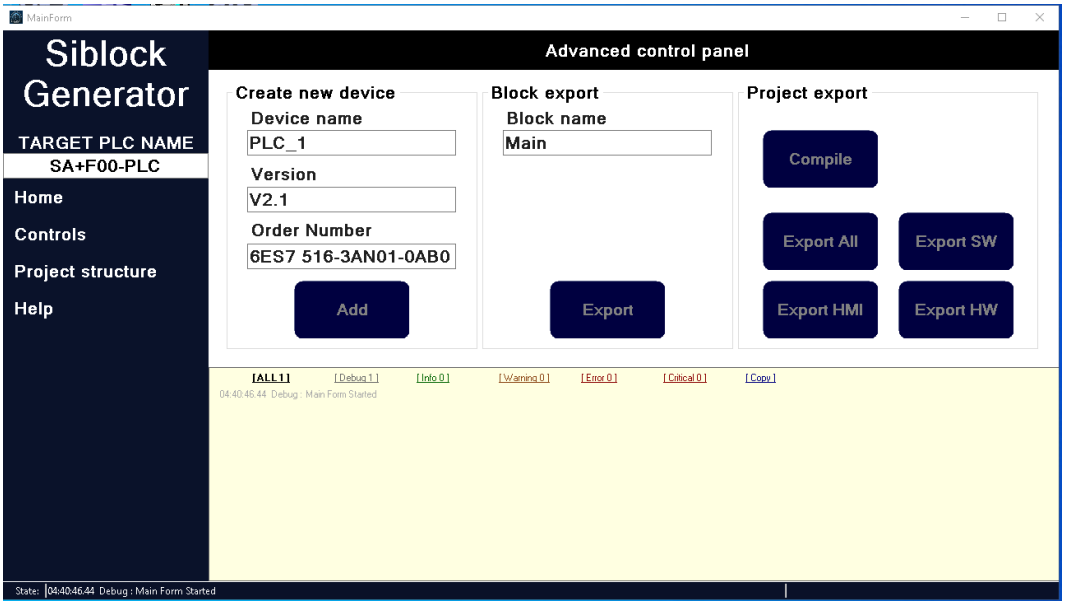
Číslo	Vysvětlení obsahu
1	**Connect to open TIA project** – Jedná se o první potřebný krok procesu, který propojí aplikaci generátoru s aktuálně otevřeným projektem v prostředí TIA Portal. Pokud, není zrovna otevřen právě jeden projekt, vypíše aplikace chybové hlášení do terminálu.
2	**Process cmd file** – Jedná se o druhý krok procesu generování. Uživatel po stisku tlačítka vybere v dialogovém okně soubor Make.xcmd, který vznikl v předchozím kroku zpracováním uživatelského vstupu z aplikace Excel. Během této operace aplikace soubor zpracuje a vytvoří na jeho základě a knihovních souborů výsledné soubory ve formátu XML, které uloží do adresáře ‚Dest', který se ve výchozím nastavení nachází v adresáři, ve kterém je umístěna aplikace SIBLOCK generátoru
3	**Use overwrite** – Aplikace při importu souborů kontroluje, zda již v projektu nejsou obsaženy stejnojmenné instance. Pokud uživatel zatrhne tuto možnost, tak se detekované kolize přepíší importovanými soubory. V opačném případě zůstanou tak jak byly.
4	**Import standard** – Importuje statickou strukturu firemního standardu nezávislou na vytvářených instancích standardizovaných objektů. Stačí provést pouze jednou při zakládání nového projektu jako součást třetího kroku před stiskem ‚Import instances'.
5	**Import instances** – Třetí krok, který importuje všechny vytvořené XML soubory definující instance standardizovaných objektů z adresáře ‚Dest' do projektu.

V rámci přizpůsobení funkcionality má uživatel možnost si v okně Custom přizpůsobit adresáře, ze kterých aplikace čerpá knihovní soubory a do kterých ukládá vygenerované soubory. Toto okno je k vidění na obrázku 45.



Obrázek 45 Okno Custom aplikace SIBLOCK generátor

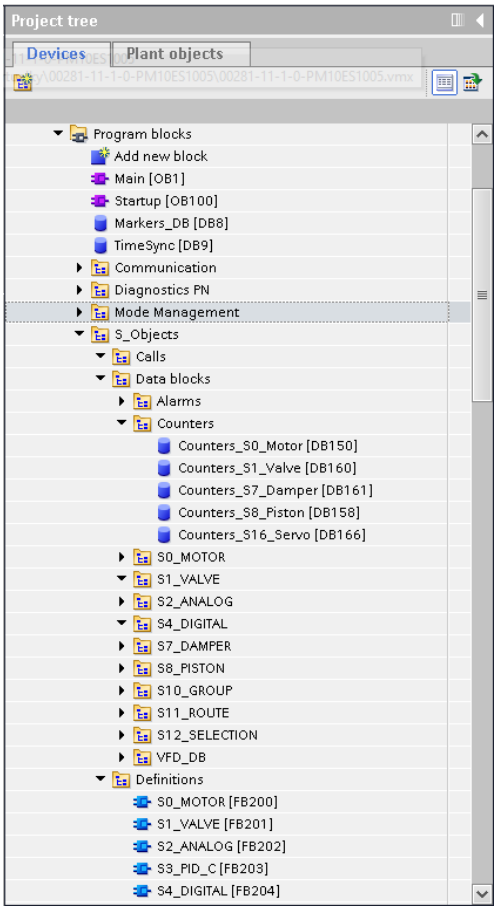
Pro úplnost zde ještě ukážeme okno Advanced, které slouží primárně k exportu objektů z projektu. Tato funkcionalita slouží především k tvorbě a aktualizaci knihovních XML souborů programátorem a běžný uživatel se s ní ve většině případů zabývat nemusí. Obrázek okna je uveden na obrázku 46.



