CHALMERS

TIA AutoGen @ Granitor

FB from Source	FB from Library	FC from Source
FB_AnalogSignal_1	FB_AnalogSignalSimple	FC_Analog
FB_ValveDigital_StepOpen	FB_DigitalSignalSimple	FC_Extra_sekvenser_Kok
FB_SequenceKok	FB_ScaleSP_4Pumps	FC_Kok
FB_ValveAnalog	FB_PumpSwitch_4Pumps	FC_Larmsandare
FB_ValveDigital	FB_PumpMotor	
FB_TempToSpeed	FB_GenericDriveSimple	
FB_MotorAnalog	FB_RunTime	
FB_FanPN_CPU1200	FB_ValveControl_B	
FB_BoolToWord	FB_ValveB	
	FB_Motor2speed	
	FB_ScaleSP_3Pumps	
	FB_PumpSwitch_3Pumps	
	Scale3To1	
	FB_ControlObject	

Autogenerering av PLC-kod

Effektivisering av PLC-programmering med hjälp av Siemens TIA Portal Openness

Examensarbete inom högskoleprogrammet Mekatronik

ERIK MÜLLER
SAMUEL ROSÉN

INSTITUTIONEN FÖR ELEKTROTEKNIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Examensarbete 2025

Autogenerering av PLC-kod

Effektivisering av PLC-programmering med hjälp av
Siemens TIA Portal Openness

ERIK MÜLLER
SAMUEL ROSÉN

CHALMERS

Institutionen för elektroteknik
Chalmers tekniska högskola
Göteborg 2025

Autogenerering av PLC-kod
Effektivisering av PLC-programmering med hjälp av Siemens TIA Portal Openness
ERIK MÜLLER SAMUEL ROSÉN

Handledare: John Tagesson Ritz, Granitor Systems AB
Examinator: Veronica Olesen, Elektroteknik

Examensarbete 2025
Institutionen för elektroteknik
Chalmers tekniska högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: En skärmdump av användargränssnittet i Excel för programmet, som ger en översikt över tillgängliga funktioner i ett projekt.

Skriven i LaTeX
Göteborg 2025

Förord

Denna rapport är ett examensarbete inom mekatronik vid institutionen för elektroteknik på Chalmers tekniska högskola. Projektet utgör den avslutande delen av utbildningen.

Vi vill rikta ett stort tack till Granitor för möjligheten att genomföra projektet hos dem. Ett särskilt tack riktas till vår handledare John och till de kollegor som stöttat oss under arbetets gång. Vi hoppas att detta arbete kan bidra till Granitors fortsatta utveckling och önskar företaget all framgång framöver.

Slutligen vill vi ge ett stort tack till vår examinator på Chalmers, Veronica Olesen, som väglett oss genom projektet, särskilt vid rapportskrivningen.

Erik Müller, Samuel Rosén, Göteborg, juni 2025

Sammanfattning

PLC-programmering är ofta en tidskrävande och repetitiv process där programmeraren manuellt implementerar återkommande funktioner, vilket leder till ineffektivitet och ökad risk för fel. Det här projektet undersöker hur delar av denna process kan automatiseras för att minska det manuella arbetet och förbättra noggrannheten. Med hjälp av TIA Portal Openness och Python, i kombination med flera bibliotek och XML-hantering, har en applikation utvecklats som automatiskt genererar en programstruktur utifrån ett befintligt mallprojekt och användardefinierad data. Rapporten innehåller även erfarenheter kring TIA Portal Openness, vilket kan vara användbart vid utveckling av program som kommunicerar med eller automatiserar funktioner i TIA Portal.

Nyckelord: TIA Portal, TIA Portal Openness, PLC, Python, XML, autogenerering

Abstract

PLC programming often involves repetitive manual tasks, which can result in inefficiencies and an increased risk of errors. This project aims to automate parts of the programming process using TIA Portal Openness and Python. The outcome is an application that, with the help of XML handling and external libraries, can generate a program structure based on a predefined template project and user-defined input data. The report also provides practical insights into working with TIA Portal Openness, which may be valuable for the future development of automation tools in TIA Portal.

Keywords: TIA Portal, TIA Portal Openness, PLC, Python, XML, autogeneration

Innehåll

Beteckningar

Nedan följer en lista över akronymer som har använts genom hela denna rapport, listade i alfabetisk ordning:

API	Application Programming Interface
BS4	Beautiful Soup 4
DB	Data Block
FB	Function Block
FBD	Function Block Diagram
FC	Function
HMI	Human-Machine Interface
IDE	Integrated Development Environment
OB	Organization Block
OS	Operativsystem
PLC	Programmable Logic Controller
pip	Pip Installs Packages
PyPI	Python Package Index
SFC	Sequential Function Chart
TIA	Totally Integrated Automation
TS	TIA Scripting
XML	Extensible Markup Language

Figurer

Tabeller

1

Inledning

1.1 Bakgrund

I takt med att automation blir allt mer central inom industriell produktion ökar också behovet av effektiva och pålitliga utvecklingsmetoder. Programmable Logic Controllers (PLC:er) används för att styra maskiner och processer inom till exempel tillverkningsindustri, infrastruktur och energisystem. De möjliggör logisk styrning av allt från transportband och pumpar till trafiksystem och ventilationsanläggningar. PLC-programmering är ofta repetitiv och tidskrävande, vilket ökar risken för fel och ineffektivitet.

En stor del av företaget Granitor Systems arbete består av styrsystem inom bland annat infrastruktur och industriell produktion som ofta förlitar sig på programmatiska lösningar, där PLC spelar en central roll. De använder sig i stor utsträckning av PLC från leverantören Siemens som programmeras i Siemens Totally Integrated Automation Portal (TIA).

Till TIA Portal har Siemens utvecklat applikationsprogrammeringsgränssnittet TIA Portal Openness, vilket används för att skapa egna applikationer för att kunna automatisera processer i TIA Portal. Exempelvis möjliggör det för användaren att kunna generera färdiga programstrukturer och kod.

Genom att använda TIA Portal Openness, ett gränssnitt för att automatisera arbete i Siemens TIA Portal, kan delar av programmeringsarbetet effektiviseras. Detta examensarbete fokuserar på hur automatisering av programstruktur med hjälp av funktionsbibliotek och användardata kan förbättra utvecklingsprocessen.

1.2 Syfte

Projektet syftar till att undersöka möjligheterna att eliminera repetitiva moment inom PLC-programmering. Detta för att spara tid och effektivisera programmerarens uppgift genom att minska monotona moment i programmeringen, där svårupptäckta fel kan uppstå.

1.3 Mål

Målet med detta projekt är att utveckla en applikation som automatiskt genererar PLC-programstruktur utifrån användardata, med hjälp av TIA Portal Openness. Avsikten är att applikationen ska bestå av ett användarvänligt gränssnitt i Excel för datainmatning. För att kunna köra programmet är avsikten att ett separat gränssnitt ska behöva utvecklas för att hantera inställningar och styrning av applikationen.

1.4 Avgränsningar

Fokus kommer att ligga på att skapa en fungerande prototyp snarare än ett fullt implementerat system. Eventuella integrationer med andra programvarusystem eller PLC-plattformar kommer inte att omfattas av projektet. Projektet genomförs under en begränsad tidsperiod (våren 2025).

1.5 Hållbar utveckling i projektet

Projektets koppling till social, ekonomisk och ekologisk hållbarhet tas i detta delavsnitt upp, samt hur den hållbara utvecklingen påverkas av automation.

1.5.1 Ekonomisk hållbarhet

Implementering av automationslösningar kan leda till minskat behov av manuellt arbete i projekten som leverantörer genomför, vilket i förlängningen kan sänka kostnaderna [1]. När en större del av processen automatiseras kan resurser effektiviseras, vilket potentiellt kan bidra till ökad kundnöjdhet och stärkt konkurrenskraft. Detta kan i sin tur ha positiv inverkan på företagens lönsamhet och, på längre sikt, även gynna samhällsekonomin.

1.5.2 Social hållbarhet

Sett ur social hållbarhet, så kan automatiseringen av repetitiva arbetsuppgifter leda till mindre mänskliga fel och på så sätt undvika farliga situationer som kan uppstå på grund av detta. Indirekt kan en minskning av repetitiva arbetsuppgifter bidra till bättre välmående bland programmerare eftersom de då får möjlighet att fokusera på mer kreativa och utvecklande uppgifter istället för monotona moment. Detta kan i sin tur minska stress och öka arbetsglädjen.

1.5.3 Ekologisk hållbarhet

Den ekologiska hållbarheten är svårare att direkt koppla till projektet. En viss påverkan kan däremot identifieras ur ett bredare perspektiv. Genom att autogenerera komplicerad kod kan behovet av manuellt underhåll minska, vilket potentiellt leder till mer resurseffektiva lösningar över tid. Det kan även minska utvecklingstiden och därmed indirekt bidra till ett lägre klimatavtryck genom effektivare användning av energi och andra resurser. Dessutom kan effektivare verktyg sänka tröskeln för att uppgradera äldre styrsystem, vilket i sin tur kan främja implementeringen av modernare och mer energieffektiva lösningar.

2
Teknisk bakgrund

I denna del av rapporten har relevant fakta sammanställts och hämtats för att skapa en förståelse för de nödvändiga faktorerna som krävs för att genomföra projektet samt för att dra slutsatser om det. De områden som har studerats är programmering, automation och de programvaror som använts under projektet.

2.1 Application Programming Interface (API)

Det underliggande konceptet för ett Application Programming Interface (API) är att det utgör ett gränssnitt som möjliggör kommunikation mellan olika mjukvarusystem [2]. Ett API fungerar som en mellanhand genom vilken applikationer kan skicka och ta emot data, utan att behöva känna till den exakta implementeringen av den andra applikationen. API:er tillhandahålls ofta som bibliotek eller moduler för att underlätta återanvändning och förenkla integration i olika programvaror.

2.2 Extensible Markup Language (XML)

Ett märkspråk (markup language) är ett språk som används för att strukturera upp och beskriva textfiler med hjälp av taggar/attribut [3]. Programmeringsspråket Extensible Markup Language (XML) är ett exempel på en typ av märkspråk. De främsta egenskaperna hos XML är att det inte är beroende av någon specifik programvara, samt att det skrivs i vanlig text vilket underlättar delningen mellan enheter och program.

Vidare har språket en del likheter med det populära märkspråket HyperText Markup Language (HTML) som vanligtvis används till webbutveckling. Skillnaden mellan språken ligger i att XML lägger till information om vilken data taggarna innehåller, och taggarna hos HTML innehåller information om format och utseende på texten.

2.3 Programmerbar styrenhet (PLC)

PLC är förkortningen för Programmable Logic Controller vilket används inom industrin för att automatisera maskiner och robotar [4]. Det är samspelet mellan den fysiska styrutrustningen (hårdvaran) och den programmerbara logiken (mjukvaran) som möjliggör ett fungerande styrsystem. Hårdvaran är uppbyggd av själva PLC-enheten samt ingångar och utgångar för att möjliggöra kommunikation med externa

komponenter. Ingångarna används för att ta emot signaler från externa sensorer och andra enheter. Utgångarna används oftast för att styra bland annat motorer och ventiler.

Programvaran som körs på PLC-enheten används för att kunna styra PLC:n och för att skapa styralgoritmer och logik. PLC kan användas för mer komplexa system såsom produktionslinjer inne på fabriker men även enklare konstruktioner såsom transportband. Systemet brukar användas inom områden där mänskliga fel kan skapa mycket stora problem, vilket kärnkraft kan vara ett bra exempel på [5].

Vilka språk som används för att programmera en PLC kan variera, två vanliga är funktionsdiagram (SFC) samt funktionsblocksdiagram (FBD) [6]. SFC används för att styra sekventiella processer där olika steg och övergångar representeras som ett typ av diagram. Styrlogiken delas upp i steg, med en eller flera åtgärder som utgör varje fas. Dessa stadier representeras oftast av sammankopplade rektanglar, med pilar som anger övergångarna mellan stegen. SFC representerar styrsekvensen på ett distinkt och strukturerat sätt vilket underlättar vid analys och utveckling av komplexa styrsystem.

I FBD byggs funktioner upp och representeras av olika block och symboler. Principen med språket baseras på att representera den avsedda styrlogiken genom att koppla samman flera funktionselement. Språket erbjuder en grafisk representation av styrlogiken där logiska block, timers och in- och utsignaler alla kan representeras av funktionsblock.

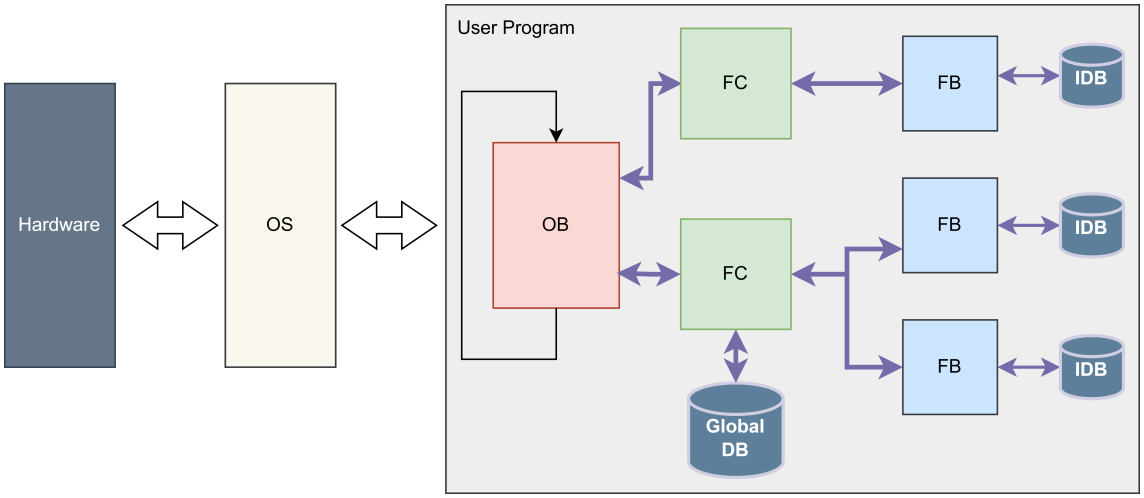
2.4 Totally Integrated Automation Portal (TIA Portal)

Siemens Totally Integrated Automation Portal (TIA-portalen) är en mjukvara från Siemens som används vid automationsprojekt [7]. Mjukvaran integrerar och omstrukturerar ett flertal av deras tidigare programvaror och miljöer. Dessa miljöer används bland annat till att konfigurera och programmera Siemens PLC:er och Human-Machine Interface-paneler (HMI). Programmeringslogiken för mjukvaran följer en blockstruktur, vilket underlättar arbetet med att utveckla och felsöka industriella processer när det görs på ett strukturerat och organiserat sätt.

