Utveckling av en XML-kompilator för automatisk generering av användarmanualer

Matias Miller

Examensarbete

Informationsteknik

2020

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	7326
Författare:	Matias Miller
Arbetets namn:	Utveckling av en XML-kompilator för automatisk generering av användarmanualer
Handledare (Arcada):	Jonny Karlsson
Uppdragsgivare:	Semantix Oy

Sammandrag:
Semantix är nordens största översättningsbyrå som erbjuder både tolk- och översättningstjänster för alla språk i världen. Semantix erbjuder också tekniskt skrivarbete för ABB:s användarmanualer för generatorer och motorer. Till användarmanualprocessen används en XML-kompilator som genererar projektspecifika manualtexter till den slutliga dokumentationen. Kompilatorn skapar manualtexter av XML-filer på basis av vilka konfigurationer som har valts på den elektroniska beställningsblanketten. Hittills har det använts en kompilator som är utvecklad för ca. 20 år sedan. Examensarbetet beskriver hur en ny XML-kompilator har utvecklats för att ersätta den gamla.

Syftet med den praktiska delen av examensarbetet är att utveckla en ny XML-kompilator med hjälp av diverse utvecklingsverktyg och externa bibliotek.
I den teoretiska delen beskrivs själva utvecklingen och motiveras vilka val som gjorts gällande programmeringsspråk och verktyg. Den teoretiska delen beskriver också testningen samt resultatet av arbetet.

Produkten har utvecklats som en skrivbordsapplikation med hjälp av Python och dess inbyggda användargränssnitt. XSLT (Extensible Stylesheet Language Transformations) har använts för att transformera och manipulera innehållet på XML filerna för att åstadkomma önskat resultat.

Som resultat har uppdragsgivaren fått en ny XML-kompilator som uppfyller kraven och förväntningarna. Den nya produkten används dagligen i arbetet och den har betydligt försnabbat och smidiggjort arbetsflödet.

Nyckelord:	XML, Python, XSLT, Skrivbordsapplikation
Sidantal:	32
Språk:	Svenska
Datum för godkännande:	31.5.2020

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	7326
Author:	Matias Miller
Title:	Development of an XML-compiler for automatic generation of user manuals
Supervisor (Arcada):	Jonny Karlsson
Commissioned by:	Semantix Oy

Abstract:
Semantix is the largest translation agency in the Nordic region and offers translation services for any language imaginable. Semantix also provides technical writing for ABB and their generator and motor user manuals. The user manual process uses an XML-compiler that generates project specific manual texts that can be used in the final documentation. The compiler creates these manual texts from XML-files based on which configurations have been selected on an electronic order form.
So far Semantix has used an old compiler that was developed nearly 20 years ago, this thesis describes how a new XML-compiler has been developed to replace the old one.

The practical part describes how the XML-compiler is developed using various development tools and external libraries. The theoretical part describes the actual development and motivates which choices have been made regarding programming language and tools.

The product has been developed as a desktop application using Python and its built-in user interface library. XSLT (Extensible Stylesheet Language Transformations) has been used to transform and manipulate the content of the XML files in order to achieve the desired result.

As a result of this thesis, the outsourcer has gotten a new XML-compiler that fulfills all the requirements and expectations. The new product is used daily and it has significantly sped up the workflow.

Keywords:	XML, Python, XSLT, Desktop application
Number of pages:	32
Language:	Swedish
Date of acceptance:	31.05.2020

INNEHÅLL

FIGURER

1 INLEDNING

Semantix Oy fungerade som uppdragsgivare för detta examensarbete. Semantix är en översättningsbyrå som erbjuder både tolk- och översättningstjänster för alla språk i världen. Uppdragsgivaren erbjuder också tekniskt skrivarbete av användarmanualer för generatorer och motorer. Hittills har det använts en kompilator som utvecklats in-house år 2001. Kompilatorn används för att generera skräddarsydda manualtexter för ABB:s generatorer och motorer. Den gamla kompilatorn är bara kompatibel med äldre versioner av Windows, den måste köras genom en virtualmaskin.

Kompilatorn är långsam och användarupplevelsen är inte bra. Användningen av kompilatorn kräver att man manuellt editerar och flyttar omkring filer vilket är tidskrävande och speciellt för personer som använder den för första gången så är den inte användarvänlig.

Kompilatorn har också flera brister vilket leder till att man måste manuellt gå in i XML-filer och editera innehållet, vilket ofta leder till att man i misstag tar bort överloppselement och XML kodtaggar som inte borde tas bort.

Den gamla kompilatorn är också en säkerhetsrisk i och med att virtualmaskinens virusskydd är föråldrat och en gammal version av Windows XP är mycket sårbar mot diverse attacker.

Examensarbetet beskriver utvecklingen av en XML-kompilator som skapar projektspecifika användarmanualtexter i XML och PDF form för ABB:s generatorer och motorer från en master XML-filkatalog. Kompilatorn skapar två avsnitt (**Section 6 – Manual** och **Section 7 – Accessory Information**) av den fullständiga användarmanualen genom att kombinera XML filer och manipulera innehållet på dem. Beroende på vilka maskinkonfigurationer som anges i samband med en ny beställning på ett elektroniskt beställningsformulär, så ser det slutliga innehållet i output XML filerna annorlunda ut.

XML masterfilerna är uppbyggda på ett sånt sätt att kompilatorn kan gömma kapitel och avsnitt i texterna med hjälp av XML attribut för att manipulera och ändra innehållet på den kompilerade XML filen.

Efter att kompilatorn har sammanställt dessa avsnitt till två slutliga XML filer, kan dessa XML filer omvandlas med hjälp av kompilatorn till en PDF fil som används i den slutliga användarmanualen. Sedan kan användarmanualerna slutföras genom att manuellt lägga till CAD-ritningar, tekniska specifikationer och testrapporter.

1.1 Syfte och målsättningar

Syftet med den praktiska delen av examensarbetet har varit att utveckla en ny kompilator som inte bara gör hela användarmanualprocessen mera automatiserad utan också förenklar och försnabbar arbetsflödet.
Målet är att nya kompilatorn också fungerar på nyaste versioner av Windows och har ett lätt användargränssnitt med en låg inlärningskurva för de som använder programmet för första gången.
Målet med den teoretiska delen är att beskriva grundligt hur den nya kompilatorn har utvecklats och att motivera de val som gjorts gällande programmeringsspråk, verktyg och funktionalitet.
Den teoretiska delen beskriver även slutresultatet av produkten, det vill säga hur den har testats, hur bra den fungerar i produktion och vad som eventuellt kunde ha förbättras eller gjorts annorlunda i den slutliga produkten.