Obrázek 46 Okno Advanced aplikace SIBLOCK generátor

4.1.4. Vygenerované objekty v projektu

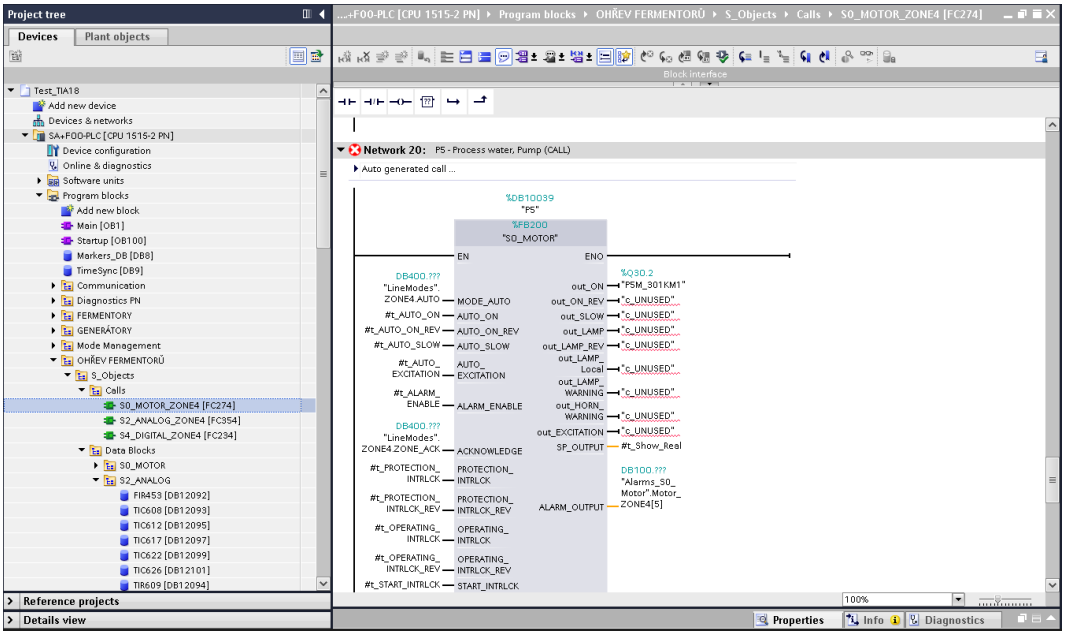
V reakci na uživatelské povely aplikaci SIBLOCK generátoru se začnou do projektu importovat XML soubory z připravených adresářů. Jelikož jsme zmínili, že po vytvoření nového projektu je potřeba prvně stisknout tlačítko ‚Import Standard', ukážeme na obrázku 47 výsledek tohoto importu v projektu. Jedná se o předpřipravenou statickou strukturu, která je neměnná napříč projekty. Hlavním cílem této operace je vytvoření základní adresářové struktury projektu, import UDT a funkčních bloků standardizovaných objektů.



Obrázek 47 Statická struktura firemního standardu importovaná tlačítkem ‚Import Standard'

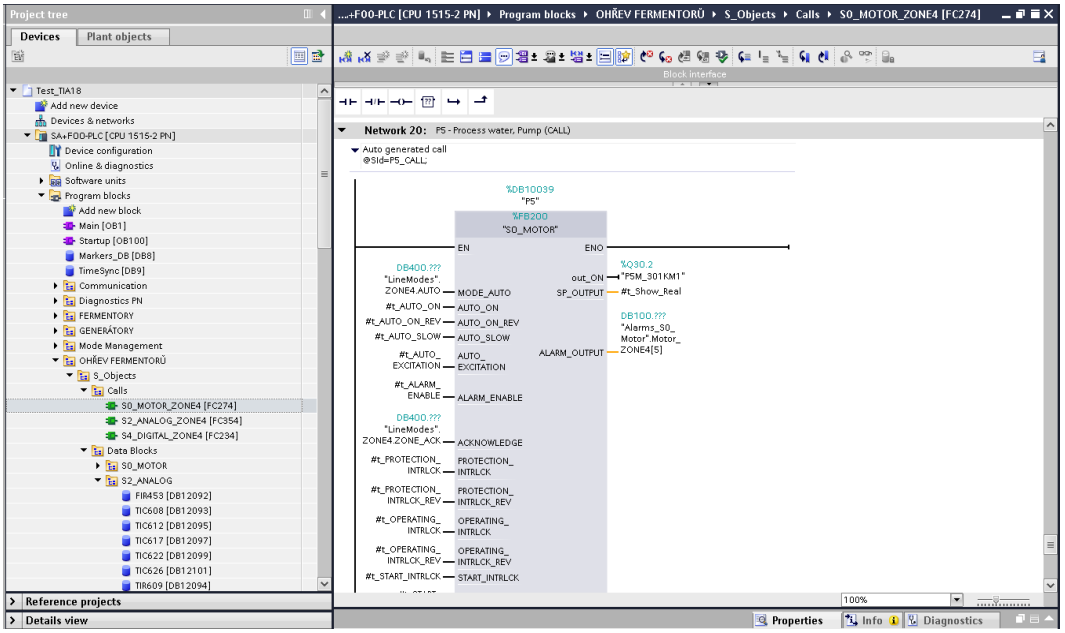
V reakci na stisk tlačítka ‚Import instances' se do projektu začnou importovat již kompletní vygenerované instance standardizovaných objektů a jejich volání. Aplikace je nastavená tak, že vygenerované soubory do projektu uspořádá do jasně dané adresářové hierarchie organizované v rámci jednotlivých podskupin, které si uživatel mohl definovat v rámci uživatelského vstupu.