Ytterligare fler funktioner med TIA-Portalen är att programmet tillåter redigering genom drag-and-drop, gemensam datahantering och delning av projekt. Mjukvarubibliotek kan även skapas för att återanvända delar av projekt i andra projekt. Ändringar som görs, exempelvis av variabler och datatyper, kan förekomma på flera ställen i mjukvaran och tillämpas därefter på hela projektet.

2.4.1 Programflöde och blockstruktur i Siemens TIA-portal

I Siemens TIA Portal är användarprogrammet strukturerat i olika typer av block som tillsammans styr PLC:ns beteende [8]. En PLC har ett inbyggt operativsystem, inte att förväxla med ett datoroperativsystem, som hanterar interna funktioner som minnesallokering, kommunikation och exekveringscykler. Detta operativsystem ansvarar även för att initiera körningen av användarens programkod i rätt ordning. Användaren programmerar applikationsspecifika delar som utför logisk styrning, till exempel övervakning av sensorer eller styrning av motorer. Figur 2.1 illustrerar hur användarprogrammet struktureras i TIA Portal med olika blocktyper och deras relation till operativsystemet (OS).



Figur 2.1: Exempel på programstruktur för TIA Portal.

Centralt i programstrukturen finns Organization Blocks (OB), markerat i rött i figuren, det fungerar som ett gränssnitt mellan användarprogrammet och operativsystemet. Ett projekt kan ha flera OB som triggas på olika saker. Ett exempel är Cyclic Interrupt när OB kan köras med fasta tidsintervall och används för tidskritiska funktioner. Dessa olika OB får då en numrering som används som referens inuti TIA Portal. Normalt används OB1 som ett huvudblock som körs cykliskt och styr sekvensen för blockanropen i användarkoden.

För att dela upp funktionalitet i mindre delar används Functions (FC), markerade i grönt i figuren, och Function Blocks (FB), markerade i blått. Båda dessa block innehåller kod som kan återanvändas i olika delar av programmet.

FC är enklare funktioner där alla värden matas in och returneras utan minneslagring mellan anrop. FB, däremot, har möjlighet till minnesbevarande variabler genom en koppling till ett Instance Data Block (IDB i Figuren), som lagrar data för varje unikt anrop av blocket. Dessa IDB skapas samtidigt som FB, där varje FB har en individuell referens till sitt IDB, vilket gör att FB kan hantera tillstånd över tid.

Det går även att skapa globala datablock i form av Data Block (DB), vilket möjliggör datalagring även för ett FC enligt Figur 2.1. Ett globalt DB kan nås från hela

användarprogrammet och används ofta för att hantera gemensam information. Till skillnad från IDB:er är dessa block inte bundna till någon enskild FB, utan fungerar fristående.
Alla dessa olika sätt att programmera och lagra data finns samlade under mappen `Program blocks` i TIA Portal.

Figur 2.1 visar en slags hierarkisk ordning mellan de olika blocktyperna. Det är dock inte ett måste att använda FB i FC:n och FC i OB:n. Användningen av blocken beror istället på vad användaren vill få ut av programmet. Det betyder bland annat att FB:n kan bli kallade direkt av OB:n. Detta möjliggör en mycket flexibel programmering utefter vad situationen kräver.

2.5 TIA Portal Openness

TIA Portal Openness är ett Application Programming Interface (API) som tillåter programmerare att själva skapa egna applikationer med önskad funktionalitet [9]. TIA Portal Openness använder sig av Dynamic-link library (DLL:er) för att få åtkomst till de funktioner och objekt som TIA Portal använder sig av. Kommandon skickas vidare till TIA Portal från Openness, vilket fungerar som ett gränssnitt som använder mjukvarubibliotek med funktionalitet från TIA Portalen.

Vidare möjliggör TIA Portal Openness skapandet av applikationer med hjälp av programmeringsspråket C#. I grunden används API:et för att kontrollera TIA Portal externt och på så sätt automatisera moment inuti TIA Portal. Dock kan Openness enbart utföra ett antal förutbestämda funktioner inuti TIA Portal. Dessa funktioner begränsar vad som är möjligt att genomföra. Projektdata kan dock exporteras och importeras i XML-format via API:n.

2.5.1 Dokumentation kring TIA Portal Openness

Siemens har en onlineportal där de lägger upp all dokumentation kring TIA Portal Openness [10]. Där finns bland annat manualer för API:erna, en hel del exempelprogram och verktyg som använder Openness. Ett urval av programmen och verktygen beskrivs i kommande avsnitt.

Vid projektets uppstart var endast manualen för TIA Portal Openness V19 tillgänglig, trots att TIA Portal V20 funnits tillgänglig sedan november 2024 [11]. Därför har TIA Portal V19 använts under projektets utveckling.

2.5.2 Uppbyggnad av XML-filer

XML-filerna är en viktig byggsten relaterad till hur TIA Portal Openness arbetar. Genom att förstå hur en XML-fil är uppbyggd kan den manipuleras och ändras utanför TIA. Varje typ av funktionsblock i TIA har en egen typ av XML-fil som bestäms av en av de första taggarna i dokumentet [12]. Figur 2.2 nedan är hämtad från ett FC. Först kommer taggen `<Document>`, som är rotnoden för hela dokumentet,

och avslutas med motsvarande `</Document>`. I rotnoden placeras all information gällande dokumentet. Först i rotnoden kommer vilken version av TIA som använts genom `<Engineering version="V19"/>` med attributen för vilken version av TIA Portal som har skapat filen. Taggen `<SW.Blocks.FC ID="0">` bestämmer vilken typ av block det är i TIA Portal (i detta fall FC) samt ID, som är en hexadecimal lokal numrering för de olika elementen i dokumentet. För att korrekt importera en XML-fil i TIA Portal är det viktigt att numreringen för `ID` är korrekt och kommer i numerisk ordning. En XML-fil kan även innehålla en annan typ av `<SW.Blocks` i form av `<SW.Blocks.CompileUnit ID="3"CompositionName="CompileUnits">`, dessa visar att det kommer ett compile unit block, vilket innehåller data för en sammanhängande del. Det skulle exempelvis kunna representera ett `Network`. Varje `<SW.Blocks.XXX ID="XX">`-tagg avslutas med tillhörande `</SW.Blocks.XXX>`.

```
▼<Document>
   <Engineering version="V19"/>
  ▶<SW.Blocks.FC ID="0">
   ...
   </SW.Blocks.FC>
 </Document>
```

Figur 2.2: En XML-fil som innehåller kod för ett FC block.

2.6 Färdiga verktyg till TIA Portal Opennes

I detta avsnitt presenteras en rad olika verktyg baserade på TIA Portal Openness som används under projektet eller gett inspiration för vidare utveckling.

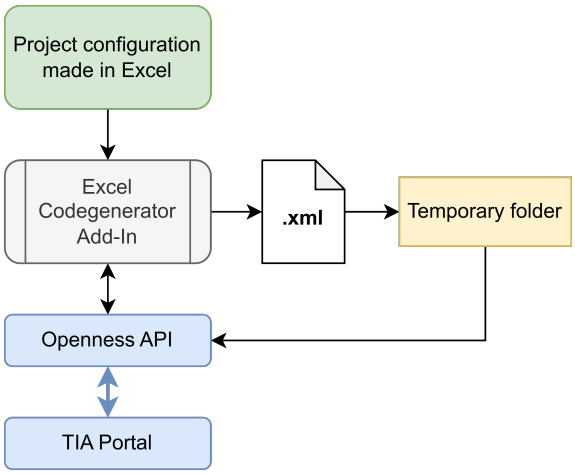
2.6.1 TIA Portal Openness Explorer

TIA Portal Openness Explorer är enbart utvecklad för att ge programmeraren en överblick av API:n och är inte tänkt att lämpas för dagligt arbete [13]. Genom att ha igång ett öppet PLC-projekt i TIA Portal kan TIA Portal Openness Explorer ansluta till det öppna projektet och tillåta användaren att navigera in i projektet grafiskt. Detta sker likt hur man hade programmerat via Openness för att komma åt samma innehåll. Genom att öppna ett existerande TIA Portal kan de olika programblocken hittas, och i sin tur exporteras som XML. Varje OB, FC, FB och DB har således en tillhörande XML-fil som går att få ut. Dessa XML-filer innehåller logik, referenser och kan innehålla flera tusen rader kod. Just TIA Portal Openness Explorer kan enbart läsa ut data, men saknar funktionalitet för att stoppa in data.

2.6.2 Excel Codegenerator av Siemens

Siemens har utvecklat ett eget exempel på hur autogenerering av kod skulle kunna se ut via Excel, där användaren kan fylla i ett Excel-dokument med information som sedan importeras och resulterar i ett program med programmerad kod [14]. Tyvärr är inte källkoden till exemplet publicerad och dokumentationen som finns visar

enbart hur användaren ska använda verktyget som är väldigt låst till ett specifikt arbetssätt. Dock har Siemens publicerat ett enkelt flödesschema på hur verktyget interagerar med TIA Portal genom Openness. Flödesschemat visar hur ett program måste byggas upp som tar in Excel-filen, gör om det till XML-filer som i sin tur via Openness API importeras in i TIA Portal. Detta demonstreras i Figur 2.3 nedan som är en omgjord förenkling av Siemens publicerade flödesschema.



Figur 2.3: Flödesschema hur Siemens Excel Codegenerator fungerar och generar XML-filer.

2.6.3 TIA Scripting Python

TIA Scripting Python är ett Python-bibliotek som används för att interagera med TIA Openness-gränssnittet genom skriven kod i Python [15]. Biblioteket lanserades i november 2024 och använder ett enkelt skriptspråk för att automatisera enkla uppgifter i TIA-portalen, därmed undviks kravet på att användaren kan avancerad programmering. Den förenklar vissa grundläggande funktioner, bland annat uppgifter som att exportera och importera data samt kompilera programblock. Modulen underlättar för användare som inte har så mycket erfarenhet av TIA Openness. Inkluderat med dokumentationen finns det bland annat två stycken Python-filer. En för att exportera ut data från PLC:n i ett TIA-projekt och ett program för att importera data in i ett TIA-projekt.

2.7 GitHub

GitHub är en molnplattform för hantering av kod [16]. GitHub använder sig av GIT, ett distribuerat versionshanteringssystem. Plattformen erbjuder versionshantering, vilket gör det möjligt för användare att gå tillbaka till tidigare versioner. En annan möjlighet med plattformen är det omfattande samarbetet mellan olika programmerare. Där flera användare kan vara med och samarbeta i samma projekt.

2.8 PyCharm

PyCharm är en Integrated Development Environment (IDE) för programmering i Python, IDE:n är utvecklad av JetBrains [17]. Funktioner som erbjuds av plattformen är kodanalys, felsökning, testning och versionshantering. Användning av tjänsten GitHub är möjlig direkt från IDE:n. Programmet finns i flera versioner, där den avancerade Professional Edition har stöd för fler tekniker och språk, medan Community Edition är en enklare och kostnadsfri version.

2.9 Python bibliotek

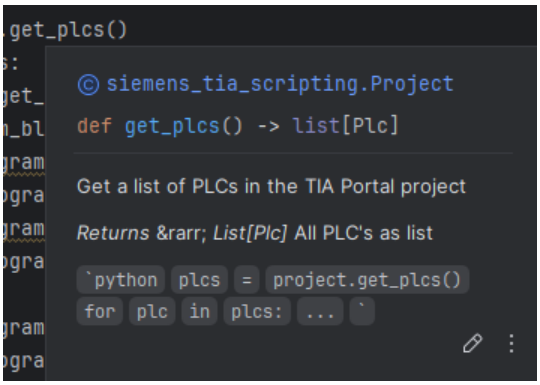
För att effektivisera programmering i Python finns det tillgängliga bibliotek [18]. Dessa bibliotek ger en samling av fördefinierade funktioner och kod för att lösa specifika problem. Biblioteken (eller paket som de ibland kan kallas) installeras oftast via Pip Installs Packages (`pip`) och Python Package Index (PyPI) där PyPI är en central databas för Python-bibliotek.

Ibland kan upphovsmän vilja distribuera paket eller bibliotek med licensrestriktioner och inte publikt via PyPI. Detta kan ske med hjälp av en wheel-fil som betecknas med filändelsen `.whl`. En `.whl`-fil innehåller alla nödvändiga filer för att använda biblioteket i ett förkompilerat tillstånd. Detta innebär att användaren inte behöver kompilera koden på en egen dator, vilket gör installationen snabbare och enklare. Dock kan en `.whl`-fil vara versionsberoende, vilket innebär att en viss version av Python måste användas. Ett exempel på när det kan bli problematiskt är när en wheel-fil är gjord för Python 3.9.x trots att en nyare version av Python finns tillgänglig.