1.2 Metoder för utveckling och testning

Produkten har utvecklats i Python (Python 2019) med hjälp av Pythons inbyggda Tkinter användargränssnitt (Tkinter 2019). Python var ett motiverat och bra val eftersom en skrivbordsapplikation krävdes.
Python är ett bekvämt kodspråk med öppen källkod och flera moduler samt externa bibliotek som underlättar utvecklandet, speciellt för att hantera XML-filer. Python har simpel kodsyntax, bra textbearbetningsverktyg och förmågan att fungera på flera olika operativsystem vilket gör det till ett idealiskt kodspråk för att utveckla skrivbordsapplikationer. En annan faktor som styrde valet mot Python var Tkinter GUI modulens omfattande dokumentation och rätt så låg inlärningskurva.
Den slutliga produkten har även testats genom att prova igenom alla möjliga konfigurationer som finns på den elektroniska beställningsblanketten och kompilera dessa testbeställningar för att se om det önskade slutresultatet uppnås. Slutprodukten har även testats i produktion för att observera hur den fungerar i det vardagliga arbetsflödet, för att se ifall det är något som behöver förbättras eller korrigeras.
Efter testningen har slutprodukten validerats och accepteras av uppdragsgivarens applikationsutvecklings team.

1.3 Struktur

Examensarbetet är strukturerat på följande sätt:
I kapitel 2 tas upp behovet av den nya kompilatorn och varför den ursprungligen utvecklades i samband med examensarbetet.
I kapitel 3 gås igenom utvecklingsverktygen samt ramverken och biblioteken som valts i samband med utvecklingen av produkten, och varför just dessa valts för projektet.
I kapitel 3 gås också igenom vad XML och XSLT är och hur det har använts i mitt arbete, här gås också igenom vilka verktyg som använts i samband med XSLT- och PDF-transformationen.
Själva utvecklingen av produkten beskrivs i Kapitel 4, det vill säga hur den har utvecklats och vilka val som gjorts i samband med utvecklingen. I kapitel 5 utvärderas slutprodukten och vad som kanske kunde ha gjorts bättre eller på något annat sätt. Kapitel 6 innehåller slutledningen av arbetet.

2 BEHOVET AV EN NY XML-KOMPILATOR

XML-kompilatorn skapades ursprungligen för att tillgodose behoven hos kunder, i bästa fall var det tio olika företag. Sedan dess har kompilatorutvecklingen haltat och många kunders behov har blivit mer komplexa. ABB är den enda kunden som har fått sina behov uppfyllda med den gamla kompilatorn.

Behovet av en ny kompilator blev uppenbart när uppdragsgivarens IT-avdelning konstaterade att den är en säkerhetsrisk eftersom den används i en mycket gammal Windowsmiljö och produkten är så instabil. Den virtuella miljön som kompilatorn används på gör användarupplevelsen mycket osmidig och långsam.

Eftersom det fanns ett kundbehov för kompilatorn, räknade uppdragsgivaren ut vad moderniseringen och uppdateringen av kompilatorn skulle kosta och om det skulle vara möjligt överhuvudtaget. Frågan ställdes till några olika programmerings- och dokumentationsföretag men produkten var ny för alla, ingen hade sätt något liknande. Det var till och med överraskande att företagen berömde hur bra och smart funktionalitet kompilatorn har, även om den är nästan 20 år gammal. Att förklara kompilatorns funktioner och hur den används i arbetet blev dock svårt, i och med att användarmanuals arbetsprocessen är mycket skräddarsydd och svår att förklara för utomstående.

Med denna information, tillsammans med IT-avdelningen, bedömdes det att det är vettigt att utveckla produkten i samband med examensarbetet, för då är slutresultatet exakt den som passar behoven. Den största orsaken till att vi valde programmera en egen kompilator var att vi visste exakt hurdan funktionalitet och vad som behövs för att åstadkomma en perfekt produkt som fungerar på ett förväntat sätt.

3 UTVECKLINGSVERKTYG

3.1 Python

Python är ett programmeringsspråk som grundades av Guide van Rossum i slutet på 1980 talet. Enligt TIOBE indexen (Tiobe 2020) rankas Python för tillfället som det tredje populäraste programmeringsspråket i världen, efter Java och C. Python påstås vara ett simpelt och kraftigt programmeringsspråk. Python som programmeringsspråk fokuserar på att ha simpel syntax och struktur, så att programmeraren kan fokusera mera på att lösa problemet än att försöka implementera och förstå kodsyntaxen.
Python används i nästan alla områden i mjukvaruindustrins värld. Pythons inbyggda bibliotek har lett till att programmeringsspråket används ofta för datavetenskap och webbutveckling, men i och med språkets extensiva dokumentation och kraftiga bibliotek, kan man använda Python för nästan vad som helst.

Python användes för detta projekt i första hand för att utvecklingen av kompilatorn skedde på både Windows och MacOS. Python gör det lätt att utveckla plattformsoberoende, koden är kompatibel och ingen extra konfiguration behövs. Det är en stor chans att kompilatorn kommer att utvecklas vidare i framtiden, och därför var Python som ett växande programmeringsspråk ett lätt val.
För att Python är så lätt att läsa och förstå, gör det också lättare för programmerare som vill fortsätta utveckla den att begripa vad som händer i koden. Dessutom är den lättlästa koden och syntaxen lätt att ändra, omarbeta och optimera om det behövs göras ändringar eller uppdateringar.

3.2 Python- bibliotek

Biblioteken som användes i detta projekt valdes på basis av deras användbarhet och prestanda. Dessutom var det viktigt att få en kraftig plattform för skrivbordsapplikationen, samt bra verktyg för hantering av XML filer och skapande av en utdelnings- och körbar slutprodukt.

3.2.1 Tkinter och ttkthemes

Python erbjuder flera alternativ för att utveckla applikationer med grafiska användargränssnitt. Av alla GUI bibliotek är Tkinter den vanligaste och mest använda. Tkinter är standard Python-gränssnittet (Lundh 1999) vilket är orsaken till dess popularitet, Tkinter kräver inte heller någon installation. Tkinter har varit relevant inom Python från början av 2000 talet, så biblioteket har en mycket extensiv dokumentation och aktiv gemenskap, vilket leder till att man lätt hittar svar på problem på internet.

För detta projekt valdes Tkinter specifikt för att den är plattformsoberoende och för att en skrivbordsapplikation krävdes. Tack vare att Tkinter är så väldokumenterat var det inte svårt att komma igång med utvecklingen.
Tkinters starka sida har aldrig varit styling, men nya versioner av Python tillåter integration av ttkthemes biblioteket, vilket gör att man kan lätt ändra stilen och temat på applikationen till ett nästan helt skräddarsytt användargränssnitt som passar den nativa skrivbordsmiljön.

3.2.2 lxml

lxml är ett externt Python bibliotek vilket gör det lätt att hantera XML filer, lxml erbjuder stöd för XPath, XML Schema, XSLT och mera. De viktigaste fördelarna med detta bibliotek är att det är mycket användarvänligt, extremt snabbt när man analyserar stora XML dokument. Dessutom är lxml mycket väl dokumenterat och stöder enkel konvertering av XML data till Python datatyper, vilket resulterar i enkel filmanipulation.
I detta projekt användes lxml i förstahand för att kombinera XML filer samt mata in olika XML element, attributer och användarinmatningar i XML dokument.