Jak již bylo zmíněno v implementační části, jsou ve funkcích volání nevyužité signály nahrazené plošným tagem jménem „C_Unused", který musí uživatel dodatečně odstranit například použitím nástroje projektu „najít a nahradit". Zároveň pracujeme s předpokladem, že všechny potřebné tagy jsou již přítomné v projektu v době importu. Vygenerované objekty v hierarchii projektu a ukázka zastoupení nevyužitích signálů po generování je na obrázku 48.



Obrázek 48 Importované instance standardizovaných objektů před odstraněním nevyužitých signálů

Funkce po hromadném odstranění zástupného symbolu nevyužitých signálů je k vidění na obrázku 49. Nepřiřazené vstupy a výstupy funkčního bloku jsou automaticky skryty, což je vlastnost prostředí TIA Portal.

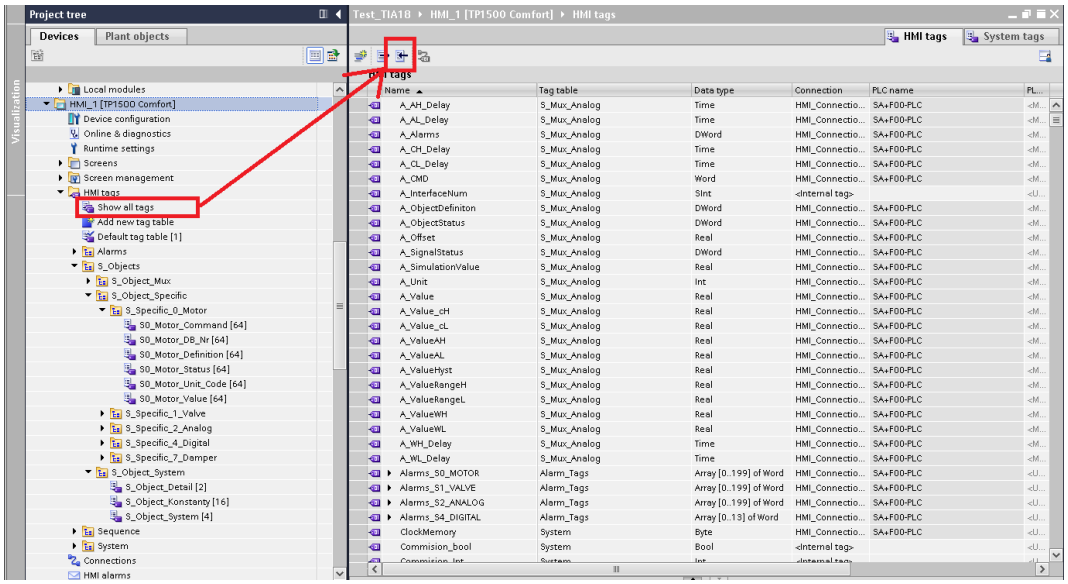


Obrázek 49 Importované instance standardizovaných objektů po plošném odstraněním nevyužitých signálů

4.1.5. Import xlsx souborů do vizualizačního prostředí projektu

Využití připravených xlsx souborů pro případný import HMI Tagů, textových listů a alarmů pro operátorské panely řady Comfort nebo Comfort Unified ukážeme velmi stručně na případu HMI Tagů.

Na cílovém zařízení totiž stačí otevřít příslušnou záložku a importovat celý list xlsx tlačítkem přímo v prostředí TIA Portal. Tento postup je naznačen na obrázku 50 s tím, že je vidět již celá importovaná struktura HMI Tagů roztříděná do jednotlivých složek a tagových listů. Roztřídění probíhá automaticky vzhledem k cestám HMI Tagů definovaných v xlsx souboru.



Obrázek 50 Importovaná struktura HMI Tagů pro operátorský panel řady Comfort

4.2. Časová úspora

Stanovit kolik tento projekt ve výsledku šetří času a práce je poměrně složitý úkon. Z případů nasazení generátoru na praktických projektech je patrné, že se jedná o významnou úsporu času. Bez rozsáhlé časové investice však nelze přesně kvantifikovat tuto úsporu v absolutních ani relativních číslech.

V následujících kapitolách ukážeme orientační časové údaje z testovacích scénářů pro jednotlivé kroky generování standardizovaných objektů a na jejich základě se pokusíme vyhodnotit přínos této aplikace v porovnání s manuální tvorbou instancí objektů v projektu.

Testovací scénáře jsou členěny podle celkového množství generovaných instancí objektů, které jsou rovnoměrně rozděleny mezi vybrané standardizované objekty. V rámci každého typu objektu jsou instance rozděleny ve stejném poměru na podskupiny generované do projektu. Přehled složení testů je vidět v tabulce číslo 27.

Tabulka 27 Soupis testovacích scénářů znázorňující rozprostření počtu instancí z jejich celkového počtu

Typ /Celkový počet	10	50	100	200	500	1000
S0_Motors	2	10	20	40	100	200
S1_Valves	2	10	20	40	100	200
S2_Analogs	2	10	20	40	100	200
S4_Digitals	2	10	20	40	100	200
S5_Dampers	2	10	20	40	100	200

4.2.1. Doba zpracování uživatelského vstupu

Při měření celkové doby zpracování uživatelského vstupu jednotlivých testů jsme brali v potaz scénář, kdy je součástí zpracování i reformátování listů Excelu, kontrola dat uživatelského vstupu a souběžné generování souborů Excelu pro standard panelů řady Comfort, navrch k zpracování příkazového souboru.

Výsledky testování jsou uvedené v tabulce 28, avšak je třeba brát na vědomí, že se jedná pouze o orientační výsledky, které jsou získané pouze jedním cyklem měření.