2.9.1 Installera TIA Scripting Python

TIA Scripting Python (TS) är enbart tillgängligt som en wheel-fil via Siemens supportsida, där både filen och dokumentationen kräver att man är inloggad med ett registrerat Siemens-konto (Siemens ID Account) [19]. För att kunna använda senaste släppta versionen av TS, (V 1.07) är det viktigt att Python 3.12.x används. Det är möjligt att framtida versioner av TS kommer att stödja senare versioner av Python, men i dagsläget fungerar endast Python 3.12.x.

För att använda TS skulle det räcka med att peka på var whl-filen är placerad och använda den den vägen. Dock rekommenderas det att installera whl-filen som ett Python-paket, för att få tillgång till så kallad Intellisense-support. Detta innebär att utvecklingsmiljön kan ge förslag, autokomplettera kod och visa dokumentation direkt medan man skriver, som visas i Figur 2.4. Installation sker genom `pip install C:/some-dir/name-of-TIA-Scripting.whl` i projektets terminalfönster. Viktigt är att specificera var whl-filen finns och dess exakta namn. En mer detaljerad beskrivning finns dokumenterad i bilaga B.



Figur 2.4: TS installerat i PyCham med Intellisense support, vilket möjliggör att se dokumentation direkt från PyCharm.

2.9.2 Beautiful Soup 4 (BS4)

Beautiful Soup 4 (BS4) är ett Python-bibliotek som används för att läsa in och bearbeta XML- och HTML-dokument [20]. Biblioteket kan användas effektivt även när XML- eller HTML-koden är slarvigt strukturerad och svår att analysera manuellt. Biblioteket skapar ett träd av strukturen på ett dokument, vilket underlättar extraktionen av information. Användningen är vanligt förekommande vid så kallat 'web scraping', där det används för att extrahera och bearbeta data från webbsidor. Beautiful Soup lanserades 2004 av Leonard Richardson och underhålls fortfarande med nya versioner, där den senaste versionen är 4.13.4.

2.9.3 Openpyxl

Openpyxl är ett Pythonbibliotek som används för att hantera Excel-filer i ett flertal format, exempelvis `.xlsx` [21]. Biblioteket innehåller funktioner för att skriva till och läsa från filer, och stödjer dessutom flera versioner av Excel. Ändringar i filer kan göras i stor utsträckning. Exempel på funktioner är formatering av celler, skapande av diagram samt hantering av formler. Programmet fungerar utan att Excel behöver vara installerat, vilket möjliggör ett bredare användningsområde.

2.9.4 Questionary

Questionary är ett Python-bibliotek som används för att skapa interaktiva kommandoradsgränssnitt [22]. Det erbjuder olika typer av inmatningspromptar, såsom textinmatning, val från listor samt ja/nej-frågor. Biblioteket underlättar användarinteraktion i terminalbaserade program genom att tillhandahålla en enkel metod för att hämta och bearbeta användarinput. Questionary är öppen källkod och licensierat under MIT-licensen.

För att få Questionary att fungera korrekt i PyCharm kan det krävas att vissa inställningar ändras på grund av hur PyCharm hanterar terminalen och interaktiva inmatningar. Detta är dokumenterat i bilaga B.

3
Metod

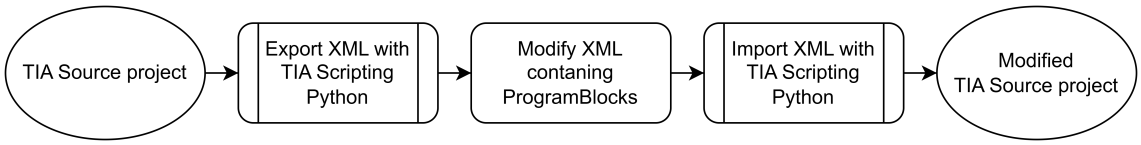
I detta kapitel beskrivs den metod som legat till grund för genomförandet av projektet. För att ge en tydlig översikt har avsnittet delats in i flera delkapitel, där varje del behandlar ett specifikt steg i projektets arbetsprocess.

3.1 Övergripande metod

Projektet inleddes med en undersökning av möjliga kopplingar till programmeringsverktyg, samt en genomgång av nuvarande metoder inom PLC-programmering för att identifiera områden där automatisering och effektivisering är möjliga.

Genom att utföra en omfattande analys av Siemens TIA Portal Openness, Siemens eget API för att automatisera olika moment i TIA Portal gavs kunskap kring hur applikationen skulle kunna struktureras. Detta uppnåddes genom att studera dokumentation, exempelprogram och exempelkod. På så sätt kunde en bild fås av vad som är praktiskt genomförbart med hjälp av API:n. Det fanns även ett behov av att utforska TIA Portal som verktyg, för att förstå hur ett projekt är uppbyggt i TIA Portal och förstå hur de olika delarna i ett projekt samverkar med varandra.

Efter genomgång av dokumentation och praktisk utvärdering av olika program identifierades det att generering av PLC-kod kräver att `Program blocks` exporteras i form av XML-filer. För att generera kod krävs det att dessa XML-filer manipuleras och ändras innan de importeras tillbaka till TIA Portal. Flödet illustreras i Figur 3.1 och representerar den övergripande strukturen för hur applikationen skulle utformas.



Figur 3.1: Struktur för utformning av applikationen.

I Siemens användarmanual för TIA Openness, särskilt för TIA Scripting Python, framkommer det att skript för Python är mer användarvänliga än skript som använder C# [23]. Manualen nämner ett flertal fördelar gentemot att använda standard Openness som skrivs i C#, bland annat flera Python-bibliotek som kan användas. Automatisk hantering av olika versioner på TIA-portalen och standardiserade funktioner för Python, vilket minskar storleken på koden.

Mot bakgrund av detta beslutades att använda TIA Scripting Python, då det bedömdes förenkla programmeringen och erbjuda goda möjligheter till återanvändning av kod genom exempelsamlingar för import och export. Under utvecklingsarbetet används TIA Portal V19 tillsammans med olika exempelprojekt för att testa och validera funktionaliteten i det egna programmet.

3.2 Integrated development environments (IDE)

PyCharm användes som utvecklingsmiljö med hänsyn till att Python valdes som programmeringsspråk. PyCharm är ett populärt val bland både erfarna och oerfarna programmerare tack vare sina kraftfulla verktyg och användarvänliga miljö. Därför valdes PyCharm som utvecklingsmiljö för projektet.

3.3 Kodlagring på GitHub

Under projektets gång laddades koden upp på GitHub, som fungerar som en förvaringsplats online. Där kan olika versioner sammanfogas och arbetet delas upp mellan projektets medlemmar. Tjänsten används för att underlätta dokumentationen av framtida implementeringar då det går att spåra tidigare ändringar i projektet och hämta tidigare versionshistorik (se avsnitt 2.7).

3.4 Kontinuerlig testning

Tester av applikationen i praktiska scenarier samt utvärdering av hur väl den löser de identifierade problemen och förbättrar programmeringsprocessen genomfördes löpande under projektets gång. Mindre tester utfördes kontinuerligt av projektmedlemmarna, exempelvis för att verifiera nya funktioner. Testerna presenterades för företaget löpande för att diskutera användarmöjligheterna och ta riktning för vad nästa utvecklingssteg i processen skulle vara.

3.5 Utvärdering av applikation

I slutfasen av projektet är det planerat att handledare John Tagesson Ritz själv ska testa applikationen. John har rollen Technical Lead på Granitor System och har stor erfarenhet kring TIA Portal. Syftet är att få en referens kring applikationens funktionalitet samt om det finns ett värde i att använda projektet i dess nuvarande utvecklingsfas. På grund av tidsbrist kommer eventuell feedback eller önskemål om ny funktionalitet inte att implementeras inom ramen för detta arbete, utan ligger istället som underlag för fortsatt analys och diskussion.

4
Genomförande

I detta kapitel beskrivs utvecklingen av programmet, inklusive dess uppbyggnad och funktionalitet. Dessutom ges information för vidareutveckling.

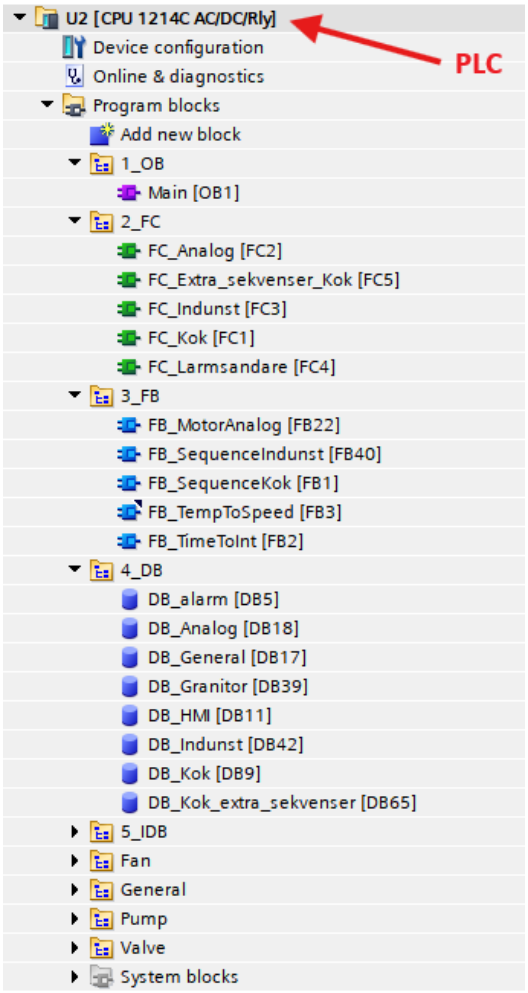
4.1 Använda TIA Scripting Python (TS)

Till TIA Scripting Python (TS) finns det ett par exempelfiler. Koden från exempelfilen `exporter.py` och `importer.py` gjordes om till funktioner som kunde köras från projektet i Python. En config-fil byggdes till där användaren kunde skriva in inställningar som behövdes för att köra projektet enligt Tabell 4.1 nedan.

Tabell 4.1: Inställningar för TIA Scripting Python (TS).

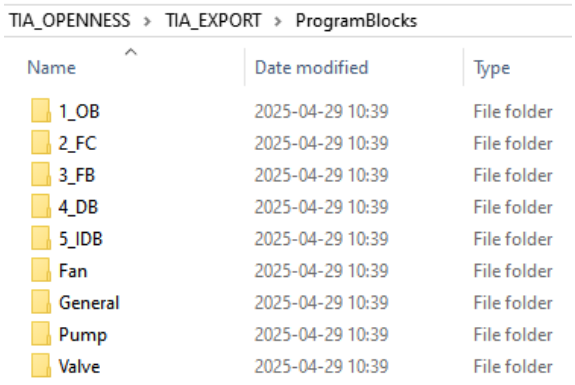
Variabel:	Data:	Förklaring:
`version`	"19.0"	Version av TIA som ska användas
`portal_mode_ui`	True	Startar grafiskt gränssnitt för TIA
`keep_tia_portal`	True	Stänger ej ner grafiskt gränssnitt
`project_file_path`	-	Fil-väg till projekt som ska användas
`temp_file_path`	-	Mapp där exporten ska sparas
`keep_folder_structure`	True	Spara ner filer i samma mapp-struktur som i TIA

När en export körs kommer TS att starta TIA Portal och försöka öppna projektfilen som är angiven enligt `project_file_path`. Detta kommer att ske grafiskt då `portal_mode_ui` är sann. TS kommer då att lista alla PLC:er i projektet och en efter en öppna trädet och leta efter mappen `Program blocks` där själva PLC-koden finns. Detta sker enligt Figur 4.1 på nästa sida.



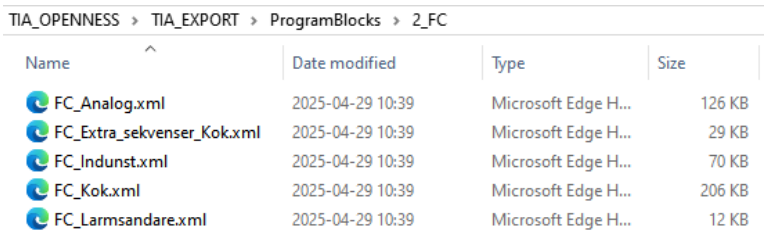
Figur 4.1: Ett projekt i TIA Portal för att visa mappstruktur.

Sedan kommer TS att försöka exportera alla `Program blocks` som XML-filer. Dessa filer kommer att sparas i samma mappstruktur som i TIA `keep_folder_structure` på platsen `temp_file_path` vilket visas i Figur 4.10.



Figur 4.2: Mappstruktur i Windows efter körd export.

Varje blocktyp kommer att få en egen XML-fil om en lyckad export sker enligt Figur 4.3.



Figur 4.3: Varje block i TIA Portal får en egen XML-fil efter export.

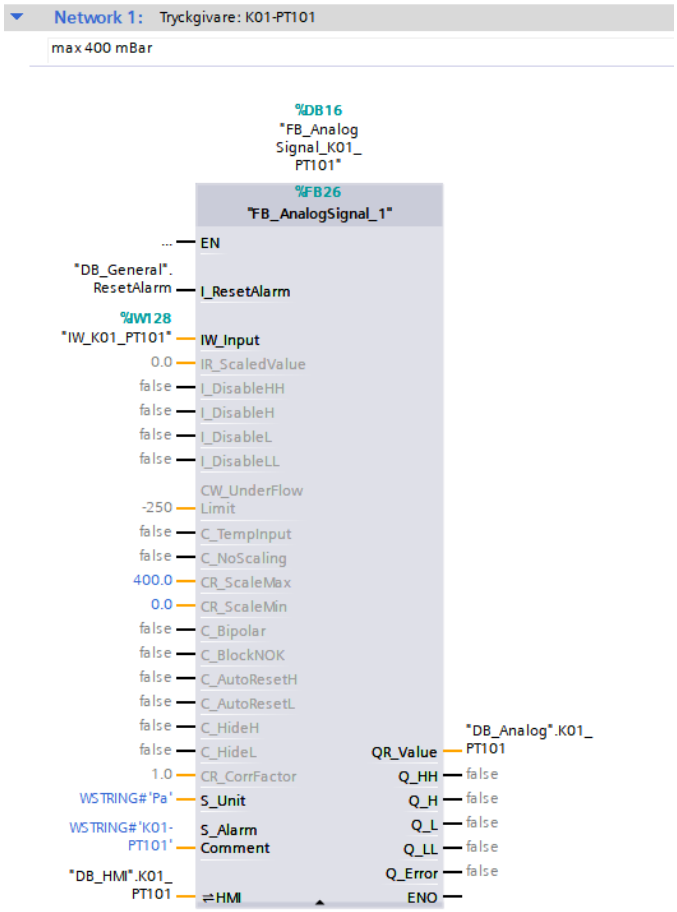
Under en import kommer TS att starta TIA Portal och öppna projektet som är angivet under `project_file_path`. TS kommer att leta efter PLC:er (markerat i Figur 4.1) och försöka importera XML-filerna som finns i mappen `temp_file_path`. Programblocken kommer därmed att uppdateras i TIA och nya programblock som läggs i mappen kan uppdateras.