3.2.3 cx_Freeze

cx_Freeze är det mest populära externa Python bibliotek för att förvandla ett Python skript till en körbar (.exe) fil. Vanligtvis, för att dela ett Python program kommer mottagaren att behöva ha samma version av Python tillsammans med alla moduler som används. Det här är ju naturligtvis ingenting som borde krävas, och därför har cx_Freeze använts för distribution

av detta projekt. cx_Freeze gör att användaren av kompilatorn inte behöver något extra installerat, utan hen kan använda applikationen direkt.

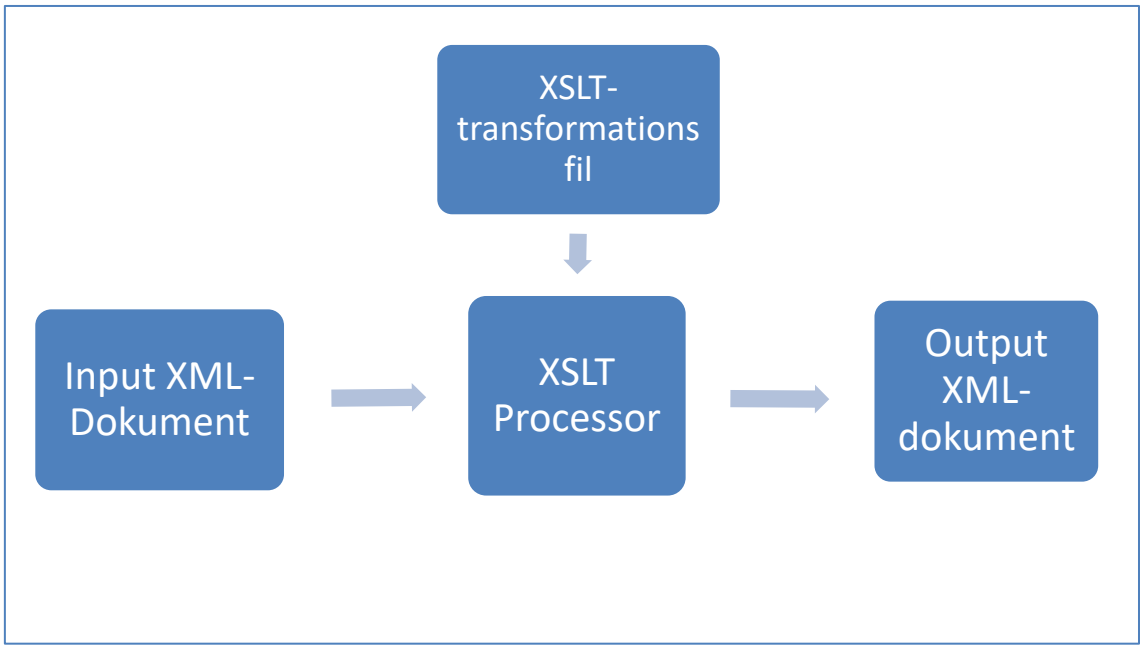
3.3 XML

Extensible Markup Language (XML 2019) är ett markeringsspråk som används för att beskriva data i ett format som är både mänskligt- och maskinellt läsbart. XML:s mål är att vara ett enkelt, användbart. Eftersom verktyg för att skriva och läsa XML finns på alla programmeringsspråk, har det blivit populärt att överföra data i XML-format mellan program och nätverk.
Den grundläggande idén med XML är element som definieras av taggar. Ett element har en börjatagg och en sluttagg, och dessa element är inkapslade i ett så kallat rotelement. Detta gör att XML stöder hierarkiska strukturer som gör det lätt att hålla reda på texter och data i stora XML-dokument. Ett XML-dokument är helt enkelt bara textdata och XML taggar är inte fördefinierade utan den som skapar filen måste definiera själv taggarna.

Masterfilerna som används i ABB:s användarmanual kompilationsprocess är i XML format. Filerna är ursprungligen byggda i XML för att det ska vara lätt att manipulera och uppdatera innehållet. Masterfilerna uppehålls på sju olika språk och XML filformatet gör det lätt att översätta filerna från översättningsminne med översättningsverktyg.

3.4 XSLT

Extensible Stylesheet Language Transformations, XSLT (Holman 2017) är ett språk som ger möjligheten att automatiskt omvandla ett XML-dokument till ett annat XML-dokument. Originaldokumentet ändras inte, utan XSLT-transformationen skapar ett nytt XML-dokument på basis av innehållet på inmatningsdokumentet och vad som man vill att ska ändras.
XSLT kan också användas för att omvandla XML dokument till en annan filtyp, till exempel HTML.
XSLT transformationer kan köras på flera olika sätt, det normalaste sättet är att använda sig av en utvecklingsmiljö eller textredigerare som har en inbyggd XSLT processor. Det finns också flera fristående XSLT processorer som ger möjlighet att köra en transformation inne i kod eller via terminalen som en kommandorad. För en bildrepresentation hur en XSLT transformation fungerar, se figur 1.



Figur 1 Bildrepresentation på hur en XSLT transformation fungerar.

3.4.1 Saxon

Som XSLT processor för projektet användes Saxon (Saxon 2020), vilken är bland de bästa XSLT processorerna enligt Rick Jellifes recension över XSLT processorer (Jelliffe 2017). Saxon var ett bra och lätt val för att nyaste versionen av mjukvaran ger möjligheten att använda sig av XSLT 3.0 vilket behövs för att köra vår XSLT transformationsfil.

3.5 RenderX XEP

RenderX XEP (RenderX 2020) är ett kraftigt verktyg som renderar XML till PDF och formaterar PDF filen enligt användarens konfigurationsfiler. I detta kompilatorprojekt användes RenderX XEP för att programmet användes tidigare i samband med arv-kompilatorn och alla styling- och formateringsfiler är samma. Det fanns ingen orsak att uppdatera och använda ett annat verktyg för PDF renderingen, i och med att transformationen fungerar fortfarande felfritt med den nya kompilatorn.

4 DEN NYA XML-KOMPILATORNS UTVECKLING

4.1 Kravspecifikation

De tekniska kraven som ställdes på kompilatorn var följande:

- Måste kunna kompilera identiska XML och PDF filer som den gamla kompilatorn, på basis av den elektroniska beställningsblanketten.
- Måste ha funktionalitet att kompilera manualer på alla masterspråk.
- Ska notifiera användaren om språk-XML filerna är utdaterade (jämfört med engelska filerna), eller om det inte finns någon specifik accessoar på en av de främmande master-språken.
- Måste ha bra felhantering, det vill säga om någon XML fil har ett tagg- eller syntaxfel ska kompilatorn notifiera användaren i vilken XML fil och på vilken kod-rad felet befinner sig.
- Ska fungera på Windows 7 och nyare operativsystem, med en lätt installationsprocess.
- Skrivbordsapplikation.

Kraven för hur applikationen ska fungera från användarens synvinkel var följande:

- Användaren ska inte behöva installera något annat än själva kompilatorn, installationsprocessen ska vara så simpel som möjligt.
- Användaren ska själv kunna ändra masterfilernas sökväg.
- Kompilatorns användargränssnitts färgschema skall stämma överens med uppdragsgivarens riktlinjer.

Kraven som ställdes på kompilatorn var viktiga att följa för att få en bra slutprodukt som kan användas i produktion.

4.2 Utvecklingsprocessen

Detta kapitel beskriver utvecklingsprocessen av kompilatorn. Kompilatorskriptet består av en enda Python-fil som innehåller en klass, som i sin tur innehåller programmets all funktionalitet.

4.2.1 Användargränssnittet

Utvecklingen av kompilatorn började med skapande av det grafiska användargränssnittet.

Fönstret för applikationen skapades med hjälp av ttkthemes biblioteket för att få bättre styling alternativ för programmet och dess komponenter.
Frame och LabelFrame är Tkinter modulens enkla behållarwidget. Huvudsakliga syftet och användningen av dessa är att fungera som en behållare för olika komponenter, till exempel knappar och fält. Skillnaden mellan LabelFrame och Frame är att LabelFrame kan visa en rubrik ovanför ramen.
I kompilatorskriptet finns det fyra Frames, som fungerar som behållare för programmets olika komponenter. För att få ramarna på plats användes Tkinters grid funktion. Grid sätter ramarna i ett ”tvådimensionellt bord” som är delad i rader och kolumner vilket gör det lätt att placera dem var man vill. Se figur 2 för en bildrepresentation hur behållarna är placerade i applikationen.

Figur 2 Kompilatorfönstrets fyra Frames

Nästa steg i utvecklingen var att fylla ramarna med komponenter. Komponenterna (Radioknapparna, knapparna, textfälten) skapades med hjälp av en enkel for-loop för att inte behöva göra varje komponent skilt, detta både försnabbande processen och gör koden lättare att läsa. Ifall man senare vill lägga till en ny komponent så går det enkelt. Som exempel hur radioknapparna implementerades i kod med hjälp av en for-loop, se figur 3. Först definierades knapparnas värden och positioner i en Python tuple, dvs en samling av objekt som har en oföränderlig och bestämd ordning. Efter detta kan knapparna skapas med att loopa igenom tuplen och föra in värden i knapparna. De andra komponenterna i applikationen skapades med samma princip.

```
#Define radiobuttons
self.radioButtons = (  # Text, Variable, Value, Column, Row
('EN', self.v, 1, 0, 0),
('DE', self.v, 2, 0, 1),
('ES', self.v, 3, 0, 2),
('FI', self.v, 4, 0, 3),
('FR', self.v, 5, 1, 0),
('IT', self.v, 6, 1, 1),
('PT', self.v, 7, 1, 2),
('RU', self.v, 8, 1, 3),
('TR', self.v, 9, 2, 0),
('HU (Only S7)', self.v, 10, 2, 1),
('EL (Only S7)', self.v, 11, 2, 2)
)

#Create radio buttons with loop
for _Text, _Variable, _Value, _Column, _Row in self.radioButtons:
    _Radio = ttk.Radiobutton(radiobuttongroup, text = _Text, variable = _Variable, value = _Value, style='TRadiobutton')
    _Radio.grid(column = _Column, row = _Row, padx = (0, 10), pady=2, sticky= "w")
```

Figur 3 Skapandet av radioknappar med for loop

Tabellen som listar upp XML-filerna som ska inkluderas i kompileringen skapades med hjälp av Treeview komponenten. Treeview visar en hierarkisk samling av objekt, det vill säga en tabell. Tkinters omfattande dokumentationen gjorde det smärtfritt att skapa Treeview tabellen och placera den in i den definierade ramen.

4.2.2 Funktioner

Kompilatorskriptet består av flera funktioner som är bundna till knapparna i användargränssnittet. Det finns dock en funktion som körs genast då man öppnar applikationen. populatePath() funktionen använder sig av pandas (Pandas 2020) biblioteket för att läsa in ”path_configuration.xslt” filen som innehåller alla sökvägarna till masterfilerna. Funktionen läser in sökvägarna in i en Python Dictionary, som sedan kan utnyttjas för att spara sökvägarna i variabler så att de kan användas i applikationen.

split_line() funktionen manipulerar input-strängen som vi får från beställningsblanketten. Funktionen tar helt enkelt bort oönskade ord och strängar och lägger till ett mellanrum mellan varje relevant ord i strängen. Efter manipulationen lägger funktionen de separerade strängarna i en Python lista.

find() funktionen tar in XML-filnamnen som argument och lägger till filerna till tabellen enligt det språk som användaren har valt. Funktionen jämför editeringsdatum på filerna för att kontrollera om språkversionerna av filerna är utdaterade, jämfört med de engelska masterfilerna. Funktionen använder sig av sökvägarna som lästes in från excel filen i

populatePath() funktionen, och kollar om filnamnen som matats in finns i sökvägen i fråga. Ifall filerna finns inne i katalogen så jämför funktionen datumen med att först konvertera datumen från en sträng till ett tidobjekt så att datumen kan jämföras med varandra. Efter detta går funktionen igenom varje filnamn som matats in och sätter dem in i tabellen med rätt ”tagg” konfiguration, till exempel om en språkfil råkar vara utdaterad så sätts den in som utdaterad och den får en skild färg och text konfiguration i tabellen.

För en översikt hur find() funktionen ser ut i kod, se figur 4 nedan.

```
def find(self, filename):

    #Assign radiobutton language choice
    choice = self.v.get()
    label = self.languages[choice]

    S7path = ''.join(label[0])

    if os.path.exists(S7path+filename):

        #Check if language version is older than english version, with minimum 1 day difference
        languageVersionTimeStamp = time.strftime('%m/%d/%Y', time.gmtime(os.path.getmtime(S7path+filename)))
        englishVersionTimeStamp = time.strftime('%m/%d/%Y', time.gmtime(os.path.getmtime(self.ENpath+filename)))

        #Convert timestamps to datetime so we can compare the dates
        langVerTime = datetime.strptime(languageVersionTimeStamp, '%m/%d/%Y')
        enVerTime = datetime.strptime(englishVersionTimeStamp, '%m/%d/%Y')

        #Insert the correct file and change colour tag correctly in Treeview table.

        #Insert with "OK" tag if dates are equal.
        if langVerTime == enVerTime:
            self.filelist.append([filename, ''.join(label[2])])
            self.tree.insert('', 'end', values=[filename, ''.join(label[2])], tags=('OK',))
            print("Language version has same date - OK.")
            self.accessoryCounter +=1

        #Insert with "OK" tag if language version is newer.
        elif langVerTime > enVerTime:
            self.filelist.append([filename, ''.join(label[2])])
            self.tree.insert('', 'end', values=[filename, ''.join(label[2])], tags=('OK',))
            print("Language version is newer - OK.")
            self.accessoryCounter +=1

        #Insert with "outdated" tag if language version timestamp is older than the english master version.
        else:
            self.filelist.append([filename, ''.join(label[2])])
            self.tree.insert('', 'end', values=[filename, ''.join(label[2])+ " > Outdated"], tags=('outdated',))
            self.ziplist.append(filename)
            print("Language version is atleast 1 day older - Outdated.")
            self.accessoryCounter +=1

    #Insert english version with "ENversion" tag incase the language version of the file can't be found.
    elif os.path.exists(self.ENpath+filename):
        self.filelist.append([filename, "EN"])
        self.tree.insert('', 'end', values=[filename, "EN > English version"], tags=('ENversion',))
        self.ziplist.append(filename)
        print("Language version not available, English version inserted instead.")
        self.accessoryCounter +=1
```

Figur 4 find() funktionen.

remove() funktionen hjälper användaren att ta bort filer från tabellen efter att de har satts in. Funktionen plockar ut tabellens objektindex och jämför den med indexet av objekten i listan av filer, och ifall användaren vill ta bort ett av filnamnen i listan så kallas denna funktion då användaren trycker på ”REMOVE SELECTED ACCESSORY FROM THE LIST” knappen.

zip_file() funktionens uppgift är att samla alla utdaterade XML-filer i ett .zip paket för användaren. Ifall användaren kompilerar till exempel en manual på spanska, och tre av accessoar XML-filerna inte är översatta till spanska, kan användaren snabbt och smärtfritt göra ett .zip paket av dessa tre filer. Funktionen använder sig av Pythons inbyggda zipfile modul som gör det lätt att göra ett skapa .zip paket.

4.2.3 Kompilering av ”S7 – Accessory Information” avsnittet

Kompileringen av ”Accessory Information” avsnittet är en simpel process med hjälp av lxml.etree modulen. creates7XML() funktionen kombinerar alla XML-filerna som listats i tabellen under en gemensam <book> XML tag. Funktionen för också in användarens inmatade produktnummer under en <productnumber> tag, som syns i sidhuvudet av den slutliga PDF filen. Titelnamnet matas in automatiskt baserat på vilket språk som valts.
Funktionen söker fram rätt filer från sökvägen, beroende på vilket språk användaren håller på att kompilera och sparar den skapade XML-filen i katalogen som matas in i funktionen, för en översikt av källkoden för createS7XML() funktionen, se figur 5.

```
def createS7XML(self, directory):

    choice = self.v.get()
    label = self.languages[choice]
    S7path = ''.join(label[0])

    #Insert language into XML file so that during PDF creation RenderX knows what language is being transformed.
    booklanguage = ''.join(label[2])
    bookroot = etree.fromstring("<book lang='%s'></book>" % booklanguage.lower())

    #insert elements into the xml
    child = etree.Element('bookinfo')
    productname = etree.Element('productname')
    productname.text = self.productnameentry.get()
    productnumber = etree.Element('productnumber')
    productnumber.text = self.productnumberentry.get()
    title = etree.Element('title')

    #Depending on language, insert correct title
    if booklanguage == "EN":
        title.text = 'Section 7 - Accessory Information'
    elif booklanguage == "DE":
        title.text = 'Abschnitt 7 - Zusatzeinrichtungen'
    elif booklanguage == "ES":
        title.text = "Sección 7 - Información sobre accesorios"
    elif booklanguage == "FI":
        title.text = "Luku 7 - Lisävarusteet"
    elif booklanguage == "FR":
        title.text = "Section 7 - Information d'accessoires"
    elif booklanguage == "IT":
        title.text = "Sezione 7 - Informazioni accessorie"
    elif booklanguage == "PT":
        title.text = "Secção 7 - Informação sobre Acessórios"
    elif booklanguage == "RU":
        title.text = "Раздел 7 - Дополнительное оборудование"
    elif booklanguage == "TR":
        title.text = "Kısım 7 - Aksesuar Bilgileri"
    elif booklanguage == "EL":
        title.text = "Ενότητα 7 - Πληροφορίες για τα εξαρτήματα"
    elif booklanguage == "HU":
        title.text = "7. fejezet - A tartozékokra vonatkozó információk"

    #append elements inside our <book></book> tag
    child.append(productname)
    child.append(productnumber)
    child.append(title)
    bookroot.append(child)

    #loop through our filelist to find the files that we want to combine
    for i in self.filelist:

        filepath = S7path+i[0]
        ENpathWithFile = self.ENpath + i[0]

        #if file cant be found in language version path, use file from EN path
        if os.path.exists(filepath):
            note_root = lxml.etree.parse(filepath).getroot()
        else:
            note_root = lxml.etree.parse(ENpathWithFile).getroot()

        bookroot.append(note_root)
        tree = etree.ElementTree(bookroot)

        #write the file to given directory
        tree.write(directory, pretty_print=True, encoding="utf-8", xml_declaration=True, standalone= True)
```

Figur 5 createS7XML() funktionen.

Användaren har två alternativ då hen vill kompilera S7 – avsnittet. Om det bara behövs en XML fil så kan generateS7XML() funktionen väljas med att trycka på knappen ”GENERATE S7-XML”. Funktionen använder sig av createS7XML() funktionen som förklaras ovan och sparar en output XML fil i en katalog som användaren kan välja själv.

Ifall användaren vill göra både XML och PDF filerna samtidigt så kan hen trycka på ”GENERATE S7 XML & PDF” knappen som gör samma sak som generateS7XML() funktionen men kör ett kommandoradsargument som använder sig av RenderX XEP verktyget för att omvandla XML filen till PDF genom att kalla på den java-drivna XEP funktionen (se figur 6). Kommandoradsargumentet behöver en konfigurationsfil, input fil, XSL-konfigurationsfil och sedan en output fil som i detta fall blir vår färdiga PDF.

```
java com.renderx.xep.XSLDriver "-DCONFIG=%~dp0\..\XEP\xep.xml" -xml "%DOCROOT%\manual_s7.xml" -xsl "%XSLROOT%\driver-manual-abb.xsl" -out "%DOCROOT%\manual_s7.pdf"
```

Figur 7 RenderX XEP kommandoradsargument.

Själva koden för funktionen att skapa XML och PDF filen är rätt så simpel, då användaren trycker på knappen så kombineras filerna i tabellen med hjälp av createS7XML() funktionen och efter att XML-filen är skapad så kallar funktionen på XEP kommandoradsargumentet som befinner sig inne i ”run-xep-abb-s7.bat” filen och som slutresultat får vi en färdigt kompilerad S7-XML och PDF fil, se figur 7 för källkoden.

```
#This function creates both the XML and PDF automatically in the predefined folder "xmltoPDF/abb"
    def generateS7XMLandPDF(self):
        try:
            if len(self.entryfield.get("1.0",'end-1c')) == 0:
                tk.messagebox.showwarning("!", "Insert filestring first!")

            else:
                self.createS7XML("xmltoPDF/abb/manual_s7.xml")
                #Call on the RenderX XEP batch file that is located inside the abb folder
                subprocess.call(r'"xmltoPDF/abb/run-xep-abb-s7.bat"')
                tk.messagebox.showinfo("!", "S7 XML and PDF created successfully.. press OK to open the directory.")

                path=os.path.realpath("xmltoPDF/abb/")
                os.startfile(path)

        except Exception as error:
            tk.messagebox.showerror("!", error)
```

Figur 6 generateS7XMLandPDF() funktionen.

4.2.4 Kompilering av ”S6 - Manual” avsnittet

Kompileringen av manualtexterna för ”S6 -Manual” avsnittet utförs på ett mycket likadant sätt som texterna för accessoarerna i ”S7 – Accessory Information” avsnittet.
Manualtexterna består av tio kapitel som alla har sin egen XML-fil. S6chapters() funktionen kombinerar dessa tio filer in i en XML-fil ”chap1-10.xml” under en gemensam <book> tagg och informationen av produktnumret, produktnamnet och titeln sparas på samma sätt under sina egna taggar i filen som det gjordes i S7 kapitlet. Då ”chap1-10.xml” filen är skapad, så körs createXMLStringForS6() funktionen som skapar en till XML fil vilken innehåller alla konfigurationer från strängen som inmatades från beställningsblanketten.

Då dessa två filer är skapade så körs en XSLT transformation på ”chap1-10.xml”. XSLT transformationen körs med hjälp av Saxon XSLT processorn och transformationen körs med ett java-drivet kommandoradsargument. Transformationen använder sig av ”transform.xsl” filen som läser in konfigurations-värdena från XML-filen som createXMLStringForS6() skapade.
Transformationsfilen kör igenom hela ” chap1-10.xml” och jämför varje XML-tagg som innehåller attributen ”condition”. Ifall condition attributen innehåller en sträng som inte finns i konfigurationssträngen från beställningsblanketten, så tas hela kapitlet med det villkoret bort. Transformationen kan också tolka XML operatörer, i detta fall ”and”, ”not” och ”or”. Transformationen förstår ifall dessa operatörer används inne i condition attributen, ett exempel på en operatör condition kunde vara t.ex **<chapter condition=”not(twin_drive) and AMG**”, i detta fall om AMG strängen finns, men twin_drive finns inte, så inkluderas kapitlet efter transformationen.
Transformationen gör det möjligt att skapa precis det innehållet som kunden vill ha, genom att ta bort de kapitel som inte behövs och lämna in det nödvändiga innehållet. Figur 8 ger en översikt hur XSLT transformationsfilen ser ut i kod.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:axsl="http://www.w3.org/1999/XSL/Transform-alias"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:mf="http://example.com"
    exclude-result-prefixes="#all"
    version="3.0">
    <!--This XSL transformation can be done with Saxon 9.7 HE or newer to transform our XML based on the condition attribute in each element.
    This program loops through the whole XML document and based on the values in the stringforS6.xml, then transforms
    our XML document and removes elements that do not match our conditions. -->

    <!--Read in our values-->
    <xsl:param name="input2" select="doc('stringforS6.xml')"/>
    <xsl:param name="true-values" as="xs:string*" select="$input2/book/values"/>

    <!--define our xpath operators-->
    <xsl:param name="xpath-operators" as="xs:string*" select="'or', 'not', 'and'"/>

    <!--Choose which elements to be transformed, in this case all elements with the attribute "condition"-->
    <xsl:param name="elements-to-be-transformed" select="//*[@condition]"/>
    <xsl:namespace-alias  stylesheet-prefix="axsl" result-prefix="xsl"/>

    <xsl:function name="mf:substitute-variables" as="xs:string*">
        <xsl:param name="expression" as="xs:string*"/>
        <xsl:variable name="substituted-expression" as="xs:string*">
            <!--Use regex "[0-9_\p{L}_][\p{L}_0-9]*" to distinguish our values. -->
            <xsl:apply-templates select="analyze-string($expression, '[0-9_\p{L}_][\p{L}_0-9]*')" mode="substitution"/>
        </xsl:variable>
        <xsl:sequence select="string-join($substituted-expression)"/>
    </xsl:function>

    <!-- Here we check whether our values are true or false-->
    <xsl:template match="*:match[not(. = $true-values)]" mode="substitution">
        <xsl:text>false()</xsl:text>
    </xsl:template>

   <xsl:template match="*:match[not(. = $true-values) and . = $xpath-operators]" mode="substitution">
      <xsl:sequence select="."/>
   </xsl:template>

    <xsl:template match="*:match[. = $true-values]" mode="substitution">
        <xsl:text>true()</xsl:text>
    </xsl:template>

    <!--if they match, that element gets included, else they will be deleted-->
    <xsl:template match="*[@condition]" mode="template">
        <axsl:template match="{node-name()}[@condition = '{@condition}']">
            <axsl:if test="{mf:substitute-variables(@condition)}">
                <axsl:next-match/>
            </axsl:if>
        </axsl:template>
    </xsl:template>

    <xsl:mode on-no-match="shallow-copy"/>

    <xsl:output method="xml" indent="yes"/>

    <xsl:template match="/">

        <xsl:sequence
            select="transform(
                      map {
                        'source-node' : .,
                        'stylesheet-node' : $stylesheet
                      }
                    )?output"/>
    </xsl:template>
</xsl:stylesheet>
```

Figur 7 transformation.xsl filen.

Efter att transformationen är färdig skapas en ny XML-fil som sedan kan genereras till en PDF på samma sätt som S7 kapitlet, det vill säga med hjälp av XEP verktyget.

4.3 Skapandet av en körbar fil

För att skapa en körbar och utdelningsbar fil av Python skriptet så användes cx_freeze biblioteket. cx_freeze kräver en konfigurationsfil där man definierar filen som ska köras till en .exe och om det krävs några tilläggsfiler. I detta fall för att applikationen är byggd med Tkinter så kräver cx_freeze att man inkluderar tkinter mappen och några andra tilläggsfiler, tkinter mappen och diverse filer hittas inne i Python installationskatalogen.

Efter att konfigurationsfilen är färdig så körs den med kommandot ”python setup.py build” och och då skapas en ny ”build” katalog som innehåller den körbara filen och alla stödjande filer som krävs för att applikationen kan köras. För en översikt av cx_freeze konfigurationsfilen, se figur 9.

```
#Setup script to create an executable out of our 'compiler.py' Tkinter project.
#Run the build process by running the command 'python setup.py build'
#If everything works well, the 'build' subdirectory should contain everything needed to run the application.

import sys
from cx_Freeze import setup, Executable
import os

base = None

build_exe_options = {"include": ["../Python/Python36-32/Lib/tkinter"], "include_files": ["../Python/Python36-32/DLLs/tcl86t.dll", "../Python/Python36-32/DLLs/tk86t.dll"]}
executables = [
    Executable('compiler.py', base=base, icon="assets/icon.ico")
    ]

setup(name='Compiler',
      version='1.0',
      description='Compiler',
      executables=executables
      )
```

Figur 8 cx_freeze konfigurationsfil.

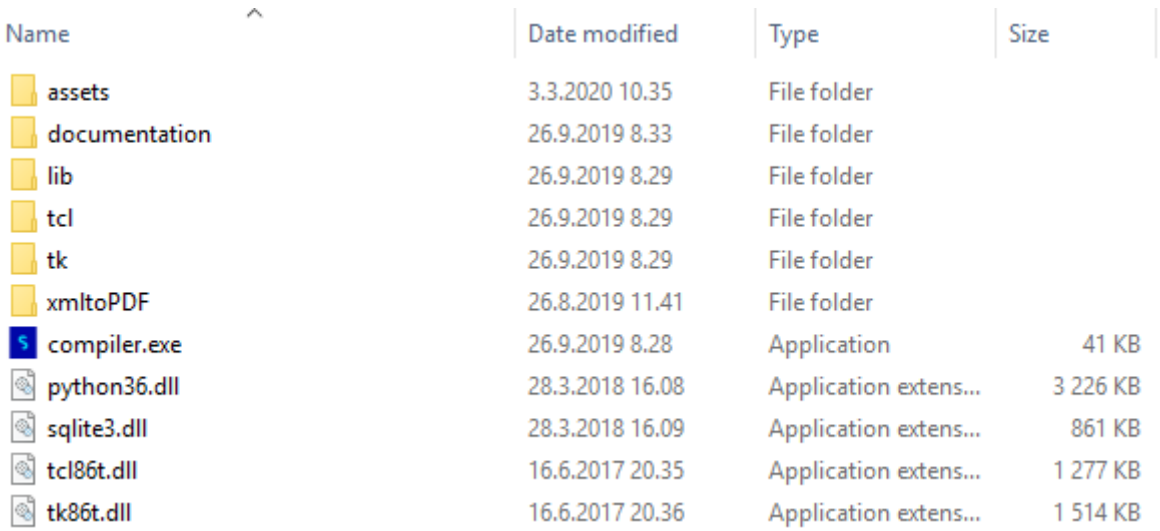
5 RESULTAT

5.1 Kompilatorn från användarens synvinkel

Detta kapitel går igenom användarupplevelsen och vilka steg som användaren behöver ta för att få kompilatorn att fungera på sitt eget system samt hur en exempel kompilering kunde göras med programmet.

5.1.1 Installation

För att användaren kan börja använda kompilatorn behöver hen ladda ner en .ZIP fil och packa upp paketet till en lokal katalog. Efter att .ZIP paketet har packats upp så ser katalogen ut såhär:



Name	Date modified	Type	Size
assets	3.3.2020 10.35	File folder	
documentation	26.9.2019 8.33	File folder	
lib	26.9.2019 8.29	File folder	
tcl	26.9.2019 8.29	File folder	
tk	26.9.2019 8.29	File folder	
xmltoPDF	26.8.2019 11.41	File folder	
compiler.exe	26.9.2019 8.28	Application	41 KB
python36.dll	28.3.2018 16.08	Application extens...	3 226 KB
sqlite3.dll	28.3.2018 16.09	Application extens...	861 KB
tcl86t.dll	16.6.2017 20.35	Application extens...	1 277 KB
tk86t.dll	16.6.2017 20.36	Application extens...	1 514 KB

Figur 9 Compiler katalog innehåll

Den ända konfigurationskontrollen användaren behöver göra är att gå in till **”assets”** katalogen och öppna en Excel-fil som heter **”path_configuration.xslt”,** se figur 11 för innehållet av Excel filen.

	A	B	C
1	Language	Path-S7	Path-S6
2	EN	M:/ABB_FI_(CM)/04_Master/EN/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/EN/
3	DE	M:/ABB_FI_(CM)/04_Master/DE/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/DE/
4	ES	M:/ABB_FI_(CM)/04_Master/ES/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/ES/
5	FI	M:/ABB_FI_(CM)/04_Master/FI/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/FI/
6	FR	M:/ABB_FI_(CM)/04_Master/FR/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/FR/
7	IT	M:/ABB_FI_(CM)/04_Master/IT/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/IT/
8	PT	M:/ABB_FI_(CM)/04_Master/PT/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/PT/
9	RU	M:/ABB_FI_(CM)/04_Master/RU/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/RU/
10	TR	M:/ABB_FI_(CM)/04_Master/_muut_kielet/TR/S7/	M:/ABB_FI_(CM)/04_Master/__CHAPTERS/TR/
11	HU	M:/ABB_FI_(CM)/04_Master/_muut_kielet/HU/S7/	S6 Chapters not translated***
12	EL	M:/ABB_FI_(CM)/04_Master/_muut_kielet/EL/S7/	S6 Chapters not translated***
13			

Figur 10 path_configuration.xslt filens innehåll

Excel-filen hjälper kompilatorn att hålla koll på varifrån diverse XML filer hämtas, och användarens uppgift är att kontrollera att sökvägen till katalogerna är korrekt. Som standard är dessa korrekt, men ifall sökvägarna i framtiden ändras måste dessa uppdateras.

5.1.2 Användarupplevelse

Efter att installationen är färdig kan användaren starta programvaran med hjälp av **”compiler.exe”** filen, se figur 12 för kompilatorns start-vy.

Figur 11 Kompilator start-vy

Kompilatorns användargränssnitt är simpelt och med hjälp av uppdragsgivarens interna arbetsinstruktioner är kompilatorn lätt att använda.

Det första steget är att användaren besöker den elektroniska beställningsblanketten och öppnar det projektet som hen vill kompilera. Beställningsblanketten ger automatiskt en ”konfigurations-sträng” som innehåller den specifika generatorns/motorns konfigurationsuppgifter. Se figur 13 för ett exempel på en konfigurationssträng.

```
proj=abb&out=xml&type=AMZ&type=integroitu_pukkilaakeri&type=DTCsynchro&type=symmetrinen
&type=joustavasti_kiinnitetty_jaahdytinyksikko_EI&type=IC8A6W7&type=default
&type=kokonainen&type=wooden_box&type=with_brushes&type=winding_with_brushes
&type=ZRD70009871781&type=ZRD6ZRC6&type=ZRC80009883058&type=RER10009871414
&type=QLKE&type=PYRR0009873253&type=PYRR0009873253&type=PYRL0009872728
&type=PXBFGH0009873400&type=OXAFLOAT10009871307&type=AGDETNS
```

Figur 12 Konfigurationssträng från beställningsblanketten

Användaren matar in konfigurations-strängen till kompilatorn och fyller i de nödvändiga fälten, efter detta trycker hen på ”FIND FILES” som söker fram accessoar-filerna, enligt den inmatade strängen och listar dem i tabellen, som syns i figur 14.

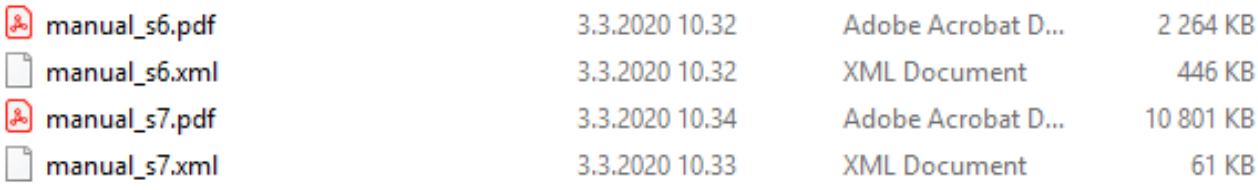
Figur 13 Kompilator vy efter att filerna i strängen sökts

Ifall ett annat språk ska kompileras väljer användaren ett annat språk och trycker på ”FIND FILES” igen för att uppdatera filerna till målspråket. I figuren nedan visas som exempel om man vill kompilera på italienska. Kompilatorn visar vilka accessoarer som hittas på italienska och de som inte hittas på målspråket ersätts med den engelska versionen av filen, se figur 15.

Figur 14 Kompilator vy då man kompilerar en italiensk manual

Efter att användaren har kontrollerat att accessoarerna är rätt och stämmer överens med vad kunden har beställt så kan hen trycka på antingen **”GENERATE S6-XML & PDF”** eller **”GENERATE S7-XML & PDF”**.

Som resultat skapar kompilatorn färdiga XML och PDF filer (Se figur 16) av både Avsnitt 6 (Manualtexterna) och Avsnitt 7 (Accessoarerna) på basis av konfigurations-strängen som användaren gett in. Användaren kan sedan använda dessa filer som en del av den fullständiga användarmanualen.



Figur 15 XML och PDF filerna som kompilatorn skapar

5.2 Testning

Metoden för testning av den slutliga applikationen utfördes i två delar, först den tekniska testningen där det evaluerades och kontrollerades att applikationen fungerar som den ska teknisk och efter detta utfördes användartestningen där applikationens funktioner testades manuellt av en användare.

Den tekniska testningen innehöll kompatibilitetstestning och funktionelltestning. Syftet med kompatibilitetstestningen var att försäkra sig att programvaran fungerar på den hårdvaran och operativsystem som uppdragsgivaren använder. Den funktionella testningen hängde lite ihop med användartestningen, men i den funktionella testningen testades applikationens felhantering och olika undantagsfall som inte inträffar varje dag. I den funktionella testningen testades också att applikationens alla funktioner fungerar felfritt.

Användartestningen gick ut på att helt enkelt manuellt göra testbeställningar som består av alla möjliga konfigurationsalternativ som finns på den elektroniska beställningsblanketten och kompilera dessa testbeställningar för att se om slutresultatet var det som önskades. Efter att applikationen försäkrades att fungera på ett önskvärt sätt både tekniskt och funktionellt så har den även tagits i bruk i produktion för att se hur den presterar i det dagliga arbetsflödet.

5.3 Utvärdering och vidareutveckling

Slutprodukten uppfyller alla krav som ställdes i kravspecifikationen. Skrivbordsapplikationen kompilerar exakt identiska XML och PDF filer som den gamla kompilatorn och kompileringen fungerar på alla masterspråk. Kravet att notifiera användaren om utdaterade eller saknade filer uppfylls också i den nya kompilatorn.

Felhanteringen av XML filerna fungerar bra, ifall någon av XML filerna som används av kompilatorn har syntaxfel så uppmanar kompilatorn användaren exakt i vilken fil och var som felet befinner sig. Kompilatorn fungerar felfritt på Windows 7 eller nyare Windows operativsystem, precis som planerat.

Ur användarens synvinkel så är också alla krav som ställdes i specifikationen uppfyllda. Kompilatorns installationsprocess är simpel och programmet fungerar utan installation av någon extra programvara. Sökvägarna för masterfilerna kan ändras lätt med hjälp av Excel-filen.

Det finns dock plats för vidareutveckling, speciellt i sättet som applikationen distribueras. För tillfället om någon uppdatering till applikationen görs så måste användaren gå och ladda ner hela compiler paketet och ersätta den med det gamla för att få uppdateringen. Man borde kunna skicka ut uppdateringar direkt till användarens applikation utan att hen behöver göra något själv.

6 SLUTLEDNING

I detta arbete har det beskrivits utvecklingen av en ny XML-kompilator för uppdragsgivaren, Semantix Oy, som ersätter den gamla, föråldrade Windows XP baserade kompilatorn som utvecklades för nästan 20 år sedan. Kompilatorn används för att skapa projektspecifika manualtexter för ABBs generatorer och motorer. Kompilatorn skapar dessa texter på basis av maskinkonfigurationer som fås från en elektronisk beställningsblankett. Applikationen skapar färdiga XML och PDF filer som sedan kan används i den slutliga dokumentationen.

Syftet med den praktiska delen av arbetet har varit att utveckla en programvara som gör ett mera automatiserat och förenklat arbetsflöde. Den teoretiska delen beskriver hur kompilatorn har utvecklats och vilka verktyg och val som gjorts i samband med utvecklingen. Målet med den nya kompilatorn har varit att göra en applikation som uppfyller kravspecifikationen, försnabbar arbetsflödet och har en låg inlärningskurva.

Slutresultatet har uppnått förväntningarna och applikationen används dagligen för det som den har utvecklats för. Applikationen har gjort arbetsflödet mycket smidigare och snabbare samt inlärningskurvan för nya användare är mycket lägre än förr. Målen som ställdes för arbetet har uppfyllts bra, både praktiskt och teoretiskt. Applikationen har som sagt låg användartröskel, den fungerar på nyaste Windows versionen och användargränssnittet är lätt och simpelt att använda. Teoretiskt så beskriver arbetet bra och grundligt hur applikationen har utvecklats, vilka verktyg som använts samt resultatet och funktionaliteten av produkten.

För tillfället finns det inga omedelbara planer att förbättra eller vidareutveckla produkten, men troligtvis så kommer uppdateringar och ändringar vara nödvändiga för applikationen i framtiden. Som framtidsvision kunde programvaran göras till en webbaserad applikation som är driven i någon molntjänst. En webbaserad applikation skulle bli av med installationsprocessen och en webb-applikation gör det också lättare att driva uppdateringar till programmet.

KÄLLOR

Python, 2019. Tillgänglig: https://www.python.org/, Hämtad 22.10.2019

XML, 2019, Tillgänglig https://www.xml.com/, Hämtad 23.10.2019

Tkinter, 2019. Tillgänglig: https://docs.python.org/3/library/tk.html Hämtad: 24.10.2019.

lxml, 2019, Tillgänglig: https://lxml.de/ Hämtad 26.10.2019

cx_Freeze, 2019, Tillgänglig: https://cx-freeze.readthedocs.io/en/latest/ Hämtad 24.10.2019

Saxon, 2020, Tillgänglig: https://www.saxonica.com/welcome/welcome.xml Hämtad: 03.02.2020

RenderX, 2020, Tillgänglig: http://www.renderx.com/tools/xep.html Hämtad: 04.02.2020

Jelliffe, R., 2017, Revving the XSLT 1.0 Engines: Are they all the same? Tillgänglig: https://www.xml.com/articles/2017/01/26/revving-xslt-10-engines-are-they-all-same/, Hämtad 22.03.2020

TIOBE, 2020, Tillgänglig: https://www.tiobe.com/tiobe-index/ Hämtad 27.03.2020

Holman, K., 2017, Tillgänglig: https://www.xml.com/articles/2017/01/01/what-is-xslt/ Hämtad 30.03.2020

Pandas, 2020, Tillgänglig: https://pandas.pydata.org/ Hämtad 31.3.2020

Lundh, F., *1999, An Introduction to Tkinter, Tillgänglig:* http://www.tcltk.co.kr/files/TclTk_Introduction_To_Tkiner.pdf Hämtad 30.4.2020