Tabulka 28 Naměřené doby zpracování uživatelského vstupu pro jednotlivé testovací scénáře

Počet generovaných instancí objektů [-]	Celková doba zpracování [s]
10	7
50	9
100	10
200	14
500	27
1000	47

4.2.2. Doba operací aplikace SIBLOCK generátoru

V této sekci se budeme zabývat popisem dob jednotlivých kroků procesů generování v aplikaci SIBLOCK generátor pro jednotlivé testovací scénáře. Zmíníme tedy orientačně, jak dlouho trvá zpracovat příkazový soubor v závislosti na počtu definovaných objektů tak, aby se vytvořily všechny potřebné XML soubory. A jak dlouho pak trvá tyto soubory importovat do projektu.

Z hlediska funkce importu přes rozhraní TIA Portal Openness tady zmíníme dva scénáře, a to když import probíhá do procesu TIA Portalu s uživatelským rozhraním, a když probíhá bez uživatelského rozhraní.

4.2.2.1. Zpracování příkazového souboru

V případě zpracovávání souboru ‚Make.xcmd' se jedná o operaci, která není nijak závislá na rozhraní TIA Portal Openness. V obou případech tedy trvá přibližně stejně dlouho s pouze zanedbatelnými rozdíly. Výsledné naměřené doby jsou uvedeny v tabulce 29.

Tabulka 29 Naměřené doby zpracování příkazového souboru pro jednotlivé testovací scénáře

Počet generovaných instancí objektů [-]	Celková doba zpracování [s]
10	3
50	7
100	14
200	25
500	54
1000	111

Pro lepší představu čtenáře o množství dat, se kterými se jak při zpracování, tak při pozdějším importu manipuluje, lze uvést, že v případě scénáře o tisíci instančních objektech obsahuje příkazový soubor 3 MB dat v textové podobě. Program tento soubor zpracuje a vytvoří z něj 280 MB dat v podobě různých XML souborů do necelých dvou minut.

4.2.2.2. Import standardizované struktury do projektu

Při importu standardizované struktury firemního standardu do cílového projektu se jedná o neměnnou operaci, která je nezávislá na generovaných instancích standardizovaných objektů. Naměřené doby importu přes TIA Openness jsou uvedeny v tabulce číslo 30.

Tabulka 30 Naměřené doby importu standardizované struktury do projektu dle konfigurace TIA Portalu

Doba importu s UI TIA Portalu [s]	Doba importu bez UI TIA Portalu [s]
135	94

4.2.2.3. Import vygenerovaných XML souborů do projektu

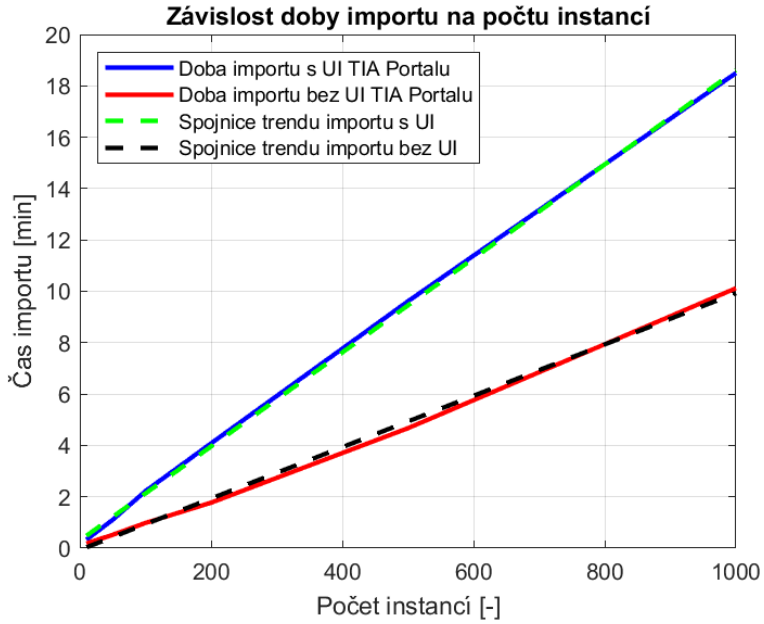
Posledním a stěžejním krokem automatického generování je import vytvořených XML souborů do připojeného projektu v prostředí TIA Portal. Tato operace probíhá přes rozhraní TIA Portal Openness a jedná se o časově nejnáročnější operaci za běhu našeho programu.

Na základě dostupných údajů v dokumentacích a předchozích zkušeností je patrné, že je jistý rozdíl v rychlosti při použití rozhraní k importu souborů do TIA Portalu s uživatelským rozhraním (UI) a v importu do TIA Portalu bez uživatelského rozhraní. V rámci testování jsme tedy naměřili rozdíly v rychlosti běhu programu mezi těmito scénáři pro náš případ. Jedná se zároveň o hlavní důvod existence této kapitoly. Následující tabulka číslo 31 uvádí naměřené údaje spolu s procentuálním zrychlením importu pro propojení TIA Portalu bez uživatelského rozhraní. Pořád je však třeba mít na paměti, že se jedná pouze o orientační údaje měřené jednorázově.

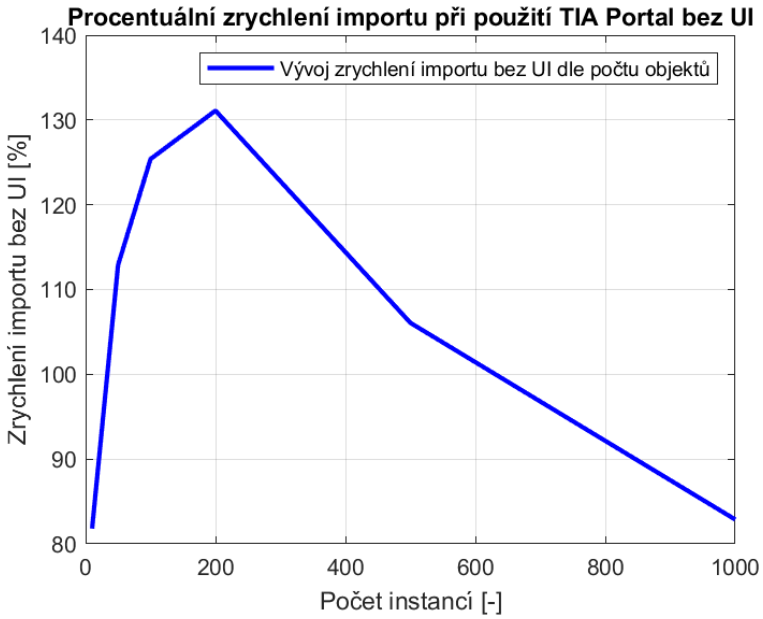
Tabulka 31 Naměřené doby importu generovaných XML souborů do projektu dle konfigurace TIA Portalu

Počet generovaných Instancí objektů [-]	Doba importu s UI TIA Portalu [min]	Doba importu bez UI TIA Portalu [min]	Zrychlení importu bez UI [%]
10	0,33	0,18	81,8
50	1,10	0,52	112,9
100	2,22	0,98	125,4
200	4,09	1,77	131,1
500	9,62	4,67	106,1
1000	18,50	10,12	82,9

Vizuální porovnání naměřených výsledků je vidět na obrázku 51. Z průběhu je patrné, že se jedná o téměř lineární závislost času importu na počtu generovaných instancí objektů.



Obrázek 51 Graf znázorňující závislost doby importu na počtu generovaných instancí objektů pro případy importu do procesu TIA Portalu s uživatelským rozhraním a do procesu bez něj



Obrázek 52 Graf znázorňující procentuální zrychlení doby importu pro případ importu do procesu TIA Portalu bez uživatelského rozhraní v závislosti na počtu generovaných instancí objektů

Zajímavým pozorováním z uvedených grafů je, že procentuální míra zrychlení importu souborů do TIA Portalu bez uživatelského rozhraní se mění napříč jednotlivými testovacími scénáři dle počtu instancí objektů, viz obrázek 52.

Zde lze spekulovat, že tento efekt může být z části daný statickým over-headem operace importu, díky čemuž se pro malý počet objektů rozdíl vynechání uživatelského rozhraní neprojeví naplno. Naopak pokles se zvyšujícím se počtem objektů může být z části způsobený navrženým testovacím scénářem. Oproti testům s menším počtem objektů totiž obsahují poslední dva testy výrazně větší počet objektů v jedné skupině, což se projeví výrazně větší velikostí importovaných XML souborů (místy i 5 MB). To může zpomalit proces importu například kvůli neefektivnímu využití prostředků přiřazených aplikaci.

4.2.3. Shrnutí

Analýza časových údajů jednotlivých procesů v předchozích sekcích ukazuje, že doby jednotlivých kroků generování se pohybují v rozmezí desítek sekund až jednotek minut, což je zanedbatelné vzhledem k množství generovaných instancí standardizovaných objektů. Hlavním faktorem ovlivňujícím dobu celého procesu generování je především rychlost uživatele při vyplňování uživatelského vstupu.

Je důležité si uvědomit, že i při manuálním vytváření instancí standardizovaných objektů musí nejprve dojít k analýze zadání a přípravě dat pro požadované instance, podobně jako v případě našeho uživatelského vstupu v aplikaci Excel. Z tohoto důvodu nedochází v tomto kroku k žádné výrazné časové úspoře.

Některé úkony, jako je příprava specifikace instancí objektů, tedy zůstávají stejné jak při manuálním, tak při automatickém generování, což celý proces tvorby objektů prodlužuje. Naše řešení však nabízí přehlednost při konfiguraci instancí objektů a možnost jednoduché zpětné kontroly vyplněných parametrů.

Ačkoliv nebylo provedeno přímé porovnání rychlosti mezi vytvořeným generátorem objektů a ruční tvorbou, zkušenosti ve firmě naznačují, že s připravenými podklady může programátor vytvořit v projektu jednu instanci objektu přibližně za 5 minut. Tento čas zahrnuje kopírování networků pro volání, přepsání názvu instance a názvů příslušných networků (ve všech jazycích), přiřazení použitých proměnných na vstupní a výstupní parametry bloku a vytvoření patřičné inicializační funkce objektu. Jedná se o odhadovanou průměrnou dobu, která závisí na typu a složitosti daného objektu.

Pokud porovnáme například scénář, ve kterém generujeme 500 instancí objektů s připravenými podklady, zjistíme, že programátor by manuální konfigurací strávil přibližně 42 hodin. Při použití vytvořeného nástroje a importu vygenerovaných instancí objektů by stejný proces zabral přibližně 10 minut se zapnutým uživatelským rozhraním TIA Portalu. Během tohoto času se navíc programátor může věnovat jiným činnostem.

Další časové úspory lze dosáhnout využitím funkce našeho nástroje pro generování HMI Tagů, alarmových hlášení a textových listů pro operátorské panely řady Comfort a Comfort Unified. K jejich tvorbě nejsou zapotřebí žádná dodatečná data navrch vyplněného uživatelského vstupu, což přináší další úsporu času odhadovanou na jednotky dní práce.

5. Závěr

Hlavním cílem této práce bylo vytvořit nástroj, který by automatizoval proces generování instancí standardizovaných objektů firmy Sidat do vývojového prostředí TIA Portal a tím výrazně zkrátit dobu potřebnou k přípravě nových projektů v oblasti průmyslové automatizace.