Detta innebär att man kan redigera XML-filerna manuellt och göra små tester för att se om det går att importera den redigerade XML-filen. Tyvärr är dokumentationen av XML mycket begränsad av Siemens och är oftast väldigt grundläggande. För att en import ska lyckas är det viktigt att rätt information kommer på rätt ställe och att filen är komplett.

4.2 Utveckling av ett exportverktyg

För att möjliggöra effektiv återanvändning av kodstrukturer utvecklades stöd för att multiplicera instanser av ett och samma funktionsblock (FB) inom en funktion (FC). Ett typiskt användningsfall är när användaren har skapat ett FB för exempelvis en tryckgivare, med all relevant logik inbyggd. Genom att kunna infoga flera instanser av detta FB, där varje instans är konfigurerad för en specifik givare, förenklas både utveckling och underhåll av programkoden. För att detta ska vara möjligt behöver exportverktyget kunna identifiera alla förekommande FC:n i projektet samt kartlägga vilka FB:n som anropas i respektive FC. Denna information presenteras sedan i Excel för användaren, samtidigt som programmet måste hålla reda på vad det är för kod som bygger upp tryckgivaren. Detta för att kunna använda XML-filen till framtida import.

Ett FC är ofta programmerat i FBD där koden struktureras i olika nätverk eller `Network` som det heter i TIA Portal. Ett FC innehåller därför flera nätverk där varje nätverk ofta ansvarar för en specifik funktion. Figur 4.4 på nästa sida visar `Network 1` som ansvarar för en tryckgivare. Själva funktionen använder sig i sin tur av `FB_AnalogSignal_1`. Om en till tryckgivare önskas skulle en bra början kunna vara att ta hela `Network 1`, kopiera det, ändra relevant data och lägga in det i ett befintligt FC.



Figur 4.4: Ett FB för en tryckgivare placerat inuti ett `Network`.

I XML-filen representeras varje nätverk med taggen `<SW.Blocks.CompileUnit CompositionName="CompileUnits"ID="XXX">` och avslutas med avslutande tagg `</SW.Blocks.CompileUnit>`. Numreringen för varje nätverk beror på vilken position de har i dokumentet. Läggs det till ett `SW.Blocks.CompileUnit` i FC:t får det nästkommande nummer. I Figur 4.4 visas `Network 1:` som är det första `SW.Blocks.CompileUnit` i filen.

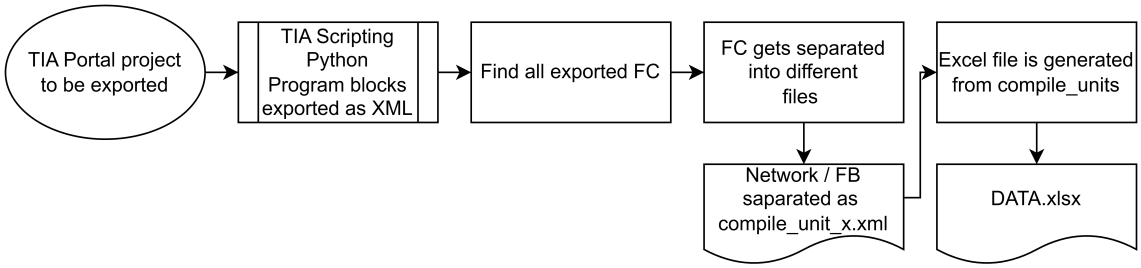
4.2.1 Uppbyggnaden av Python script för export

Genom att bygga upp ett Python-script som använder biblioteket BS4 för att tolka XML-strukturen kan innehållet i FC-filen analyseras och element som `CompileUnit` identifieras. Innehållet för dessa `CompileUnit`'s sparas som separata filer och får numreringen med hjälp av ID-taggar. Till exempel innehåller `compile_unit_0.xml` all kod som representerar `Network 1` i FC:t.

Dessa filer kan sedan läsas av och relevant information kan plockas ut med hjälp av BS4. Information som läses ut i versionen som utvecklas i projektet är `Network Name`, `Network Comment` och `Connected IDB`. Detta sparas i en Excel-fil som är autogenererad med hjälp av openpyxl. Dock måste en del information tas ut i ett tidigare skede, eftersom den inte enbart går att läsa ut från `compile_unit_0.xml`,

utan måste hämtas direkt från själva FC:t. Ett exempel är från vilket FC en viss `CompileUnit` är hämtad. Detta kan lösas genom att, vid uppdelningen av FC:t, samtidigt skicka med en array som innehåller information om ursprunget. Genom ID-taggarnas ordning kan positionen i arrayen och filnumreringen synkroniseras. Som exempel motsvarar position 3 i arrayen (`arr[2]`) filen `compile_unit_2.xml`.

Genom att spara alla `compile_unit_XX.xml` och data i Excel kan programmet avsluta efter en export. Vid en import kan data återupptas genom att läsa av Excel-filen och rätt `CompileUnit` kan hittas. Hela exportsekvensen förklaras av flödesschemat i Figur 4.5 nedan.

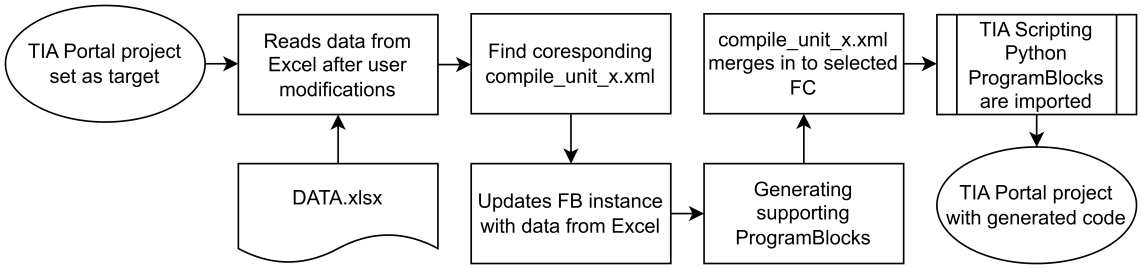


Figur 4.5: Flödesschema hur en exportsekvens går till.

4.3 Utveckling av ett importverktyg

För att autogenereringen ska fungera behöver användaren ange önskade ändringar. Detta görs genom att modifiera den exporterade Excel-filen och fylla i vilka instanser som ska läggas till, detta fungerar som en slags beställning. Beställningen sker genom att specificera vilka ID-taggar som ska användas vilket beskrivs mer ingående i avsnitt 5. Med hjälp av dessa ID-taggar kan motsvarande exporterad kod lokaliseras i form av `compile_unit_XX.xml`. Det blir då möjligt att manipulera XML-filen innan importen till TIA-portalen sker.

Denna process illustreras översiktligt i flödesschemat i Figur 4.6. Funktionaliteten har successivt byggts ut för att automatisera fler aspekter och hantera mer generella fall. I de kommande avsnitten beskrivs hur olika delar programmerades och vilka problem som uppstod under programmeringsfasen.



Figur 4.6: Förenklat flödesschema för programmering utav importsekvens.

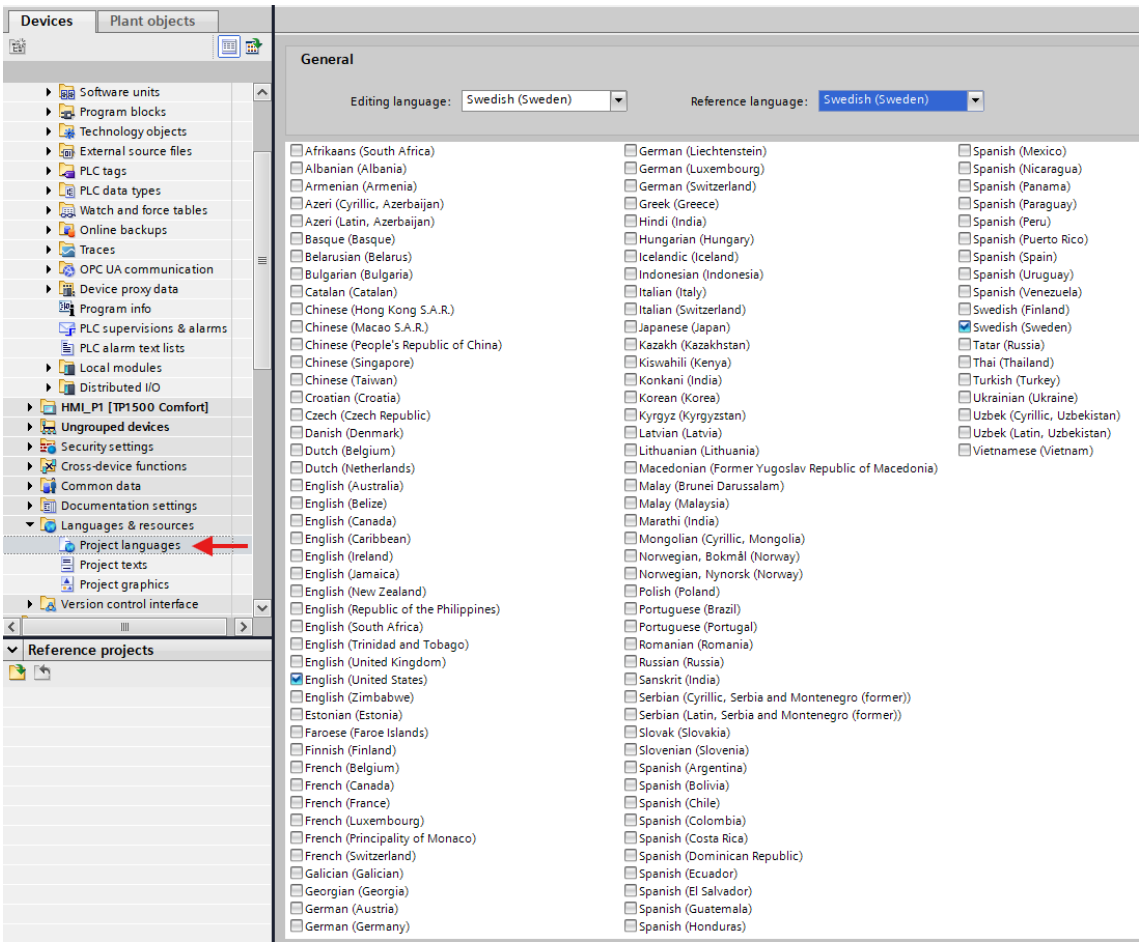
4.3.1 Ändra nätverksnamn och kommentar

I `compile_unit_XX.xml` finns `<ObjectList>` med taggen `<MultilingualText>`. Som har ett attribut för vilken data den i sin tur innehåller. Figur 4.7 nedan visar en kommentar som är angiven, den innehåller i sin tur en `<ObjectList>`. I denna `<ObjectList>` kommer ytterligare `<MultilingualText>` med en `<AttributeList>` som i sin tur innehåller själva informationen, dels vilket språk kommentaren är skriven i samt själva kommentaren. Genom att använda BeautifulSoup (BS4) kan XML-trädet navigeras för att lokalisera rätt element med specifika attribut och därmed möjliggöra ändring av exempelvis kommentarer. För att byta namn på nätverk krävs dock att element `<MultilingualText>` med attributen `Title"` modifieras, istället för `"Comment"`. Det är alltså texten som står inom taggen som uppdateras för att ändra nätverksnamnet.

```
▼<ObjectList>
  ▼<MultilingualText CompositionName="Comment" ID="5">
    ▼<ObjectList>
      ▼<MultilingualTextItem CompositionName="Items" ID="6">
        ▼<AttributeList>
            <Culture>sv-SE</Culture>
            <Text>max 400 mBar</Text>
          </AttributeList>
        </MultilingualTextItem>
```

Figur 4.7: XML för att ändra kommentar i samma `Network` som visas i Figur 4.4.

För att importen i TIA Portal ska lyckas är det avgörande att den XML-fil som importeras överensstämmer med projektets struktur. Ett problem som kan uppstå är att TIA genererar ett nytt `MultilingualText`-element med attributet `"Items"` för varje språk som är aktiverat, för att möjliggöra flerspråkiga kommentarer. TIA kommer att avvisa importen om XML-filerna innehåller olika språkkonfigurationer, eller om något språk saknas i någon av filerna. Den mest praktiska lösningen är därför att, enligt Figur 4.8, säkerställa att samma språk är aktiverade i projektets inställningar. Alternativt att endast använda applikationen på enspråkiga projekt.



Figur 4.8: Inställnings sida i TIA Portal för att ändra språk, då det är viktigt att denna inställning stämmer överens.

4.3.2 Hur IDB hanteras

Vanligtvis är ett FB kopplat till ett IDB-block. Om ett FB ska replikeras måste därför även motsvarande IDB skapas för varje instans. Genom att söka igenom koden för ett visst `compile_unit_XX.xml` kan kopplingen till IDB hittas. Då kan XML-filen för motsvarande IDB hittas och kopieras. Dock måste namnet på den nya IDB-filen uppdateras till ett unikt namn, samtidigt som referens till filen i `compile_unit_XX.xml` stämmer överens. En mer detaljerad bild över detta fås i Figur A.1 bifogad i bilaga A.

4.3.3 Sätta ihop XML-filer i ett FC

När önskvärda ändringar är gjorda i respektive `compile_unit_XX.xml` läggs de in i ett FC. Detta utförs enligt användarens önskemål, beroende på om koden ska infogas i ett befintligt FC eller om ett nytt FC ska skapas. Genom att identifiera den befintliga FC-filen och med hjälp av BS4 lokalisera den sista taggen `Network` i form av `</SW.Blocks.XXX>`, tas innehållet ur `</SW.Blocks.XXX>` och kopieras in i XML-filen för FC:t. Den nya FC-filen innehåller därmed ett extra `Network`, och efter en lyckad import kommer TIA Portal att visa detta nya `Network`.

Vid skapandet av ett nytt FC tas en kopia på en av XML-filerna för ett FC. Innehållet töms sedan genom att rensa ut alla `<SW.Blocks.XXX>` och dess innehåll. Sedan ändras data som exempelvis namn på FC:t. Då fås ett nytt FC i TIA Portal med endast det `Network` som kopierades in ifrån `</SW.Blocks.XXX>`.

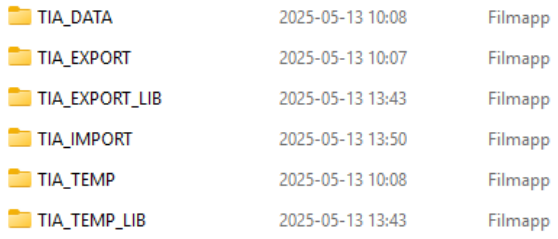
I XML-filerna finns det flera element med attributen `ID` som visas i Figur 4.9. Innan en import körs måste dessa `ID` numreras om. Dels för att `compile_unit_XX.xml` är tagna ur sin kontext, men även för att nya block som läggs in innan sista attributen med `ID` kommer upp. Därför byggdes en funktion som går igenom alla filer i en vald mapp och, med hjälp av BS4, öppnar dem för att numrera om ID:n hexadecimalt så att de hamnar i rätt ordning. Det finns även element med attributen `UId`. Dessa ska inte numreras om. De har istället att göra med kopplingen till funktionsblockets in- och utgångar.

```
▼<SW.Blocks.CompileUnit ID="3" CompositionName="CompileUnits">
```

Figur 4.9: ID från filen `compile_unit_0.xml` som behöver numreras om.

4.3.4 Använda externt bibliotek

Genom att hålla en noga filstruktur för temporära filer som export av projekt och `compile_unit_XX.xml` kan ytterligare ett TIA Portal-projekt exporteras och stoppas i en separat mapp. Detta möjliggör för användaren att kopiera testad kod från ett biblioteksprojekt och flytta in det i ett nytt projekt. Dessa temporära filer struktureras upp enligt Figur 4.10. Efter att projektet körts skapas två exportmappar samt två temporära exportmappar. Vid import används endast de XML-filer som har ändrats, vilket effektiviserar importprocessen och minskar tiden det tar att uppdatera programmet.



Figur 4.10: Filstruktur över temporära filer som skapas av Python.

4.4 Dokumentation och användarmanual

Under programmeringsutförandet har det varit väldigt viktigt att dokumentera koden, dela upp funktionalitet i olika funktioner och filer. Detta för att kunna återanvända funktioner under projektets gång men även för att göra en smidig överlämning för eventuell påbyggnad. Till det har en README-sida formulerats, dels med information kring filstruktur, men även funktionalitet och kort installationsmanual för att köra `requirements.txt`. Filen `requirements.txt` specificerar vilka Python-paket som används i projektet. När denna fil körs med `pip` installeras rätt version av varje paket. Även TS installeras automatiskt genom att en whl-fil bifogas i projektmappen.

Som komplement till README har en manual skrivits för att installera Python, GIT och PyCharm samt hur projektet klonas från GitHub. Detta syftar till att förenkla installationsprocessen, särskilt för användare som saknar tidigare erfarenhet av Python. Manualen har överlämnats till Granitor Systems medan README.md finns bifogad som bilaga B.

4.5 Test och utvärdering

Under hela utvecklingen har programmet testats med hjälp av ett nyskapat projekt i TIA-portalen samt med två projekt tillhandahållna av Granitor Systems. I början användes enbart det nyskapade "Exjobbsprojektet" för att testa funktionaliteten och se ifall det var möjligt att ändra filerna utanför TIA Portal. När problem uppstod försökte de lösas innan nästa funktionalitet implementerades. När en stabil punkt uppnåddes byttes "Exjobbsprojektet" ut mot ett av de tillhandahållna projekten. Detta ledde ofta till att buggar uppstod, vilket främst berodde på skillnader i komplexitet mellan de olika projekten.

Exempelvis kunde block skapas i fel ordning, vissa beroenden saknades eller så fungerade inte automatiska namnkopplingar som förväntat eftersom strukturen i det nya projektet avvek från den som prototypen initialt testats mot.

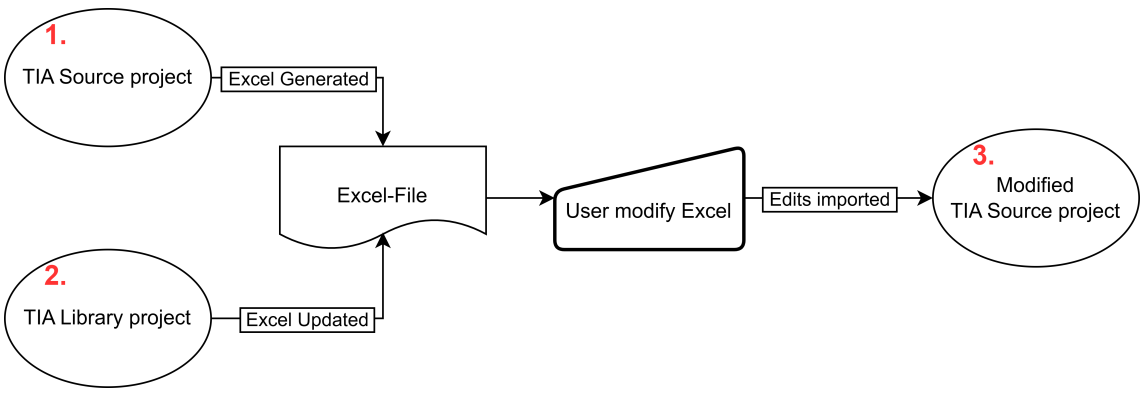
5

Resultat

Projektet har resulterat i ett program som automatiserar export, modifiering och återimport av funktionsblock mellan olika PLC-projekt. Leveransen omfattar ett terminalbaserat användargränssnitt, en Excel-baserad konfigurationsfil med navigeringsfunktionalitet, samt automatisk generering och hantering av XML-filer i bakgrunden.

5.1 Programmets funktion

Programmets funktion beror på vad användaren vill uppnå. Antingen används ett projekt i steg 1 i Figur 5.1, där projektet läses in och en Excel-fil genereras. Alternativt kan ett ytterligare projekt användas för att hämta funktionalitet från det, enligt steg 2. När filen har skapats kan användaren modifiera den utifrån sina behov. Därefter importeras de gjorda ändringarna tillbaka till projektet i steg 3.



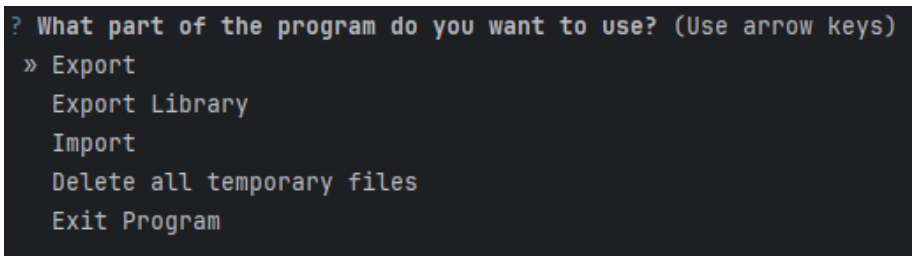
Figur 5.1: Flödesschema för programmets funktion.

5.1.1 Förutsättningar för att köra programmet

Alla inställningar för programmet anges i filen `config.py`. Här definieras bland annat sökvägar till projekt- och biblioteksfiler, versionsinställningar samt val för körläge. För en fullständig beskrivning av inställningarna, se bilaga B.

5.2 Användargränssnitt i terminalen

När programmet körs används terminalen i den IDE som användaren använder som användargränssnitt. Här får användaren välja mellan fyra olika val, fyra funktioner. `Export`, `Export Library`, `Import` och `Delete all files`. Dessa funktioner representerar olika steg i arbetsflödet för hantering av TIA Portal-projekt via Openness. Varje val leder till ett specifikt arbetsflöde, vilket förklaras mer detaljerat i följande avsnitt. Användargränssnittet illustreras i Figur 5.2 nedan.

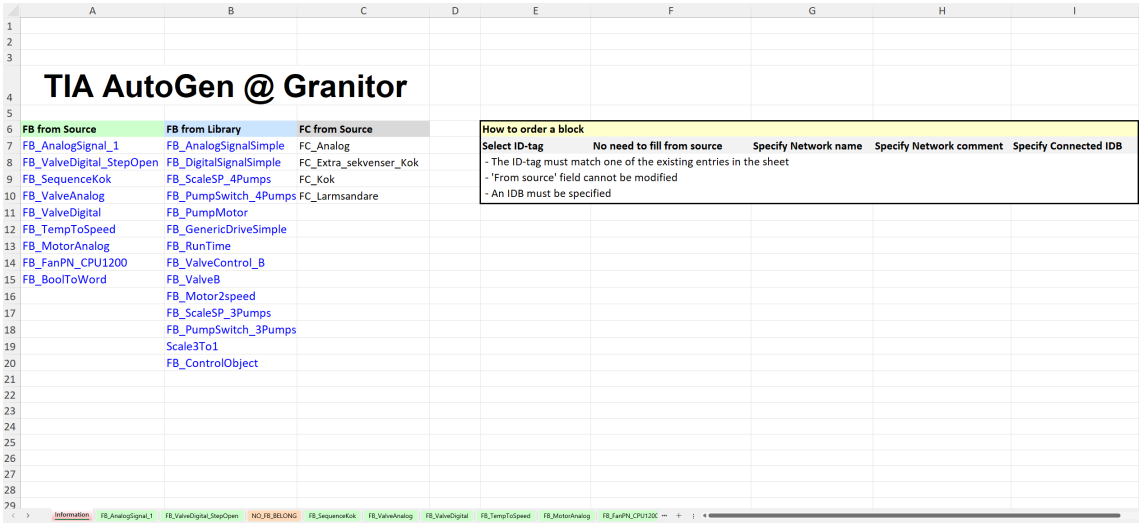


Figur 5.2: Användargränssnitt i terminalfönstret.

5.3 Användargränssnitt i Excel

Första sidan i Excel-filen fungerar som ett informationsblad som ger användaren en översikt över tillgängliga funktioner i projektet, deras placering samt logisk struktur för att underlätta navigering i större projekt. Genom inbyggda hyperlänkar (klickbara länkar inom kalkylbladet) kan användaren snabbt hoppa till rätt ark. I den gula rutan återfinns även en kort beskrivning av hur ett block beställs. En mer utförlig beskrivning återfinns i avsnitt 5.3.2.1.

Arken är färgkodade beroende på innehåll. Grönt indikerar att blocket finns i det ursprungliga projektet, blått visar att det kommer från ett externt projekt, rött markerar informationssidan, och orange används för funktionsblock som saknar en grundinstans i projektet. Ett exempel på hur detta ser ut illustreras i Figur 5.3 på nästa sida.



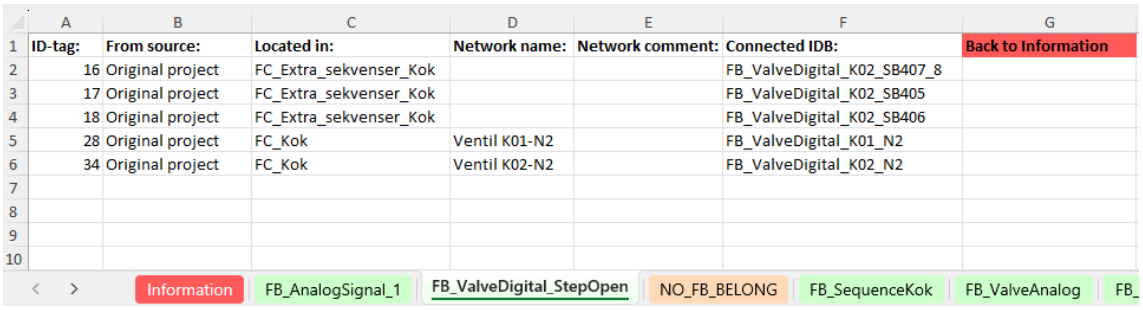
Figur 5.3: Användargränssnitt i Excel.

5.3.1 Export

Funktionen `Export` tar ett projekt som användaren väljer och extraherar alla instanser där ett visst funktionsblock används. Dessa delas upp i separata ark, ett för varje blocktyp. Informationen som presenteras för användaren för varje instans inkluderar nätverksnamn, nätverkskommentar samt eventuellt namn på ett kopplat DB-block. För att underlätta navigering finns en hyperlänk med texten `Back to Information` som leder tillbaka till informationssidan.