Celý proces generování byl navržen tak, aby uživatelsky přívětivým způsobem umožňoval zadávání specifikace požadovaných instancí objektů prostřednictvím tabulek v Excelu. Následně byla data z těchto tabulek zpracována a převedena do formátu XML, který lze přes rozhraní TIA Portal Openness importovat do projektu v prostředí TIA Portal. Vytvořený generátor objektů byl následně nasazen a testován na několika firemních projektech, kde se ukázala jeho funkčnost a efektivita.

V současné podobě dokáže vytvořený nástroj zpracovat uživatelský vstup takovým způsobem, že z něj vytvoří jak programové bloky specifikující instance standardizovaných objektů, tak i soubory xlsx, které lze importovat do projektu na operátorské panely řady Comfort nebo Comfort Unified k vytvoření HMI tagů, alarmových hlášení a textových listů navázaných na vytvářené instance.

Na základě testovacích scénářů jsme ukázali, že doba potřebná k vykonání jednotlivých programových operací, od zpracování uživatelského vstupu až po import vytvořených XML souborů do projektu, je zanedbatelná ve srovnání s časem potřebným k vyplnění uživatelského vstupu. Vytvoření specifikace instancí standardizovaných objektů je však nezbytným krokem při tvorbě každého nového projektu, a proto tento aspekt neuvažujeme při srovnání rychlosti manuálního a automatického vytváření instancí objektů. Lze tedy konstatovat, že naše práce šetří významné množství času, který by jinak byl potřebný na manuální vytvoření instancí objektů v projektu.

Zajímavým pozorováním během testů byl rozdíl v rychlosti importu vytvořených XML souborů mezi případy, kdy byl spuštěný proces TIA Portalu s uživatelským rozhraním, a když byl spuštěn bez něj. V některých případech se jednalo i o více než dvojnásobné zrychlení procesu importu. Jelikož však sloužilo testování pouze pro orientační účely, nelze výsledky použít k vyvození obecného závěru. To by vyžadovalo větší různorodost testů a sérii opakovaných měření.

Mezi hlavní přínosy vytvořeného nástroje, popsaného v této práci patří:

- Automatizace repetitivních úkonů při přípravě nového projektu v prostředí TIA Portal.
- Zvýšení efektivity a produktivity programátorů.
- Snížení počtu chyb při konfiguraci instancí standardizovaných objektů.
- Zjednodušení procesu aplikace firemního standardu na nově vytvářené projekty.

Tento nástroj bude v rámci firemní organizace začleněn přímo do struktury firemního standardu pro vývojové prostředí TIA Portal. Bude se tedy využívat během přípravy všech budoucích firemních projektů realizovaných v tomto prostředí. Projekt je již z vývoje tomuto požadavku přizpůsoben tím, že zajišťuje kompatibilitu použitých nástrojů i s dosud nevydanými verzemi prostředí TIA Portal. V případě úprav či změn ve struktuře generovaných instancí objektů pak není nutné zasahovat do kódu a stačí upravit pouze šablony, ze kterých program vychází.

Budoucí směr vývoje aplikace se výhledově bude soustředit na implementaci dalších nástrojů, jako například eliminaci nevyužitých signálů u vygenerovaných instancí objektů zlepšením manipulace s XML soubory. Další oblastí rozvoje aplikace by mohlo být generování grafických ikon instancí objektů na obrazovkách operátorských panelů s již navázanými potřebnými parametry, což by usnadnilo tvorbu vizualizace výrobních procesů v pozdějších etapách vývoje projektů.

Pro lepší pochopení praktického nasazení aplikace je v dodatku C této práce přidán odkaz na video, které demonstruje celý proces jejího nasazení na jednoduchém příkladu od specifikace uživatelského vstupu až po přehled vygenerovaných objektů.

6. Seznam literatury

[1] SIEMENS. Programming Guideline for S7-1200/1500. Online. 2018. Dostupné z: https://cache.industry.siemens.com/dl/files/040/90885040/att_970576/v1/81318674_Programming_guideline_DOC_v16_en.pdf. [cit. 2024-05-19].

[2] SIEMENS. STEP 7 and WinCC Engineering V17. Online. 2021. Dostupné z: https://cache.industry.siemens.com/dl/files/671/109798671/att_1071920/v1/STEP_7_WinCC_V17_enUS_en-US.pdf. [cit. 2024-05-19].

[3] SIEMENS. TIA Portal Openness: API for automation of engineering workflows. Online. 2021. Dostupné z: https://cache.industry.siemens.com/dl/files/533/109798533/att_1069908/v1/TIAPortalOpennessenUS_en-US.pdf. [cit. 2024-05-19].

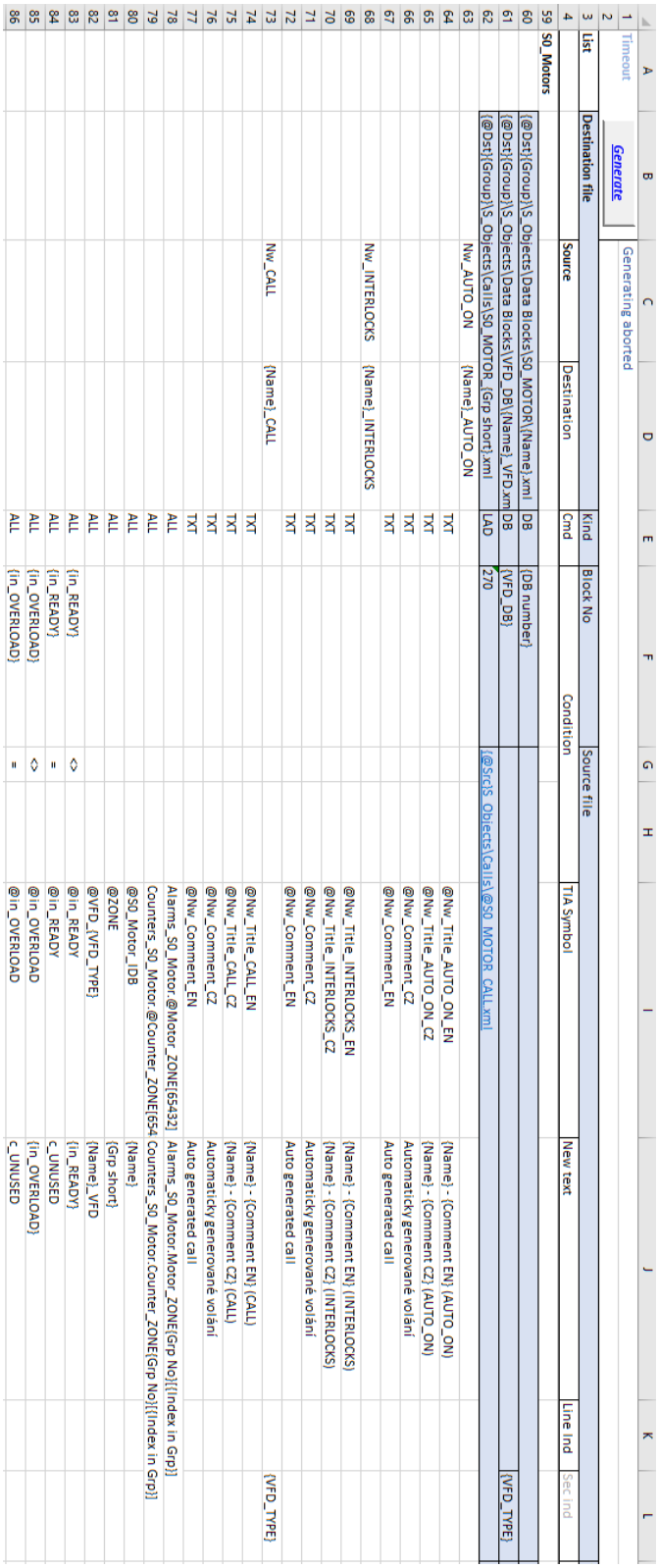
[4] SIEMENS. TIA Portal Openness: Introduction and demo application. Online. 2023. Dostupné z: https://cache.industry.siemens.com/dl/files/692/108716692/att_1131753/v2/108716692_TIA_PortalOpenness_GettingStartedAndDemo_V17_en.pdf. [cit. 2024-05-19].

[5] AMAZON. What is XML? Online. © 2024. Dostupné z: https://aws.amazon.com/what-is/xml/. [cit. 2024-05-19].

[6] SIEMENS. TIA Portal Openness: Referencing the Siemens.Engineering.dlls and Assembly Resolve. Online. 2023. Dostupné z: https://cache.industry.siemens.com/dl/files/895/109815895/att_1127786/v3/109815895_AssemblyResolve_V1_0_EN.pdf. [cit. 2024-05-21].

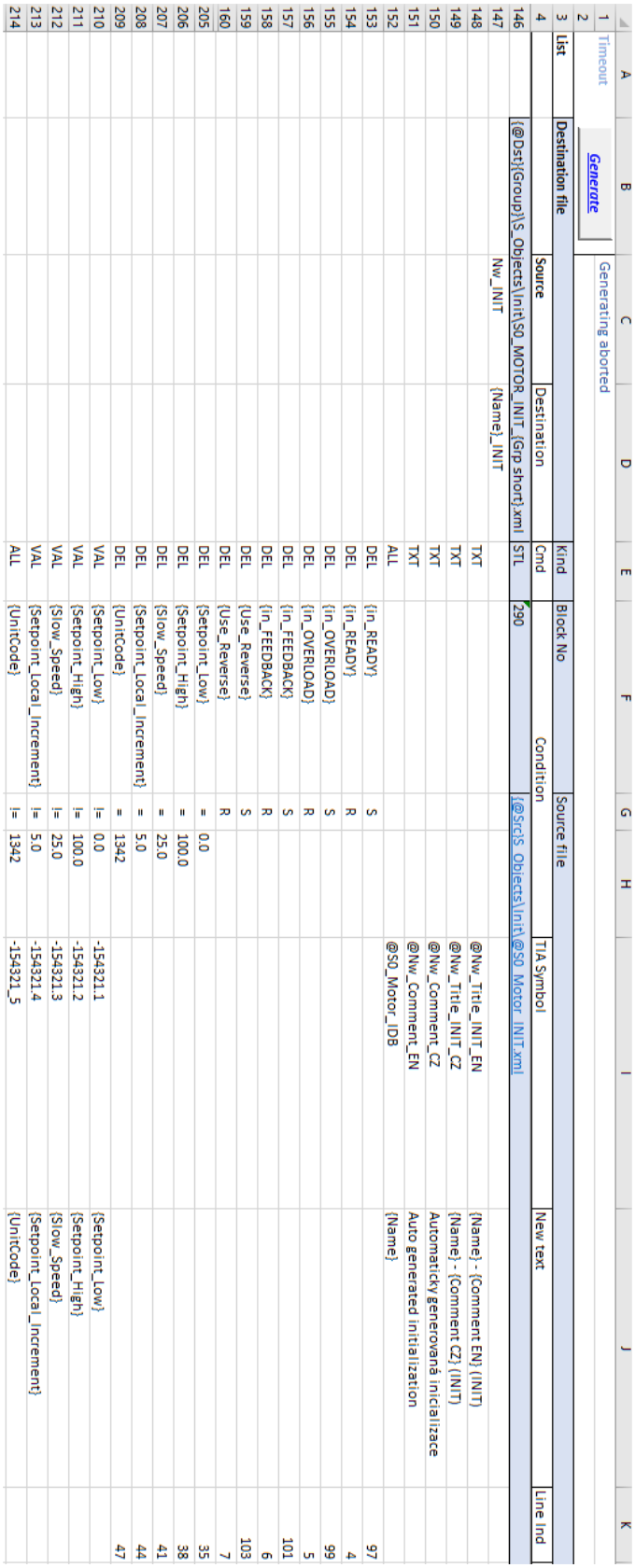
7. Přílohy

A - Šablona generátoru pro volání instance motoru



	A	B	C	D	E	F	G	H	I	J	K	L
1	Timeout	Generate	Generating aborted									
2												
3	List	Destination file			Kind	Block No	Source file					
4		Source	Destination		Cmd	Condition		TIA Symbol	New text	Line Ind	Sec Ind	
59	S0_Motors											
60		{@Dst}{Group}\S_Objects\Data Blocks\S0_MOTOR\{Name}.xml			DB	{DB number}						
61		{@Dst}{Group}\S_Objects\Data Blocks\VFD_DB\{Name}_VFD.xml			DB	{VFD_DB}						{VFD_TYPE}
62		{@Dst}{Group}\S_Objects\Calls\S0_MOTOR_{Grp short}.xml			LAD	270	{@Src}S_Objects\Calls\@S0_MOTOR_CALL.xml					
63			Nw_AUTO_ON	{Name}_AUTO_ON								
64					TXT			@Nw_Title_AUTO_ON_EN	{Name} - {Comment EN} (AUTO_ON)			
65					TXT			@Nw_Title_AUTO_ON_CZ	{Name} - {Comment CZ} (AUTO_ON)			
66					TXT			@Nw_Comment_CZ	Automaticky generované volání			
67					TXT			@Nw_Comment_EN	Auto generated call			
68			Nw_INTERLOCKS	{Name}_INTERLOCKS								
69					TXT			@Nw_Title_INTERLOCKS_EN	{Name} - {Comment EN} (INTERLOCKS)			
70					TXT			@Nw_Title_INTERLOCKS_CZ	{Name} - {Comment CZ} (INTERLOCKS)			
71					TXT			@Nw_Comment_CZ	Automaticky generované volání			
72					TXT			@Nw_Comment_EN	Auto generated call			
73			Nw_CALL	{Name}_CALL								{VFD_TYPE}
74					TXT			@Nw_Title_CALL_EN	{Name} - {Comment EN} (CALL)			
75					TXT			@Nw_Title_CALL_CZ	{Name} - {Comment CZ} (CALL)			
76					TXT			@Nw_Comment_CZ	Automaticky generované volání			
77					TXT			@Nw_Comment_EN	Auto generated call			
78					ALL			Alarms_S0_Motor.@Motor_ZONE[65432]	Alarms_S0_Motor.Motor_ZONE[Grp No][{index in Grp}]			
79					ALL			Counters_S0_Motor.@Counter_ZONE[654	Counters_S0_Motor.Counter_ZONE[Grp No][{index in Grp}]			
80					ALL			@S0_Motor_IDB	{Name}			
81					ALL			@ZONE	{Grp short}			
82					ALL			@VFD_{VFD_TYPE}	{Name}_VFD			
83					ALL	{in_READY}	<>	@in_READY	{in_READY}			
84					ALL	{in_READY}	=	@in_READY	c_UNUSED			
85					ALL	{in_OVERLOAD}	<>	@in_OVERLOAD	{in_OVERLOAD}			
86					ALL	{in_OVERLOAD}	=	@in_OVERLOAD	c_UNUSED			

B – Šablona Generátoru pro inicializaci instance motoru



	A	B	C	D	E	F	G	H	I	J	K
1	Timeout		Generating aborted								
2		Generate									
3	List	Destination file			Kind	Block No	Source file				
4		Source	Destination		Cmd	Condition		TIA Symbol	New text		Line Ind
146		{@Dst}{Group}\S_Objects\Init\S0_MOTOR_INIT_{Grp short}.xml			STL	290	{@SrcIS_Objects\Init\@S0_Motor_INIT.xml				
147		Nw_INIT	{Name}_INIT								
148					TXT			@Nw_Title_INIT_EN	{Name} - {Comment EN} (INIT)		
149					TXT			@Nw_Title_INIT_CZ	{Name} - {Comment CZ} (INIT)		
150					TXT			@Nw_Comment_CZ	Automaticky generovaná inicializace		
151					TXT			@Nw_Comment_EN	Auto generated initialization		
152					ALL			@S0_Motor_IDB	{Name}		
153					DEL	{in_READY}	S				97
154					DEL	{in_READY}	R				4
155					DEL	{in_OVERLOAD}	S				99
156					DEL	{in_OVERLOAD}	R				5
157					DEL	{in_FEEDBACK}	S				101
158					DEL	{in_FEEDBACK}	R				6
159					DEL	{Use_Reverse}	S				103
160					DEL	{Use_Reverse}	R				7
205					DEL	{Setpoint_Low}	= 0.0				35
206					DEL	{Setpoint_High}	= 100.0				38
207					DEL	{Slow_Speed}	= 25.0				41
208					DEL	{Setpoint_Local_Increment}	= 5.0				44
209					DEL	{UnitCode}	= 1342				47
210					VAL	{Setpoint_Low}	!= 0.0	-154211.1	{Setpoint_Low}		
211					VAL	{Setpoint_High}	!= 100.0	-154211.2	{Setpoint_High}		
212					VAL	{Slow_Speed}	!= 25.0	-154211.3	{Slow_Speed}		
213					VAL	{Setpoint_Local_Increment}	!= 5.0	-154211.4	{Setpoint_Local_Increment}		
214					ALL	{UnitCode}	!= 1342	-154211_5	{UnitCode}		

C – Odkaz na video demonstrující funkci projektu

https://youtu.be/XjjQpiqAXPE