De genererade kalkylarken är delvis låsta för att säkerställa att endast specifika celler kan redigeras av användaren. Samtidigt skapar programmet en mappstruktur i bakgrunden där motsvarande XML-filer exporteras. Ett unikt ID-nummer som identifierar varje nätverk genereras, samt en tillhörighetsmarkör som anger om blocket har hämtats från det ursprungliga projektet eller från ett annat projekt (se avsnitt 5.3.2).

Figur 5.4 visar ett exempel på hur denna information presenteras för användaren i Excel. Varje rad motsvarar ett nätverk, där kolumner visar information som nätverksnamn, kommentar, blockets ursprung samt det genererade ID-numret.



Figur 5.4: Information från exporterat projekt utskrivet i Excel.

5.3.2 Export Library

Användaren kan välja att enbart arbeta med ett projekt, vilket innebär att endast funktionen `Export` används. Om användaren däremot vill utnyttja möjligheten att använda det ena projektet som ett bibliotek för att hämta funktioner därifrån, används istället `Export Library`.

Även i detta fall organiseras instanser av varje funktion i separata ark, ett per blocktyp. Informationen som presenteras för användaren är densamma som i Export-funktionen. Om två block med samma namn förekommer, ett från vardera projekt, samlas dessa i samma ark. I sådana fall specificerar kolumnen `From source` vilket projekt blocket hämtas från.

Figur 5.5 illustrerar hur ett exporterat projekt presenteras i Excel. I detta exempel visas hur block av typen `Function` från två olika projekt samlas i ett gemensamt kalkylblad. Kolumnrubriker som `Name`, `Network Comment`, `Connected IDB` och `From source` gör det möjligt för användaren att identifiera och jämföra block mellan projekten.

	A	B	C	D	E	F
1	**ID-tag:**	**From source:**	**Located in:**	**Network name:**	**Network comment:**	**Connected IDB:**
2	14	Library project	FC_Digital	Digital in: Gaslarm klordioxid		DigitalSignal_GS1
3	15	Library project	FC_Digital	Digital in: Utlöst säkring		DigitalSignal_AE1_FS
4	16	Library project	FC_Digital			DigitalSignal_GL503
5	17	Library project	FC_Digital			DigitalSignal_GL504
6	18	Library project	FC_Digital			DigitalSignal_UPS_OK
7	19	Library project	FC_Digital			DigitalSignal_UPS_RDY
8						
9						
10						

FB_MotorAnalog | FB_FanPN_CPU1200 | FB_AnalogSignalSimple

Figur 5.5: Information från två exporterade projekt utskrivna i Excel.

5.3.2.1 Beställning

För att göra en beställning av ett block fyller användaren i den nästa tillgängliga tomma raden i det ark som motsvarar det block som ska beställas. Användaren väljer en ID-tagg som representerar den specifika instansen av blocket. Därefter anges i vilken funktionskomponent (FC) det nya blocket ska placeras. Det är möjligt att välja bland befintliga FC:er i det ursprungliga projektet eller att skapa en ny, där användaren själv bestämmer namnet.

Utöver detta anger användaren nätverksnamn, nätverkskommentar samt om ett tillhörande datablock (IDB) ska kopplas. Kolumnen `From source` är låst och kan inte redigeras av användaren eftersom ursprunget till blocket istället bestäms indirekt via den valda ID-taggen, där referensen till ursprungsobjektet finns angiven. Beställningen av det nya blocket är markerad i grått i Figur 5.6. Eftersom ID-tag 14 har valts kommer ett block med ID 14 att hämtas från Library project i detta fall.

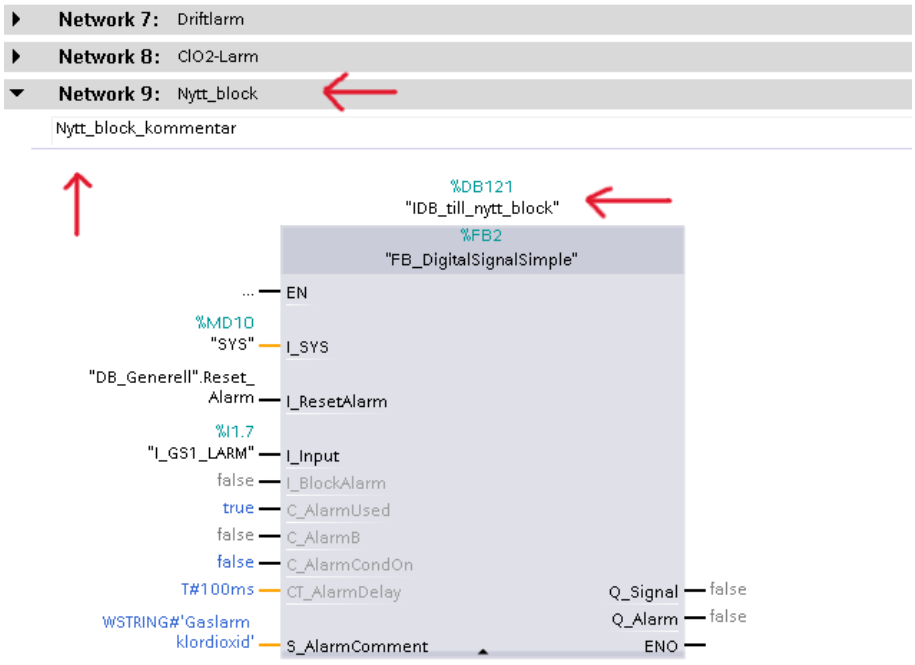
	A	B	C	D	E	F
1	**ID-tag:**	**From source:**	**Located in:**	**Network name:**	**Network comment:**	**Connected IDB:**
2	14	Library project	FC_Digital	Digital in: Gaslarm klordioxid		DigitalSignal_GS1
3	15	Library project	FC_Digital	Digital in: Utlöst säkring		DigitalSignal_AE1_FS
4	16	Library project	FC_Digital			DigitalSignal_GL503
5	17	Library project	FC_Digital			DigitalSignal_GL504
6	18	Library project	FC_Digital			DigitalSignal_UPS_OK
7	19	Library project	FC_Digital			DigitalSignal_UPS_RDY
8	14		Befintligt FC	Nytt_block	Nytt_block_kommentar	IDB_till_nytt_block
9						
10						
11						
12						
13						
14						
15						
16						
17						

FB_AnalogSignalSimple | FB_DigitalSignalSimple | FB_ScaleSP_4Pun

Figur 5.6: Beställning av funktionsblock inuti Excel.

5.3.3 Import

För att modifiera projektet och importera de block som lagts till används funktionen `Import`. De modifierade XML-filerna genereras utifrån användarens ifyllda information i Excel, och importeras sedan automatiskt tillbaka till TIA Portal-projektet. Ett exempel på hur det kan se ut i TIA Portal efter att funktionen körts och blocken har importerats visas i Figur 5.7.



Figur 5.7: Importerat funktionsblock från Excel.

5.3.4 Delete all temporary files

Funktionen `Delete all temporary files` används för att återställa programmet så att det kan köras igen. Här tas all nuvarande mappstruktur bort för att möjliggöra för användaren att använda ett nytt projekt. Detta inkluderar rensning av temporära mappar och tidigare exporterade XML-filer.

Innan borttagningen genomförs visas en kontrollfråga som bekräftar att användaren verkligen vill radera filerna, se Figur 5.8. Det är viktigt att notera att själva PLC-projektet som importerats till programmet inte påverkas av denna funktion.

```
? What part of the program do you want to use? Delete all temporary files
? Are you sure you want to delete all temporary files (including DATA.xlsx)? (Y/n)
```

Figur 5.8: Kontrollfråga vid körning av `Delete all temporary files`.

6
Diskussion

Detta kapitel innehåller diskussion och reflektion kring projektet samt förslag på framtida utvecklingsområden.

6.1 Uppföljning av syfte och mål

För att kunna uppnå målet med projektet, att effektivisera programmeringsarbetet genom automatiserad kodgenerering med hjälp av TIA Portal Openness, undersöktes en rad tekniska möjligheter och begränsningar. Inledningsvis analyserades befintliga lösningar och exempelprojekt för att kartlägga hur TIA Portal Openness används i praktiken. Det konstaterades att de flesta funktioner i TIA Portal kan utföras programmässigt via Openness, vilket bekräftade verktygets potential.

Vidare prövades om befintliga funktionsblock kunde konfigureras automatiskt. Det visade sig vara möjligt att ändra blocknamn, nätverkskommentarer och associerade datablock via export och import av XML-filer. Genom att kombinera denna funktionalitet med användardata blev det också möjligt att styra anpassningen dynamiskt, vilket visar att automatiserad konfiguration av funktionsblock är genomförbar.

Däremot uppstod begränsningar när det gällde att generera en komplett ny PLC-struktur från grunden. TIA Portal Openness lämpar sig bättre för modifiering av befintliga projekt än för att skapa helt nya, vilket innebar att ett mallprojekt användes som utgångspunkt.

Slutligen utvärderades Excel som gränssnitt för datainmatning. Resultatet visade att Excel fungerar väl för ändamålet och kan användas på ett effektivt sätt tillsammans med ett Python-baserat terminalgränssnitt för att styra programmet.

6.2 Utvärdering av applikationen

I detta delkapitel utvärderas projektets resultat med fokus på den utvecklade applikationen. Utvärderingen bygger både på praktisk testning och dialoger med anställda från Granitor Systems, samt egna reflektioner kring applikationens funktionalitet, användarvänlighet och framtida potential.

6.2.1 Synpunkter från Granitor Systems

Applikationen som togs fram under projektet lyckas generera kod enligt beställning i Excel, om koden kommer från det ursprungliga projektet fungerar det mycket bra. Även om applikationen behöver utökas för att konfigurera vissa delar såsom in- och utgångar. Användningen av Excel ger en mycket bra övergripande bild över de olika funktionsblocken. Informationssidan med hyperlänkar uppskattas för att växla mellan de olika flikarna på ett effektivt sätt. Ifyllnaden går betydligt snabbare med hjälp av Excels olika verktyg, vilket möjliggör en mer korrekt och effektiv automatisk namngivning. Funktionen för kopiering och inklistring, tillsammans med möjligheten att generera nummerserier, underlättar processen ytterligare. Enligt handledaren John, bedöms applikationens nuvarande utformning redan idag kunna bidra till tidsbesparingar i arbetet.

6.2.1.1 Framtiden för autogenerering med Openness

Möjligheterna med Openness och XML är mycket omfattande, och detta projekt har endast skrapat på ytan av vad som är möjligt att åstadkomma. För att citera John Tagesson Ritz: "Hade man kunnat utveckla projektet och implementera möjligheten att konfigurera in- och utgångar, hade det minst varit 10 gånger så effektivt". Förhoppningsvis kan detta projekt fungera som ett startskott för Granitor Systems i Mölndal att våga utforska och vidareutveckla egna applikationer baserade på detta arbete.

6.2.2 Användarvänligheten av applikation

Trots bra dokumentation kring hur en installation av applikationen ska gå till är tillvägagångssättet problematiskt. Rätt versioner av mjukvaror ska laddas ner, tillsammans med kod som hämtas via GitHub. Filvägen till själva projektet måste vara formaterad på ett visst sätt och ligga i en viss fil. Det finns flera moment där det kan gå fel trots manualer, om inte blir det tidskrävande och osmidigt. Att behöva installera mjukvaror på flera olika håll är även en säkerhetsrisk då utveckling i viss mån behöver ske offline för att skydda klienter. Här hade en `.exe`-fil som går att hämta lokalt vara en stor vinning. Problemet detta leder till är att en färdig installationsfil kräver en stor tillförlitlighet för programmet. Idag finns det möjlighet att läsa ut debugs på ett tydligt sätt och möjlighet att kunna lösa buggar på ett smidigt sätt, vilket inte hade gått med en färdig `.exe`-fil.

Terminalfönstrets användargränssnitt är enkelt och intuitivt, vilket gör det lätt att följa och välja vilken del av programmet som ska köras. Den extra frågan om att bekräfta `Delete all files` fungerar väl som en dubbel säkerhetsåtgärd för att förhindra oavsiktliga borttagningar. I dagsläget kräver vissa inställningar att användaren manuellt redigerar en Python-fil, vilket kan upplevas som otydligt. Exempelvis hade valet av projektfil kunnat lösas smidigare än genom nuvarande implementering med Questionary.

I versionen som togs fram under projektet måste användaren stänga ner projektet i TIA Portal innan applikationen körs, annars kommer TS försöka öppna ett redan öppnat projekt. Detta kommer resultera i att applikationen kraschar, då det inte går att ha samma projekt öppet i två instanser av TIA Portal. Det hade varit intressant att se över möjligheten att kunna ansluta till ett redan öppet projekt i TIA-portalen för att slippa starta om TIA Portal.

Användargränssnittet i Excel fungerar som ett effektivt verktyg för användarinteraktion med programmet. Informationssidan ger en tydlig överblick över projektet och visar vilka typer av block som ingår. Färgkodningen av bladen underlättar identifiering av funktionernas ursprung och deras tillgänglighet. Dropdown-menyn för valet av `Located in:` gör det enkelt för användaren att navigera bland befintliga FC:n i projektet.

6.2.3 Problematik med applikationen

Applikationen har möjlighet att använda ett biblioteksprojekt vilket möjliggör flytt av funktionsblock till ett annat projekt. Tanken är väldigt bra men i verkligheten är användbarheten ganska låg då det oftast inte sker utan problem. Detta beror på komplexiteten för verkliga funktionsblock och strukturer (structs) som ofta används. Det förekommer även fall där ett FB i sin tur kan innehålla flera FB:n. Då måste alla referenser flyttas med in i det nya projektet. Problematiken kan bli när vissa FB:n bygger på kod utgiven av Siemens själva, oftast ger de inte ut koden till sina egna funktionsblock. Det gör att användaren får ett block som saknar referenser som manuellt måste flyttas, då det inte går att få ut XML-filer till dessa block gjorda av Siemens.

En ytterligare begränsning med programmet kopplat till användningen av två projekt är att de båda måste vara av samma version av TIA Portal för att kunna användas tillsammans. Om versionerna inte matchar krävs det att projektet uppgraderas i TIA först. Generellt är det ett problem då vissa kunder önskar att en äldre version av TIA används för att bibehålla kompatibilitet. En generell applikation som fungerar på flera versioner är svår att underhålla.

Detta resulterar i att det i dagsläget enbart går att importera mycket enkla och primitiva funktionsblock som inte innehåller structs och utan referenser till funktionsblock i dem. Dock är detta enbart ett problem när det kommer från ett utomstående biblioteksprojekt. Finns referenserna i projektet går det fortfarande att ha flera dubbletter. Flera av problemen skulle förmodligen kunna lösas med mer programmeringstid, med stor risk för fel.

En ytterligare problematik som upptäcks är att PLC-koden ibland är dåligt kommenterad. Kommentarer och nätverksnamn kan saknas. Detta gör det mycket otydligt att identifiera koden i Excel. Ett annat problem är att kommentarerna inte stämmer överens mellan språken. Ett exempel är att det skulle kunna stå tryckgivare på svenska och motorstyrning på engelska, då användaren bara ser det inställda språket. När en algoritm sedan ska läsa av XML-filen vet den inte vilket språk som ska användas.

Ett till problem är att XML-filerna är uppbyggda på olika sätt beroende på vilket projekt de tillhör och vilken typ av block de representerar. Detta innebär att applikationen är begränsad till att hantera filer med en specifik struktur. Programmet förutsätter att XML-filerna följer det format som använts under utvecklingen, vilket innebär att det inte är generellt tillämpbart. Om användaren försöker använda ett projekt med en annan funktionalitet eller struktur än de som testats, kan filerna inte importeras korrekt. Denna begränsning minskar programmets flexibilitet och gör att det enbart är användbart för ett snävt urval av projektkonfigurationer.

6.3 Fortsatt utveckling

Även om applikationen redan kan användas för att generera delar av PLC-program, finns det behov av vidareutveckling för att öka dess användbarhet.

6.3.1 Rekommenderad vidareutvecklingen

När applikationerna växer ökar även funktionaliteten, vilket kan leda till en ökad tidsbesparing för varje ny funktion som implementeras. Samtidigt ökar risken för buggar, vilket gör det svårt att hitta en balanserad väg framåt. Ett möjligt steg för Granitor Systems skulle kunna vara att anställa en mjukvaruutvecklare med ansvar för att utveckla och underhålla interna verktyg som effektiviserar arbetet för PLC-programmerare. Utvecklarens roll skulle även kunna inkludera att implementera automatiska tester för att säkerställa hög tillförlitlighet och snabb identifiering av fel.

En mer realistisk framtidsplan är att stegvis vidareutveckla applikationen med fokus på att behålla enkelheten. Parallellt vore det värdefullt att ta fram riktlinjer för hur ett TIA-projekt bör struktureras, för att applikationen ska kunna användas så effektivt som möjligt. Det hade också varit mycket givande att lansera en första version i en verklig arbetsmiljö och samtidigt samla in feedback från användare. Den fortsatta utvecklingen kan delas in i följande riktningar.

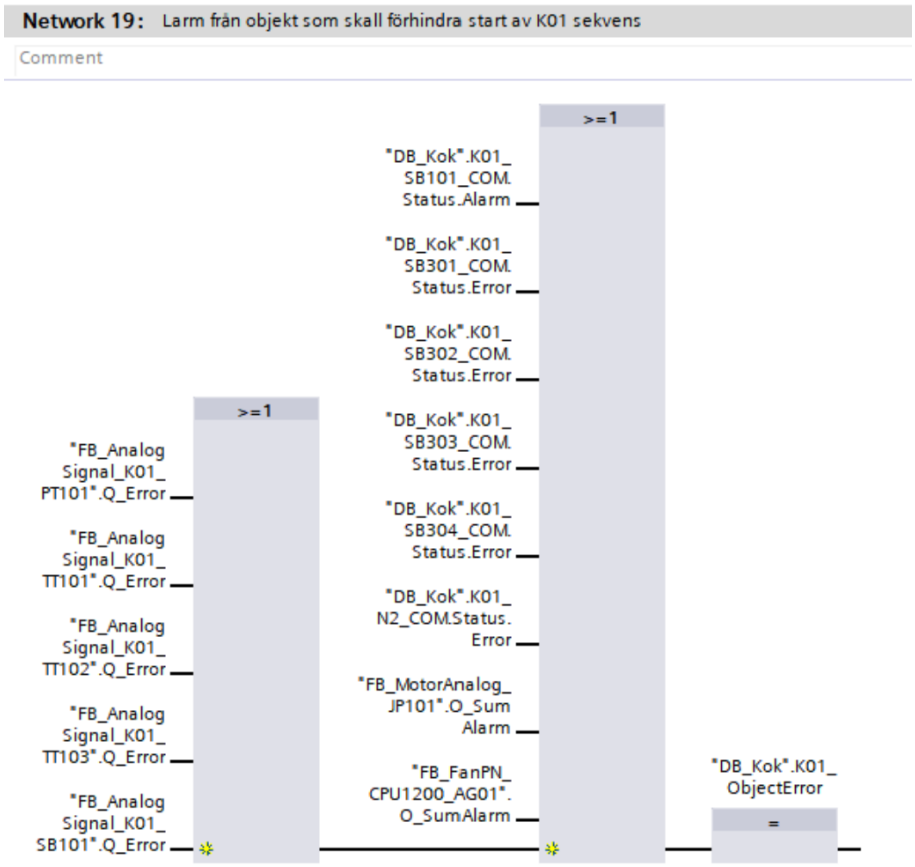
6.3.1.1 Paketering av projekt och användarvänlighet

För att få ut programmet bland programmerarna på Granitor krävs att installationen förenklas, dels säkerhetsmässigt, men även för att göra det enklare. För att göra detta kommer ett enkelt användargränssnitt att behövas istället för terminalen och Questionary. Trots att Excel används som gränssnitt för projektet, hade framtida arbete kunnat fokusera på att få till ett mer anpassat användargränssnitt för att underlätta för användarna. Det skulle till exempel kunna innebära att skapa ett fristående program med tydligare inmatningsfält och bättre överblick, vilket skulle underlätta användningen för slutanvändaren. Möjlighet att kunna peka på ett projekt i utforskaren hade gjort det enkelt, och på så sätt slippa redigera i olika konfigurationsfiler.

6.3.1.2 Enbart fokusera på ett TIA Portal projekt

För att öka driftsäkerheten och pålitligheten bör applikationen förenklas. Att använda ett externt projekt i TIA-portalen för import är en ambitiös lösning, men det kan vara mer ändamålsenligt att låta användaren själv lägga in funktionsblocken manuellt i sitt projekt. Därefter kan en applikation användas för att automatiskt identifiera och hantera de befintliga blocken.

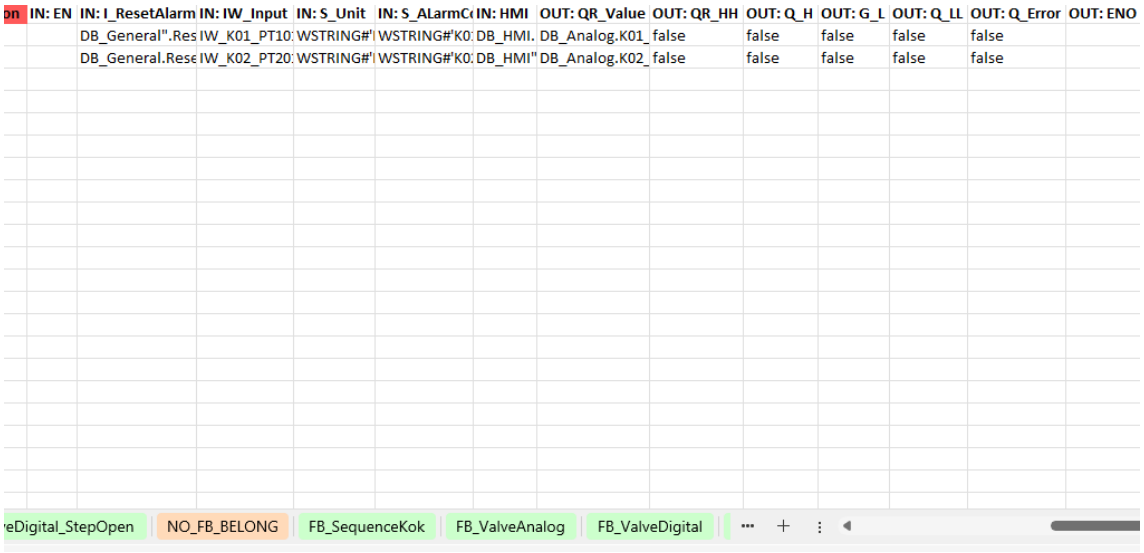
Ett syfte med att kunna importera extern kod är att begränsa funktionaliteten till enbart grundläggande funktioner, såsom larmsummering visad i Figur 6.1. Genom att tillåta användaren att skicka med ett par XML-filer för olika typer av larmsummering, valbara direkt i Excel, skulle man kunna erbjuda stor flexibilitet utan att komplicera användningen nämnvärt. Det skulle även vara möjligt att bygga upp ett bibliotek med summering för olika versioner av TIA-portalen. Exempelvis skulle användaren, vid val av TIA Portal V16, endast få möjlighet att välja XML-filer som är kompatibla med just den versionen.



Figur 6.1: Exempel på alarmering där flera alarm sammanfogas till en signal.

6.3.1.3 In- och utgångar

Att kunna implementera in- och utgångar i applikationen hade varit en stor vinst. I slutet av projektet kom det fram att det även hade varit vinning att enbart kunna presentera in- och utgångar i Excel. Det skulle även underlätta felsökning i flera instanser av funktionsblock. Figur 6.2 illustrerar ett exempel på hur det skulle kunna se ut i Excel. Efter de redan existerande rubrikerna skrivs först blockets ingångar med namn på dessa följt av utgångar. Ingångar markeras förslagsvis med `IN:` och utgångar med `OUT:`.

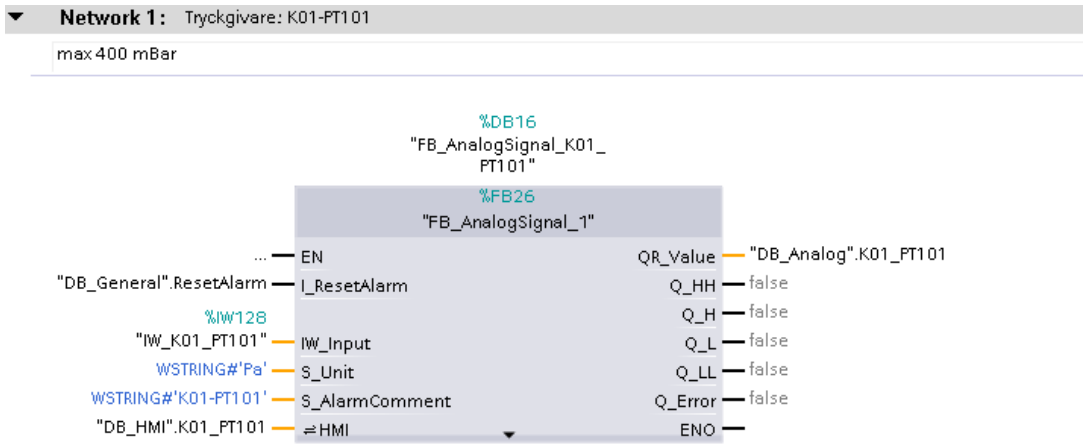


on	IN: EN	IN: I_ResetAlarm	IN: IW_Input	IN: S_Unit	IN: S_ALarmC	IN: HMI	OUT: QR_Value	OUT: QR_HH	OUT: Q_H	OUT: G_L	OUT: Q_LL	OUT: Q_Error	OUT: ENO
		DB_General".Res	IW_K01_PT10	WSTRING#'I	WSTRING#'K0	DB_HMI.	DB_Analog.K01_	false	false	false	false	false	
		DB_General.Rese	IW_K02_PT20	WSTRING#'I	WSTRING#'K0	DB_HMI"	DB_Analog.K02_	false	false	false	false	false	

eDigital_StepOpen | NO_FB_BELONG | FB_SequenceKok | FB_ValveAnalog | FB_ValveDigital

Figur 6.2: Förslag på hur ingångar och utgångar kan skrivas ut i Excel.

Figur 6.3 nedan visar ett exempel på ett block i TIA Portal med in- och utgångar.



Figur 6.3: Ingångar och utgångar i TIA Portal för ett funktionsblock.

Genom att integrera denna funktion skulle programmets användningsområde kunna breddas ytterligare. Det skulle till exempel möjliggöra mer detaljerad konfiguration av blocksamband, vilket är särskilt värdefullt i mer komplexa system.

En sådan utökning av programmet medför dock ökad komplexitet i programkoden. Eftersom antalet in- och utgångar varierar mellan olika block, krävs det att funktionen implementeras på ett dynamiskt sätt. Dokumentinläsningen behöver då hantera ett större informationsinnehåll, vilket även ökar kraven på hantering och uppdatering av XML-filerna för att uppnå önskade förändringar. Den uppskattade tidsåtgången för att implementera denna funktionalitet är svår att fastställa, då den beror på flera faktorer såsom komplexitet i datastrukturen och behovet av gränssnittsjusteringar.

6.3.1.4 Redigera befintlig kod

I dagens utförande av projektet går det enbart att redigera kod som ska importeras in i projektet. Ska en ändring göras måste detta ske i TIA-Portalen. En utveckling av programmet skulle kunna vara att göra det möjligt att ändra data i redan existerande kod. Detta skulle göra det möjligt att ändra flera block samtidigt. Om ett konfigurationsfel skulle upptäckas skulle detta snabbt kunna åtgärdas. Ett annat ändamål är att kunna kommentera koden efteråt för dokumentation.

7
Slutsats

Projektets syfte var att undersöka hur repetitiva moment inom PLC-programmering kan elimineras för att effektivisera arbetsflödet och minska risken för fel. Genom att utveckla en applikation som automatiskt genererar PLC-programstruktur utifrån användardata har detta mål uppnåtts. Applikationen, som baseras på TIA Portal Openness och styrs via ett användargränssnitt i Excel, visar att det är fullt möjligt att automatisera delar av det annars monotona programmeringsarbetet.

Resultatet visar att en viss tidsbesparing kan uppnås redan med den version som utvecklades under projektet, samtidigt som risken för fel minskar när manuella moment elimineras. Arbetet bekräftar därmed att ett användarvänligt gränssnitt och ett flexibelt importflöde kan bidra till både ökad effektivitet och högre driftsäkerhet. Vidare pekar projektet mot flera möjliga utvecklingsvägar, vilket gör lösningen skalbar och anpassningsbar för olika behov inom industriell automation. Med de föreslagna framtida förbättringarna kan detta projekt fungera som ett exempel på hur liknande koncept kan utvecklas och vidareutvecklas i framtiden.

Litteraturförteckning

[1] Siemens AG, *TIA Scripting Python: Application Example Manual*, Entry-ID: 109742322, V1.0.7, Nov. 2024. [Online]. Available: https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(tia-scripting-python)

[2] M. Reddy, *API Design for C++*. Elsevier Science Technology, 2011. [Online]. Available: Ebook Central, EBC Academic Complete. [Accessed: Apr. 3, 2025].

[3] E. Mohn, *XML (Extensible Markup Language)*, *Salem Press Encyclopedia of Science*, 2024.

[4] M. Osbeck and G. Hult, *Styrteknik PLC*. Göteborg, Sverige: Chalmers tekniska högskola, 2020

[5] J. Jansson, *PLC-programmering: Vad är det och hur fungerar det?*, Teknik Media, 2022. [Online]. Available: https://www.teknik-media.se/plc-programmering-vad-ar-det-och-hur-fungerar-det/. [Accessed: Apr. 6, 2025].

[6] Flaus J-M. Cybersecurity of Industrial Systems. John Wiley & Sons (US); 2019. Accessed April 22, 2025. https://research.ebsco.com/linkprocessor/plink?id=5d709551-71bb-36e4-9f79-5faf06fa570e

[7] Siemens AG, Programming Guideline for S7-1200/1500, Entry ID: 81318674, V1.6, Dec. 2018. [Online]. Available: https://support.industry.siemens.com/cs/document/81318674. [Accessed: Apr. 8, 2025].

[8] L. Lins, The Complete Practical Guide to Siemens Tia Portal Programming, SolisPLC. [Online]. Available: https://www.solisplc.com/tutorials/a-practical-guide-to-siemens-tia-portal-programming. [Accessed: Apr. 8, 2025].

[9] T. Gaspar, TIA Portal Openness [PowerPoint presentation], Siemens, 2018. [Online]. Available: https://assets.new.siemens.com/siemens/assets/api/uuid:0fdd52a4-c384-4e55-a89dba9181d17fc7/tia-openness.pdf

[10] Siemens AG, "TIA Portal Openness: Automation of Engineering Workflows," Siemens Industry Online Support, [Online]. Available: https://support.industry.siemens.com/cs/document/109792902/tia-portal-openness-automation-of-engineering-workflows?dti=0&lc=en-SE. [Accessed: 18-May-2025].

[11] Siemens AG, "TIA Portal Version 20: Enabling peak performance and efficiency," Press Release, Nov. 7, 2024. [Online]. Available: https://press.siemens.com/global/en/pressrelease/tia-portal-version-20-enabling-peak-performance-and-efficiency

[12] Siemens AG, "How is the XML file structured for blocks (Export/Import in TIA Portal Openness V13 SP1)?", FAQ Entry ID: 109480446, Dec. 14, 2015. [Online]. Available: https://support.industry.siemens.com/cs/document/109480446

[13] Siemens AG, TIA Portal Openness Explorer,Äpplication example, Entry ID: 109760816, Sep. 26, 2022. [Online]. Available: https://www.siemens.com/cybersecurity#Ouraspiration. [Accessed: 21-May-2025].

[14] Siemens, “Excel code generator for TIA Portal Openness,” TIA Portal Openness / Excel code generator, Entry ID: 109770550, V2.0, 11/2024. [Online]. Available: https://support.industry.siemens.com/cs/document/109770550. [Accessed: 13-May-2025].

[15] Siemens AG, *TIA Scripting Python: Application Example Manual*, Entry-ID: 109742322, V1.0.7, Nov. 2024. [Online]. Available: https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(tia-scripting-python)

[16] A. Pipinellis, GitHub Essentials: Unleash the Power of Collaborative Development Workflows Using GitHub. Packt Publishing, 2018. [Online]. Available: Ebook Central (EBC Academic Complete)

[17] A. Aly, Hands-On Enterprise Automation with Python: Automate Common Administrative and Security Tasks with Python. Packt Publishing, 2018. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=39a452c1-b392-3794-921a-9a3228fa3cd5. [Accessed: Apr. 18, 2025].

[18] Python Packaging Authority, “Overview of Python Packaging,” *Python Packaging User Guide*, 2024. [Online]. Available: https://packaging.python.org/en/latest/overview/. [Accessed: May 19, 2025].

[19] TIA Scripting Python: Using Python Scripts to Automate TIA Portal Processes, Siemens Industry Online Support, Entry-ID: 109742322, Version 1.0.7, Nov. 2024. [Online]. Available: https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(tia-scripting-python)

[20] L. Richardson, *Beautiful Soup Documentation*, [Online]. Available: https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed: Apr. 8, 2025].

[21] S. Sanderson and D. Kun, 1.7.4 Retrieving Sheet Data with openpyxl, in Extending Excel with Python and R, Packt Publishing, 2024. [Online]. Available: https://research.ebsco.com/linkprocessor/plink?id=27a1eafc-6428-3383-9b3b-44ad89f233ea. [Accessed: Apr. 18, 2025].

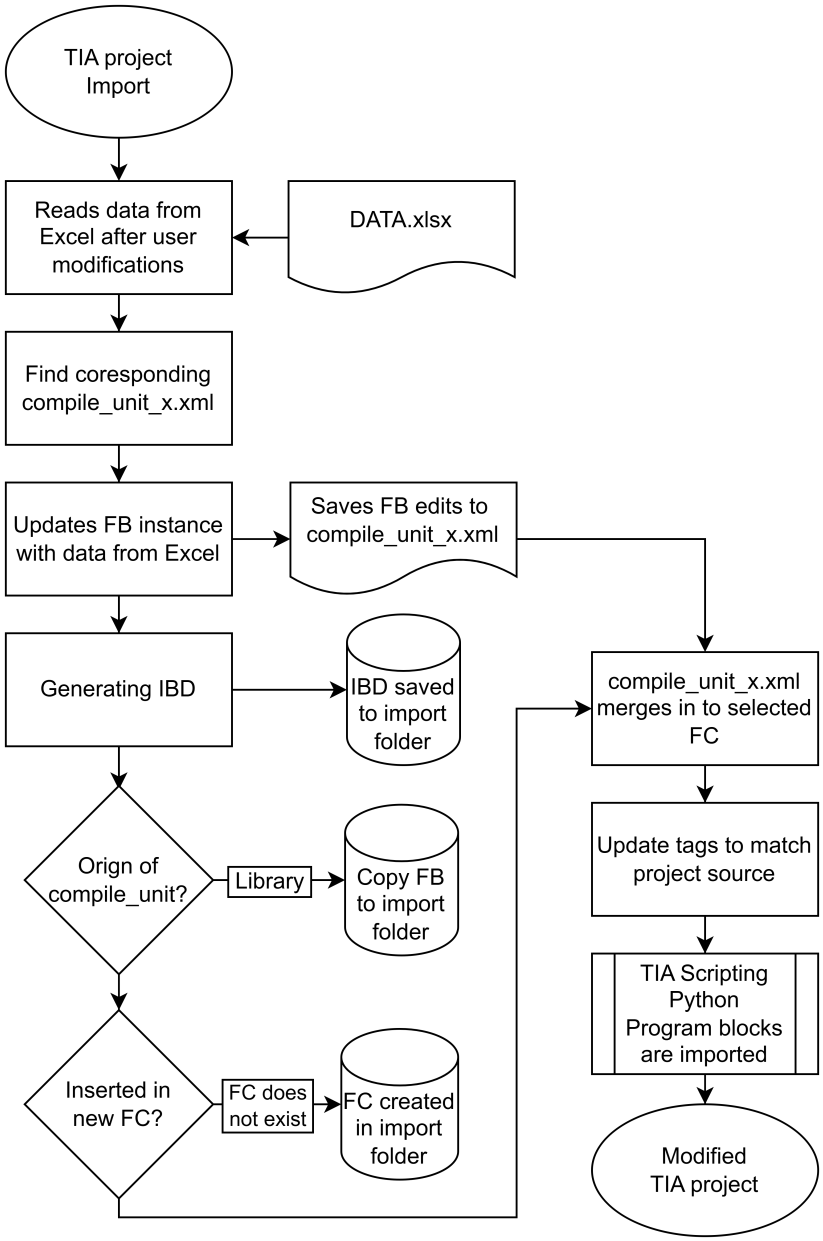
[22] T. Bocklisch, Questionary: Python library for beautiful command line interfaces, version 1.10.0. [Online]. Available: https://github.com/tmbo/questionary [Accessed: May 18, 2025].

[23] TIA Scripting Python: Using Python Scripts to Automate TIA Portal Processes, Siemens Industry Online Support, Entry-ID: 109742322, Version 1.0.7, Nov. 2024. [Online]. Available: https://support.industry.siemens.com/cs/document/109742322/tool-for-easier-use-of-the-tia-portal-openness-interface-(tia-scripting-python)

A

Flödesschema import

Figur A.1 visar ett mer detaljerat flödesschema än det som refereras i Figur 4.6. Ger en mer noggrann bild om vilka filer som ändras och skapas under en import.



Figur A.1: Detaljerat flödesschema för importsekvens.

B

Dokumentation: README.md

README.md är en fil som används för att att samla information för att programmerings projekt med relevant information. På nästkommande sida kommer README-filen för projektet som byggdes upp.

TIA AutoGen @ Granitor

Thesis work of Samuel Rosén and Erik Müller.
This project explores the possibility of automatically generating PLC code using TIA Openness.

Prerequisites

- Windows OS with TIA Portal installed. Only version 19 is tested.
- Python version 3.12.x
- [Optional] PyCharm IDE for better development experience

Setup

> **Important:** This project requires **Python 3.12.x**

If you're using **PyCharm**, make sure to enable the "Emulate terminal in output console" option in your Run/Debug Configuration.
This is required for the `questionary` prompts to function properly.

Enabling terminal emulation in PyCharm

1. Click the three dots in the top-right corner of the Run/Debug configuration panel, then select **Edit...**
2. Open the dropdown: **Modify options**
3. Under **Add run options**, check the box: **Emulate terminal in output console**

Install Packages

Install dependencies by running:

```
python -m pip install -r requirements.txt
```

TIA Scripting Python (Manual Installation)

> Normally installed automatically via `requirements.txt`.
> If you need to update or install it manually, follow these steps:

1. Download the `.whl` file for **TIA Scripting Python**
2. Unzip to a folder on your machine
3. Open a terminal and navigate to the extracted folder:

```
cd C:\path\to\your\folder
```

4. Install using:

```
pip install siemens_tia_scripting-1.0.7-cp312-cp312-win_amd64.whl
```

Figur B.1: Sida ett av README.md.

> **Note:** Update the command to match the correct version and Python version.
> Remember to update `requirements.txt` and replace the `.whl` file in the `wheels/` directory.

How to Use TIA AutoGen

1. Open and edit `config.py` to specify paths and settings.
2. Ensure that your TIA Portal Project uses the same multilingual settings.
3. **Close** TIA Portal before running the script (it cannot be open simultaneously).
4. Run export of the source project by selecting `Export`.
5. Run export of the library project by selecting `Export Library`.
6. Update `DATA.xlsx` with your custom data structure.
7. Run import by selecting `Import`.
8. Clean up with `Delete all temporary files`.

Editing `config.py`

All configuration settings for the program are located in the `config.py` file.
This is where you define file paths, version settings, and runtime options. Before running the program, make sure the TIA Portal project is closed.

Below is a breakdown of the parameters, including what must be set and what is optional.

Required Settings

- `temp_file_path`: Path where temporary files will be stored during runtime. Missing folders will be created automatically.
- `project_file_path`: Full path to your TIA Portal project file (`.ap19`)
- `library_file_path`: Full path to your TIA Portal library file (`.ap19`)

Optional Settings

- `version`: Target TIA Portal version (e.g., `"19.0"`). Must be installed on your system.
- `portal_mode_ui`: Set to `True` to open TIA Portal with GUI, or `False` to run in background (headless mode).
- `keep_tia_portal`: If `True`, TIA Portal will remain open after script execution.
- `keep_folder_structure`: If `True`, original folder structure will be preserved during export/import.

Project Structure

The project is organized as follows:

Figur B.2: Sida två av README.md.

`config.py`

User settings: file paths, debug mode, etc.

`run.py`

Starts execution, handles CLI prompts

`wheels/`

Location of wheel files (.whl) (TIA Scripting Python)

`src/tia_client`

TIA Portal interaction functions (import/export)

`src/user_interface`

Excel generation for user input

`src/utils`

General helper functions

`src/main.py`

Main entry point for logic

Figur B.3: Sida tre av README.md.

INSTITUTIONEN FÖR ELEKTROTENIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS