Antti Kupiainen

# Electronic Application for a Summons

## Process Design and C# Library

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

8 April 2020

Abstract

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Antti Kupiainen<br>Electronic Application for a Summons<br>Process Design and C# Library<br>32 pages + 9 appendices<br>8 April 2020 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Professional Major | Software Engineering |
| Instructors | Susanna Havanka, Development Manager<br>Janne Loikkanen, Software Architect<br>Janne Salonen, Head of ICT, Principal Lecturer |

In this final year project, a process design for the creation of electronic applications for a summons from a debt collection system was developed. The project was conducted for a professional debt collection agency in Finland. The debt collection agency was developing a new debt collection system and was using Agile methodologies, so new system features were planned and scheduled according to client needs. Concurrently, changes in the XML schema of the application for a summons were planned by the government.

The project was developed on the .NET Core platform using the object-oriented paradigm. Development was conducted by studying existing implementations and documents offered by the government and carrying out a literature review. The contents of an application for a summons were analyzed with the help of a schema draft received from the government and classes for representation of an application for a summons were designed. Finally, a process for utilizing the classes was designed and the classes were implemented.

The project produced a process description and a C# library which are in production use.

| | |
|---|---|
| Keywords | OOP, XML, software design |

| | |
|---|---|
| Tekijä | Antti Kupiainen |
| Otsikko | Electronic Application for a Summons Process Design and C# Library |
| Sivumäärä | 32 sivua + 9 liitettä |
| Aika | 8.4.2020 |
| Tutkinto | insinööri (AMK) |
| Tutkinto-ohjelma | Information Technology |
| Ammatillinen pääaine | Software Engineering |
| Ohjaajat | Susanna Havanka, Development Manager<br>Janne Loikkanen, Software Architect<br>Janne Salonen, Head of ICT, Principal Lecturer |

Insinöörityössä suunniteltiin prosessi sähköisten haastehakemusten tuottamiseen perintäjärjestelmästä. Työ tehtiin suomalaisessa ammattimaisessa perintätoimistossa. Perintätoimistolla oli meneillään uuden perintäjärjestelmän kehitysprojekti, jossa käytettiin ketteriä ohjelmistokehityksen menetelmiä, joten uudet ominaisuudet otettiin kehitettäväksi asiakkaiden tarpeiden mukaisesti. Samanaikaisesti kehitysprojektin kanssa valtionhallinto suunnitteli sähköiselle haastehakemukselle uutta XML-skeemaa.

Kehitys tapahtui .NET Core -alustalla oliopohjaisia menetelmiä hyödyntäen. Insinöörityössä tutustuttiin olemassa olevien järjestelmien toteutukseen vastaavasta toiminnallisuudesta, tutkittiin valtionhallinnon tarjoamaa dokumentaatiota, ja tutustuttiin kirjallisuuteen. Haastehakemuksen sisältöön tutustuttiin ja kehitettiin luokkia kuvaamaan haastehakemusta. Lopuksi kehitettiin luokille toteutus ja niitä hyödyntävä prosessi.

Insinöörityön tuloksena syntyi tuotantokäytössä oleva prosessikuvaus ja C#-kirjasto.

| | |
|---|---|
| Avainsanat | OOP, XML, Ohjelmistosuunnittelu |

## Contents

Appendices

# 1 Introduction

Digitalization of the public services provides fast and convenient channels for interaction with the authorities, while reducing the burden on public resources. In the judicial system digital services allow citizens and businesses to deliver applications concerning undisputed receivables electronically, thus reducing need of manual labor for courts and litigants alike. As of 1 September 2019, only private individuals may submit applications for a summons for a summary civil case as a paper copy in Finland, apart from exceptional cases. (Legal Register Centre 2019; Ministry of Justice 2019.)

**The Legal Register Centre** offers a web service, where applications for a summons can be filed and their status can be followed. However, this is feasible only when the number of applications is small. For applicants who file large numbers of applications, the Legal Register Centre offers a system called **Santra**, which allows electronic filing via Extensive Markup Language (XML) documents. (Legal Register Centre 2019.)

This thesis project was conducted as a part of a larger project developing a new debt collection system. The aim of this thesis project was to create a process and a C# library for creating electronic applications for a summons in XML from a debt collection system.

## 1.1 Business Context

**Svea Group** is a financial company group offering administrative and financial services and payment solutions. It was formed in Sweden in 1981 and now spans several European countries. Svea Group has over 2,000 employees. In Finland it started in 2002 through corporate acquisitions. (Svea 2020a; Svea 2020b.)

**Svea Perintä Oy** (later Svea Perintä) is a Finnish debt collection agency that is a part of the Svea Group. Svea Perintä employs approximately 60 persons in two offices in Helsinki and Sastamala. Services include payment reminders, voluntary collection, legal collection, and international debt collection. Svea Perintä has both public and private sector clients. (Svea 2020b.)

This thesis was written at the System Development department of Svea Perintä in Helsinki. Svea Perintä is developing a new debt collection system to replace several older systems.

### 1.2 Business Challenge, Objective and Outcome

Svea Perintä is currently in the process of developing a new debt collection system. The new system did not have software for the automatic creation of an electronic application for a summons implemented before the final year project was carried out. Concurrently with Svea Perintä’s development process, the Legal Register Centre was developing a new version of the XML schema for the application for a summons in their Santra system.

*The task given was to design a process for the automatic creation of electronic applications for a summons in XML, incorporating the new features of the new schema.* The outcome of the project is a C# library and a process description for the creation of these applications.

### 1.3 Thesis Outline

This thesis project was conducted by researching the Legal Register Centre’s documentation, proprietary source code and system documentation at Svea Perintä, and by reviewing pertinent literature.

The thesis discusses the electronic application for a summons as defined by the Legal Register Centre of Finland. Because of differences in legislation and governance, the findings may not apply to other countries.

This thesis is divided into 6 sections. Section 1 is an introduction to the thesis. Section 2 discusses the methods and material used for research. Section 3 contains current state analysis and analyzes planned changes. Section 4 introduces available knowledge on the technologies and methodologies used. Section 5 describes how the project was conducted, and section 6 contains summary and conclusion.

## 1.4 Key Concepts

| | |
|---|---|
| Summons | An invitation to the plaintiff to attend court. |
| Application for a summons | A request to the court for a summons to be issued. |
| Collection | A process for recovering amounts owed that are past due date. |

# 2 Methods and Material

This section describes the methods and material used in the project. The first subsection describes the research design for the project, in the second subsection a plan for the project is described, and the third subsection briefly introduces the material collected.

## 2.1 Research Design

The project was conducted in eight phases as shown in the diagram below.

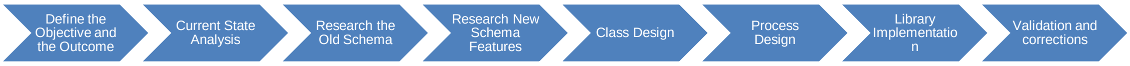


Figure 1. Research Design Diagram

As shown in Figure 1, the project launched with defining the objective and the outcome of the project. The definition of the objective also started a review of literature concerning technologies used in the project. This literature review continued for most of the project’s duration.

After defining the project, current state analysis was conducted. The goal of this analysis was to better understand how similar implementations were designed in the past and what the current situation of the new system in development is.

In the third phase, the old XML schema for an electronic application for a summons was studied. This phase had some overlap with the previous phase, as a part of the current state analysis was studying actual XML files produced by the existing systems.

The fourth phase consisted of studying the new features that the Legal Register Centre was implementing in the new XML schema. The fourth phase continued concurrently with the later phases until the end of the project because the new schema design was not finalized and the Legal Register Centre changed the design several times during the project.

After researching the old schema in use and the new schema, the design for classes representing the elements of an application for a summons was conducted. During and after the class design, a process for gathering and inserting data required for an application was designed and the implementation of the classes in C# began.

After the class and process designs were concluded, the C# library implementation was finalized by creating the methods for the creation of applications represented by the class design. Finally, the resulting process and library were validated and corrections were applied.

## 2.2 Project Plan

The project was conducted in the System Development department of Svea Perintä as a part of a larger debt collection system project. The author, a 4th year Software Engineering student, was instructed by Development Manager Susanna Havanka and Software Architect Janne Loikkanen.

The new debt collection system is already in production use, but as the system project uses Agile methodologies, new features are not implemented until they are required (Sommerville 2015: 50). Initially the objective of the project was intended to be the implementation of the complete application for a summons module, but during the definition of the objective and the outcome, it was soon realized that schedule constraints necessitated assigning the components responsible for collecting the data and triggering the process to other System Development employees.

Some preliminary research was conducted during the summer of 2019, but the objective and the outcome of the project were defined in September 2019, and the research and implementation phases were carried out from September to November 2019. Validation and corrections were mostly conducted in December 2019, apart from some source code documentation revisions done in January 2020.

The project was an actual business project based on the analysis of data from multiple governmental documents. These data sources are described below.

## 2.3 Collected Data

The information concerning the electronic application for a summons is scattered within multiple documents, none of which are comprehensive and some of which are even contradictory. This resulted in a large number of data sources for merely analyzing the information content of the application. These data sources are described in the table below.

Table 1. Documents used in the analysis and design

| | Name of the document | Amount/number of pages | Description |
|---|---|---|---|
| A | SANTRA-JÄRJESTELMÄ Järjestelmän kuvaus ja ohjeita Santra-hakijoille | 13 pages | Description of the Santra system and guidelines for applicants using it. |
| B | XML-muotoisen haastehakemuksen formaatti | 39 pages | Description of the elements in the XML formatted application. |
| C | SantraYleinen.xsd | 1 schema | General schema of Santra elements. |
| D | SantraHakemus.xsd | 1 schema | Schema of Santra elements specific to an application for a summons. |
| E | Hakemusesimerkki.pdf | 2 pages | An example of an application for a summons. |
| F | XMLSantra-koodistoja | 2 pages | Description of certain code values used in applications. |
| G | Tuomioistuintunnukset | 1 page | Listing of code values for district courts. |
| H | XMLSantra-muutoksia 2010 | 4 pages | Description of changes made to the XML schema in 2010. |
| I | Viivästyskorko-koodit/määräytymistavat 2013 | 2 pages | Description of some code values used for interest percentages in the applications. |
| J | Kuluttajaluottouudistus 2019 Info-tilaisuus Santra-hakijoille | 23 pages | Supporting documentation for a briefing held about changes to the application for a summons schema. |
| K | Hakijoiden kysymykset Santra-infossa 26.9.2019 (vastaukset 3.10.2019) | 4 pages | Questions asked at the briefing about changes to the schema and answers to those questions. |

| | | | |
|---|---|---|---|
| L | Kuluttajaluottouudistus, Santra-muutokset | 7 pages | Further information about the changes to the application for a summons schema. |
| M | SantraHakemus.svg | 1 chart | Graphical representation of the schema. |

Some of the documents listed in Table 1 described the application for a summons in general while others focused on specific changes made to the application. Some of the documents are publicly available from the web page of the Legal Register Centre, but a few were at least initially available only to parties implementing changes to the systems that already have the Legal Register Centre's approval for using the Santra system. For example, SantraHakemus.xsd (item D in Table 2) was revised several times during the autumn 2019 but was not available on the Legal Register Centre's web page until February 2020.

The Legal Register Centre also sent several advisories via email and answered some questions. These are shown in Table 2.

Table 2. Email advisories from the Legal Register Centre

| | **Topic** | **Date** |
|---|---|---|
| 1 | Answer to a question about a specific element in an application | 30 August 2019 |
| 2 | Answer to a further question about the element | 30 August 2019 |
| 3 | Initial version of the new schema | 4 September 2019 |
| 4 | Changes to the schema. New schema version included | 25 September 2019 |
| 5 | Items J, K, and L in Table 1 | 3 October 2019 |
| 6 | Changes to the schema. New schema version included | 18 October 2019 |
| 7 | Changes to some code values | 7 November 2019 |
| 8 | Changes to the schema. New schema version included | 28 November 2019 |
| 9 | Changes to the schema. New schema version included | 19 December 2019 |

Many of the advisories seen in Table 2 contained new versions of the new schema.

Information about the old and new debt collection systems was gathered by researching the code base and running the systems with employees of Svea Perintä. In the next section, analysis of both the application for a summons and the debt collection systems at Svea Perintä are discussed.

# 3 Current State Analysis and Schema Changes

Although some time was allocated to examining the old systems in use at Svea Perintä, the current state analysis phase primarily focused on understanding the new collection system being developed. The analysis then proceeded to old schema features and new features added in the new schema. The first subsection describes the implementation of the application for a summons in the old systems; the second subsection briefly introduces the design of the new system being developed; and the finally both the old and the new schemas for an application for a summons are analyzed.

## 3.1 Implementation in the Old Systems

The old systems differed in their approach to applications for a summons. One debt collection system was primarily focused on serving public entity clients. Public entity clients' debts can often be put to distraint without a court order as dictated by the Act on the Enforcement of Taxes and Public Payments, chapter 3. They usually also have their own legal departments, which may sue the debtors after voluntary collection, if a court order is needed. Therefore, the debt collection cases in the system which were brought to court by Svea Perintä had their applications for a summons filed using the Legal Register Centre's web service.

The other old debt collection system had the Santra system integration built into it. The company that developed the system had designed the system in such a manner that most of the system apart from the user interface was implemented as stored procedures in the database management system (DBMS). The DBMS used in that system is Oracle Database, and the Santra integration was implemented by using Oracle Database's XML functions. This implementation of the Santra integration was not a suitable reference for the new implementation, but as the documentation of elements offered by the Legal Register Centre is often quite brief, the applications produced by it were helpful in determining the actual usage of the elements in the applications.

## 3.2 New Debt Collection System

The new debt collection system is being developed in-house at Svea Perintä by consultants and Svea Perintä employees. According to corporate policy, Microsoft’s .NET is used for backend development. The specific platform chosen is .NET Core, and the programming language chosen for .NET development is C#. The programming paradigm used is Object-Oriented Programming. The new system is designed to run as an intranet web application.

The exact design of the system is confidential, but the basic design principles are that different functions of the debt collection process are implemented in such a way that their positions in the process are largely interchangeable, that adding and removing functions from the process is effortless, and that implementing new functions for the process is simple. This allows designing different processes for different use cases, and prepares the system for changes in legislation, which may change the debt collection process.

## 3.3 Old XML Schema for an Application for a Summons

The old XML schema for a Santra application can be divided into a few main parts. An element called *hakemusera* groups several applications into a batch allowing filing of multiple applications in a single file. A batch contains multiple *hakemus* elements representing the individual applications. (Oikeushallinnon tietotekniikkakeskus 2010.)

The contents of a single application represented by a hakemus element can further be divided into three main groups: general information about the application, the parties to the case, and information about receivables. General information contains, for example, the text of the claim in the application, information about the entity responsible for the Santra integration used, the date of the application, and which district court receives the application (Oikeushallinnon tietotekniikkakeskus 2010).

Information about the parties is largely identical for all parties. The root elements for litigant and plaintiff are called *kantaja* and *vastaaja*, respectively, and the child element *asemaKdi* has different values for litigant and plaintiff, but otherwise the elements have the same contents. Both contain elements for personal information of the party, such as the name and the date of birth, the address of the party, and other contact details, such as the phone number and the email address. There can be an arbitrary number of both litigants and plaintiffs. (Oikeushallinnon tietotekniikkakeskus 2010.)

The information about the receivables have their own elements, with every *saatava* element containing a single receivable. The child elements of the saatava element describe the type and amount of the receivable, the due date, and possible interest rate. They also contain an explanation for the receivable and whether the receivable is based on a consumer loan agreement. The number of saatava elements on an application is unlimited. (Oikeushallinnon tietotekniikkakeskus 2010.)

Many of these elements remain the same in the new schema. All the changes concern consumer loan agreements (Virta 2019). These are described below.

## 3.4 Changes in the New XML Schema

According to Virta (2019), the aim in the development of the new schema for the Santra application was that if the application does not concern consumer loan agreements, the contents remains identical to the applications formed using the old schema. A few elements that were used for consumer loan agreements were removed, because new features in the new schema made them redundant (Virta 2019).

If some of the receivables in a Santra application are based on a consumer loan agreement, a new child element called *kuluttajaluottosopimus* is created under the hakemus element. This kuluttajaluottosopimus element has mandatory elements, such as when the consumer loan agreement was made and what is the annual percentage rate. (Virta 2019.)

In addition, kuluttajaluottosopimus has one of these elements depending on the time frame during which the consumer loan agreement was made:

- *tehtyEnnen2010*, if the agreement was made before 1 February 2010
- *tehty2010-2019Valilla*, if the agreement was made between 1 February 2010 and 31 August 2019
- *tehty2019Jalkeen*, if the agreement was made on 1 September 2019 or later.

The chosen time frame affects whether the contract's expiration date is mandatory, and if the time frame is 1 September 2019 or later, the element has additional child elements. (Virta 2019.)

tehty2019Jalkeen has a child element that is called either *liikennevalineTaiAsluotto*, if the agreement concerns financing of either a vehicle or a dwelling, or *eiLiikennevalineTaiAsluotto*, for other agreements. eiLiikennevalineTaiAsluotto has yet more child elements for the amount of the agreement, the end date of the agreement, and the interest rate agreed upon. (Virta 2019.)

If the kuluttajaluottosopimus element is used, the receivables that are based on the consumer loan agreement are listed as child elements of this element. Otherwise, the receivables are direct child elements of the hakemus element, as in the old schema. (Virta 2019.)

This section described the findings of the current state analysis and reviewed the old and the new schema. The next section introduces available knowledge and best practices on technologies and methodologies relevant to the topic.

# 4 Available Knowledge and Best Practices

This section contains a brief introduction to the available knowledge and best practices on *Agile software development*, or *Agile*, object-oriented programming (OOP), .NET, and XML. First, Agile and more specifically the *Scrum* framework are discussed. Then, basic concepts of object-oriented programming are introduced. In the penultimate subsection, .NET is discussed with emphasis on C#, and the section finishes with an overview of XML.

## 4.1 Agile Software Development

Sommerville (2015: 22) observes that for the creation of web-based systems a flexible approach that can handle changing requirements should be considered. In the Agile Manifesto, the word Agile is chosen to represent this flexibility (Agile Alliance n.d.). One of the core values presented in the Agile Manifesto is “Responding to change over following a plan” (Beck et al. 2001). According to Agile Alliance (n.d.), Agile software development differs from other development in that it focuses on the members of the teams and their cooperation. However, what are often overlooked are technical practices that prepare for uncertainty (Agile alliance n.d.).

The principles behind the Agile Manifesto state that development should aim for delivering software swiftly and often, respond to changes in requirements even in late phases, and that development should be conducted in collaboration with people working in the business, not just by developers. They also emphasize the importance of communication, having the right people in the team, and the team adjusting its behavior. (Beck el al. 2001.)

The Agile principles are central to several different frameworks and practices. One of these frameworks is Scrum.

### 4.1.1 Scrum

Scrum is a framework for using different processes for development of complex projects. It is often used in software development, but Scrum is not limited to software development, and can be used for almost any kind of development. The core idea of Scrum is making progress in increments and using the knowledge gained in the process to adjust the direction of the project if needed. Schwaber and Sutherland (2017) state that Scrum has been used in release cycles which have multiple releases per day. (Schwaber and Sutherland 2017: 3–5.)

The people engaged in Scrum form a *Scrum Team*, which has a few different roles. The *Product Owner* is the person who manages all the work and requirements of the product. These form the *Product Backlog*. However, the Product Owner does not dictate who conducts the work. (Schwaber and Sutherland 2017: 6–7, 15.)

The *Scrum Master* is the person responsible for ensuring that everyone involved understands Scrum and helps the people outside the Scrum Team interact with the members of the *Development Team* in a productive manner. The Scrum master also helps the Product Owner in managing the Product Backlog and the Development Team with problems preventing progress. (Schwaber and Sutherland 2017: 7–8.)

The Development Team is a group of developers, which is not divided into further subteams and where members do not have titles. The Development Team is self-organizing, meaning that the Development Team alone decides how the requirements of the Product Backlog are best fulfilled, and who conducts the work. The Development Team takes responsibility for the project together, but members may have areas of expertise. (Schwaber and Sutherland 2017: 7.)

The work is conducted in *Sprints*, which are time periods of one month or less. Items from the Product Backlog are put into *Sprint Backlog*, which is the list of work conducted during the Sprint. If necessary, the items are refined so they can be completed during the Sprint. The items completed during a Sprint form an *Increment*, which must be a useable product; in other words, partial implementations are not allowed. (Schwaber and Sutherland 2017: 9, 16–17.)

During the Sprint, the Development Team gathers daily in *Daily Scrums* to discuss how to progress towards the Sprint goals, and after a Sprint all the work is inspected in *Sprint Retrospective* to help with preparation for the next Sprint, and possible adjustments in the processes are made. A *Sprint Review* is also conducted with the Scrum Team and stakeholders to inspect the work, and possible adjustments to the Product Backlog are conducted. (Schwaber and Sutherland 2017: 12–14.)

## 4.2 Object-oriented Programming

Object-oriented programming is a programming style, or a paradigm, where the design of a program is based on objects. Eng (2017) presents a car as an example of an object. An object may have attributes and behavior represented by methods. For a car the attributes might be the color of the car or how much it weighs, and the representation of its behavior could contain methods for starting the engine, turning, accelerating and so forth. (Eng 2017.)

A *class* is a concept of an object that describes the methods and attributes. A specific object that is represented by this concept is an *instance* of this class. For example, a single car is an instance of the general car class describing the attributes and behavior all cars have in common. (Eng 2017; Tiel 1996.)

A class may duplicate and modify the attributes and behavior of another class through *inheritance* and *polymorphism* (Eng 2017). Tiel (1996) describes inheritance as a a-kind-of relationship. For example, a Siamese is a kind of cat, so it has the attributes and behavior of a cat. Polymorphism is mechanism that allows a class to modify inherited behavior, thus allowing a class that inherits another class to have the same interface but different implementation (Tiel 1996). Eng (2017) states that another kind of relationship between classes is a has-a relationship, where a class contains another class. For example, a car has a motor.

Tiel (1996) explains that the rationale for applying the object-oriented paradigm is that it allows closer modeling of the real world, promotes reusability, and enables decentralization of architecture, thus limiting the effects that changes in one part of a program have on other parts. However, Cardelli (1996) states object-oriented programming results in less efficient programs that those designed using the procedural paradigm, and their compilation time may grow faster than their size. Pobar (2008) suggests that object orientation may encourage developers to rigidly define interfaces and data types in advance, which may not suit large projects with multiple teams or agile methods such as rapid prototyping. Further, in an interview with Peter Seibel, Joe Armstrong expressed that

> I think the lack of reusability comes in object-oriented languages, not in functional languages. Because the problem with object-oriented languages is they've got all this implicit environment that they carry around with them. You wanted a banana but what you got was a gorilla holding the banana and the entire jungle. (Seibel 2009.)

In contrast, Bernie Cosell recommended studying object-oriented programming in its abstract form (Seibel 2009).

The object-oriented paradigm is supported by numerous programming languages. One of these is C#, which is one of the languages supported by .NET.

### 4.3 .NET and C#

According to Microsoft (2020) .NET is a developer platform supporting multiple languages that can be used for development of web, mobile, IoT and desktop applications on multiple platforms.

.NET, originally known as Next Generation Windows Services, was announced by Microsoft in 2000. Initially, it was described as a layer running on servers and clients, offering access to networked services running either on the web or on company servers. It was envisioned to have “a sparse browser-like user interface without any menu bars”. (Deckmyn 2000.)

Despite these early visions describing even the user interface of the services, .NET extended to become a more generalized programming framework called .NET Framework, a simpler and more flexible alternative for the Component Object Model (COM), which was frequently used for the creation of Windows applications. Like older software based on COM, software developed using .NET could interoperate with existing software and support multiple programming languages, but .NET had a simpler deployment model, as .NET libraries did not require registering to the system registry. (Troelsen and Japikse 2017: 3–4.)

.NET Framework is a Windows implementation of the ECMA Common Language Infrastructure (CLI), which was implemented on other platforms in projects such as Mono. However, in 2016 Microsoft released .NET Core for Windows, macOS and Linux. This allows both development and execution of .NET Core software on all these platforms. (Microsoft 2020; ECMA 2012; Mono Project 2020; Troelsen and Japikse 2017: 30–32, 1245–1246.)

While .NET Core initially coexisted with .NET Framework, in 2019 Microsoft announced that the path .NET would take in the future would be .NET Core. Version 4.8 would be the last version of .NET Framework, and the next version of .NET Core, planned to be released in 2020, would be called .NET 5. (Lander 2019; Hunter 2019.)

The languages supported by .NET Core are C#, F#, and Visual Basic. For object-oriented programming the main language is C#, a type-safe object-oriented programming language in the C family of programming languages. Its syntax resembles closely Java, another member of the C family, but also includes aspects for non-C family languages such as Visual Basic, LISP, and Haskell. For example, C# supports the use of class properties instead of getters and setters that are more commonly used in the C family languages, and functional programming features such as lambda expressions. (Microsoft 2020; Microsoft 2015; Troelsen and Japikse 2017: 5.)

C# supports encapsulation, inheritance, and polymorphism, as needed in object-oriented programming. All variables and methods are encapsulated in classes, which may inherit a single parent and implement additional interfaces. C# also supports garbage collection and exception handling.  (Microsoft 2015; Microsoft 2017.)

Regardless of programming language, code targeting .NET runs on Common Language Runtime (CLR) or on CoreCLR, .NET Core’s version of CLR. CLR is .NET’s implementation of Virtual Execution System (VES) specified in CLI. Microsoft calls code targeting

.NET *managed code.* Managed code compiles to Intermediate Language (IL) instead of machine code. IL code is packaged in assemblies, which the CLR then compiles into machine code just before execution. The runtime manages tasks such as object referencing and dereferencing, threading, and integration and exception handling across languages. (Microsoft 2019; ECMA 2012; Gregory 2003; Troelsen and Japikse 2017:4, 8.)

## 4.4 Extensible Markup Language

Encyclopaedia Britannica (2011) defines a markup language as a system for describing a document's structure and layout within the document with text symbols that can be distinguished from the content. Extensible Markup Language, or XML, is a markup language derived from Standard Generalized Markup Language (SGML) that was designed for electronic publishing but has an important role as a data exchange format (Hemmendinger 2020; Quin 2016).

Unlike Hypertext Markup Language (HTML) used for the creation of basic web pages, XML allows the definition of new symbols, called elements, allowing document structures to be tailored for specific use cases (Hemmendinger 2020). A trivial example of XML and custom elements is presented in Listing 1 below.

```
<exampleContainer>
     <example>This is an example of XML elements</example>
</exampleContainer>
```

Listing 1. A trivial example of XML.

The example begins with an opening XML tag that marks the beginning of the element exampleContainer. Tags are marked by enclosing the text between less than and greater than signs. Everything from an element's opening tag until the element's closing tag, which is similar to the opening tag but prefixed with a slash (/), is contained within that element. In the example listing, this means that the element called example is contained within the exampleHolder element. The example element is called a subelement of the exampleContainer element. The phrase contained between the example element's opening and closing tags is the content of that element. (W3C 2004.)

In addition to content, XML elements may have attributes, as shown in Listing 2 below.

```
<toybox>
     <toy color=”red”>ball</toy>
     <toy color=”yellow”>car</toy>
</toybox>
```

Listing 2. An example of XML attributes.

Attributes define the properties, in the given example the color, of a specific element. The element toybox contains two toy elements, which both have color defined. The first element represents a ball, which is defined to be red, and the second element a car, which is yellow. (W3C 2004.)

There are several ways to define these elements, one of them being XML Schema language (W3C 1999).

### 4.4.1 XML Schema

As XML is derived from SGML, it supports the definition of elements and document structure constraints in Document Type Definition (DTD) files. However, to extend functionality and to allow constraints that differ from those possible in DTD, the XML Schema language was created. (Hemmendinger 2020; W3C 1999.)

The XML Schema language allows the definition of schemas in XML; in other words, it is a schema for writing schemas. It defines mechanisms that allow schema writers to define datatypes, elements, namespaces, constraints and so forth. Schemas written using the XML Schema language are contained within an XML Schema Definition (XSD) file. (W3C 1999; W3C 2004.) An XSD file containing the schema for the XML document in Listing 2 can be seen in Listing 3.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="toy">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="color" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="toybox">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="toy" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 3. An XSD file defining a simple XML document

The schema starts with a schema tag which opens the schema element. Inside the schema tag a namespace prefix “xs” is defined. This means that all elements the names of which start with “xs:” are part of the namespace “http://www.w3.org/2001/XMLSchema”. This allows multiple authors to create elements with similar names, as they can be recognized from each other by their namespaces. Although the name of the namespace looks like a web address, it is used just for naming. (W3C 2004; W3C 2012.)

Then an element named toy is defined. The toy element is defined to be of a complex type, meaning it can contain more than text. This is needed because the toy element has an attribute. Apart from the attribute, the element is defined to contain simple content, meaning that the element does not have subelements. The element is defined to be an extension of the string type and may contain an optional attribute color. (W3C 2004; W3C 2012.)

After the definition of the toy element, an element named toybox is defined. It is likewise a complex type. The toybox element is defined to contain a sequence of elements. The elements in the sequence must be of type toy and there may be an arbitrary number of them. The order of the elements inside the toybox must match the order defined in the sequence. In the example, the toybox element is defined to contain only toy elements, so the sequence could be replaced in the schema by an any element, which defines which elements it may contain, but they may be in an arbitrary order. (W3C 2004; W3C 2012.)

This section has described various methodologies and technologies, including Agile software development, .NET and C#, and XML. The next section demonstrates how these were applied in a real-life project.

# 5 Electronic Application for a Summons

This section introduces how the knowledge and practices presented in the previous section were applied to a project, the aim of which was to design a process and create a library for the creation of electronic applications for a summons. The project was a part of a larger collection system project at Svea Perintä. This section presents the steps taken in the project in chronological order. First, preceding events are outlined. This is followed by defining the goals of the project, and analysis and research. Then the design and implementation are presented, and finally validation and corrections are discussed.

## 5.1 Preliminary Phase

During the spring of 2019 the collection system project at Svea Perintä was in such a phase that it was deemed necessary to implement an electronic application for a summons. The clients that were handled in the new system did not need this functionality at the time, so adhering to the Agile principles, the functionality had not been implemented yet. However, the project neared a phase where the migration of the clients from the old system would begin. It was already known that legislation concerning consumer loan agreements would be changing in September 2019, and that the Legal Register Centre was designing a new schema for the electronic application for a summons that would take these changes into account. The Legal Register Centre was planning to deploy the new schema in early 2020.

At the time, the author needed a subject for his thesis project, so Development Manager Susanna Havanka suggested that the electronic application for a summons could be a candidate. This subject was accepted by Janne Salonen, head of ICT at Metropolia University of Applied Sciences. The upcoming summer vacation season and the author’s other duties at Svea Perintä’s System Development department postponed the start of the thesis project until the autumn of 2019, but preliminary research into the Legal Register Centre’s existing documentation was conducted during the summer.

## 5.2 Definition of the Objective and the Outcome

As the changes in the schema were imminent, it was deemed impractical to implement the old schema for a few months. Thus, the objective of the project was to design the process adhering to the new schema, and initially the outcome was planned to be a module for the automatic creation of electronic applications for a summons. Even though the collection system project is using Agile methodologies, and the Scrum principles state that all the work conducted during a sprint should form an increment, all the user stories concerning the electronic application for a summons were initially assigned to the author. While the objective clearly could not be completed during one sprint, dividing all the work for the electronic application for a summons to multiple developers would prevent the author from writing his thesis.

## 5.3 Analysis and Research

After the objective and the outcome of the project were defined, current state analysis was conducted and features of the old and the new schema were researched. Research suggested that although the new version of the schema was not finalized, the possible changes to the schema would all concern consumer loan agreements. Thus, large parts of the process design could be based on the old schema and the draft of the new schema. Findings from current state analysis and research can be found in section 3.

During the scheduling of the work needed for the electronic application for a summons module, it was determined that the author’s other duties at the company and the schedule of the project necessitated assigning some of the work to other developers. To keep the subject of the thesis well defined, it was decided that the author would design the process for the creation of the applications and implement a library for their creation, but the implementation of the logic in the collection system that triggers the creation and collects the data necessary was assigned to other developers.

## 5.4 Design and Implementation

The collection system project adheres to the object-oriented paradigm and uses .NET Core for the platform. As the outcome of the thesis project is not an independent program but a library used in the development of collection system, the library must also adhere to the object-oriented paradigm and use .NET Core. While in principle F# or Visual Basic could have been used to implement the library, C# is the .NET language designed for object-oriented programming (Microsoft 2020). As the rest of the collection system project is implemented in C#, it was also the most reasonable choice in terms of maintainability.

With the paradigm, the platform and the programming language defined, design and implementation proceeded. As the structure of an application is predefined, the author decided that it would be reasonable to begin with the design of the classes needed for representation of an application and then proceed to design of the process for creation of the application. It was requested that the application for a summons and writing of the XML file were separated from each other, so classes representing applications and a separate writer class for these were designed. The writer class was developed concurrently with the process design.

### 5.4.1 Classes for Applications and Batches

As the application for a summons has all its element named in Finnish, it was decided that the classes would be named similarly to make the association between the elements and the classes apparent. Thus, the class representing an electronic application for a summons is named *SantraHakemus*. A simple container class representing a batch of applications named *SantraHakemusera* was also defined. While SantraHakemusera has only one property, a list of SantraHakemus objects, which could be handled in the program using the library, conceptually this is an important class, as the XML file delivered to Santra systems contains a batch of multiple applications.

The XML application contains multiple complex types representing concepts such as parties to a case or receivables. Some of the complex types even contained other complex types. It is evident that there is a has-a relationship between, for example, an application and its parties, and the division of the elements in the XML application is logical, so the XML schema was used as a basis for further class design. In the final class design, many of the complex types are represented by their own classes. The classes for representing the application batch and the applications are:

- SantraHakemusera, to represent a batch of applications,
- SantraHakemus, to represent an application,
- *SahkoinenHakija* for the entity filing the application,
- *Asianosainen*, *to* represent both the plaintiffs and the litigants of the case,
- *Hennimi* for the names of the parties,
- *Osoite* for the addresses of the parties,
- *Kuluttajaluottosopimus*, to represent consumer loan agreements,
- *Saatava* for receivables, and
- *Viivastyskorko*, to represent late payment interest

SahkoinenHakija represents the entity filing the application, whether the original debtee or a professional collection agency. In the collection system for which this library is designed for the entity is presumably always Svea Perintä, but the class allows information of the entity on the application to be freely defined.

Asianosainen represents both the plaintiffs and the litigants of a case, whether a natural or a legal person. The class contains properties for personal and contact information of the parties and for information on a possible representative to the party. The name of the party is contained in Hennimi. The XML application for a summons has an element which represents the full name of a legal person, but in case of a natural person contains just the last name, while the first name is in an additional element. Both cases are represented by this class. Osoite contains all the address information of a party to the case or a representative to the party.

Kuluttajaluottosopimus represents consumer loan agreements. Applications concerning consumer loan agreements contain a considerable amount of information that is otherwise not present, such as the expiration date of the agreement and the time frame during which the agreement was made, as this affects the legislation used to decide the case. This information is contained in this class.

Saatava represents the receivables in an application. It contains the type, the amount, and an explanation of the receivable, as well as information on the late payment interest of the receivable in a class named Viivastyskorko.

The source code of all the classes is documented with XML documentation features of C#. The documentation contains brief description of the classes and their members, and some additional information about the properties containing data for some of the less obvious features of the application for a summons. All classes also have at least empty constructors explicitly defined. While a class without a constructor would be provided with an empty constructor automatically, possible addition of a non-empty constructor in the future would remove this and break all code using the implicit empty constructor. The values of properties are checked against the constraints set by the schema and if values are outside the constraints an exception is thrown. Even in cases where the class could modify the value to fit in the property, for example by truncating the names to lengths allowed by the schema, throwing an exception is preferable as this forces the user of the library to decide how the situation is handled and minimizes the possibility of unintended consequences.

### 5.4.2 Writer Class

As requested, the writing of XML data was separated into a writer class *SantraWriter*. SantraWriter has methods for writing either a single application or a batch of applications. The methods write the XML data into a file, as this was the only output requested. This was deemed sufficient for the initial version and could be developed further in the future.

An application for a summons has many complex dependencies, where the existence of an element may require the existence of some other elements and prohibit others. Some of these are not even enforced by the schema, so it is possible to create an application which would validate against the schema but which would be dismissed because of the contents. To help the developers of software creating applications for a summons, SantraWriter tries to detect these logical errors and throw exceptions.

There are many optional elements, which must be checked during writing, and some elements which are optional in principle but mandatory in practice. These, combined with the checks for logical errors, result in a considerable amount of conditional logic. To keep the source code more readable, guard clauses are used to reduce nesting in conditional logic in the private methods. Guard clauses check conditions essential to the execution before the execution proceeds, so these conditions may be ignored later in the method. For example, there are several ways to define late payment interest, but only one of these may be applied at a time. A guard clause in the method responsible for the creation of the viivastyskorko element checks that only one of these is defined before continuing the execution. If a definition of a late payment interest is detected later in the execution, it can be used without checks, as it is the only one.

### 5.4.3 Process Design

The process of the creation of applications for a summons was designed concurrently with the SantraWriter class, as understanding of the process was essential for the logical integrity checks mentioned in 5.4.2. The guiding principle in the process design was that the user of the library would not need to be familiar with the contents of an electronic application for a summons, but to only be able to provide information defined by the process description.

To achieve this, the process description has step by step instructions for the user. The process was divided into multiple subprocesses to make the process chart easier to perceive. Part of the process chart can be seen in Figure 2 below.

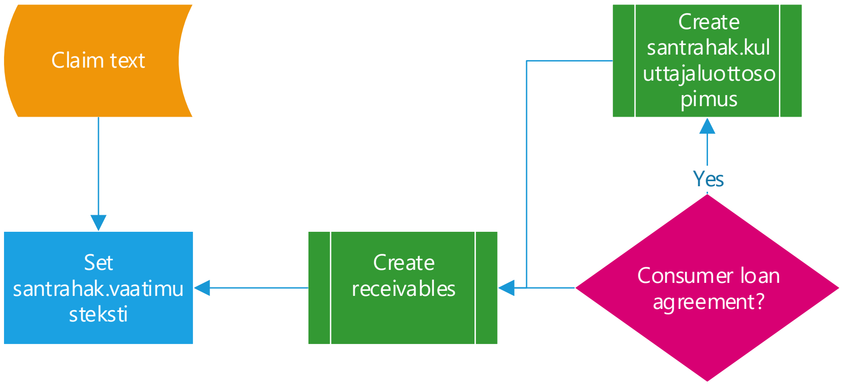

Claim text
Create santrahak.kuluttajaluottosopimus
Yes
Set santrahak.vaatimusteksti
Create receivables
Consumer loan agreement?

Figure 2. Extract from the Process Chart

Division to subprocesses generally mirrors the class division of the application for a summons and some subprocesses have further subprocesses. While the process charts describe input data on a general level, such as in Figure 2 above, Svea Perintä received written instructions containing more detailed information for their specific use cases. In addition to the main process, the final process description has subprocesses for:

- Setting the electronic applicant for the application for a summons,
- Setting basic application information,
- Creating parties to the case,
- Creating consumer loan agreements,
- Creating receivables,
- Setting the names of the parties,
- Setting the addresses of the parties, and
- Setting the late payment interest of a receivable

All the process charts can be found in Appendices 1–9.

## 5.5 Validation and Corrections

The initial design of the process and implementation of the classes were finished in November 2019. During validation in December 2019, a single fault was discovered. SantraWriter did not recognize that a single application for a summons could have some receivables that are based on a consumer loan agreement and some that are not, resulting in some of the receivables not being written. This was corrected. Some additional XML documentation for the source code was also added during December 2019 and January 2020.

The library produced in the thesis project is already in use at Svea Perintä.

# 6 Summary and Conclusions

This section contains a summary of the thesis, and conclusions of the thesis project. The first subsection summarizes the thesis; the second part discusses how the library could be further developed. Then the thesis is evaluated, and the section finishes with final words.

## 6.1 Executive Summary

The thesis project was conducted in the System Development department of Svea Perintä. At the time, Svea Perintä was developing a new debt collection system. Concurrently, the Legal Register Centre was developing a new version of the schema for the application for a summons. The task given was to design a process for automatic creation of electronic applications for a summons in XML, incorporating the new features of the new schema.

The project was conducted by inspecting the implementations for applications for a summons in the existing systems at Svea Perintä, by studying documentation offered by the Legal Register Centre, and by reviewing literature.

During the current state analysis, it was found out that the technologies used in the existing system would not be suitable for implementation in the new system, but the material they produced could be used as a reference. The review of the old schema and the draft of the new schema revealed that the changes in the schema mostly concerned consumer loan agreements.

The thesis project was conducted with the C# programming language on the .NET Core platform adhering to the object-oriented programming paradigm. Classes for representing an application for a summons and a writer class for writing the applications in XML form were developed, along with a process for creating these applications.

The outcome of the project was a process for creating an electronic application for a summons, and a supporting C# library. The library is currently in production use at Svea Perintä.

## 6.2 Next Steps

While the library has been adopted to production use, there are some aspects that could be refined. First, some of the class properties could be derived from other properties, leaving the user of the library fewer details to handle. For example, currently creating an instance of Kuluttajaluottosopimus requires providing both the date and the time frame when the consumer loan agreement was made, even though the time frame could be derived from the date.

Second, constructors designed for the most common use cases of the classes could be defined. Currently properties are mostly left to be manually set, as at the time of development it was not clear which properties would be most used. Simplified process charts for these use cases could also be designed.

Third, SantraWriter could be further developed to support streams for output. Initially only support for writing files was requested, but even then this was described as a good basis for further development. Using streams would enable numerous different output targets.

## 6.3 Thesis Evaluation

Initially the aim of the thesis project was to create a complete solution for creating applications for a summons. Unfortunately, it soon became clear that the project plan had to change in order to meet deadlines. However, a coherent subject for a thesis could be defined. The redefined aim of the project was to design the process for the creation of applications for a summons and a supporting C# library. This was achieved.

Even the revised schedule was somewhat tight, so the functionality of the library could be refined. Nevertheless, the library implements all the necessary features and the process description is detailed enough for a person with no prior knowledge of the topic to create software that creates applications for a summons. This was demonstrated at Svea Perintä, which uses the library and the process in a production system.

## 6.4 Final Words

This thesis exists because of support from Svea Perintä and its System Development department. Especially Development Manager Susanna Havanka and Software Architect Janne Loikkanen were crucial for the project. When the schedule of the collection system project was at odds with the thesis project, they found ways to reschedule so the thesis could be finished. For those in the same situation in the future, do not despair, because, to paraphrase Helmuth von Moltke the Elder,

> “No battle plan survives first contact with the enemy.”

## References

Agile Alliance. (n.d.). Agile 101. [online] Available from: https://www.agilealliance.org/agile101/ [Accessed 22 February 2020].

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. (2001). *Manifesto for Agile Software Development*. [online] Agilemanifesto.org. Available at: http://www.agilemanifesto.org/ [Accessed 22 February 2020].

Cardelli, L. (1996). Bad Engineering Properties of Object-Oriented Languages. ACM Computing Surveys, 28(4es), Article 150.

Dean, J., Grove, D. and Chambers, C. (1995). *Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis.* [online] Seattle: University of Washington. Available from: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.2420 [Accessed 22 February 2020].

Deckmyn, D. (2000). Update: Microsoft stakes future on .Net strategy. [online] Available from: https://www.computerworld.com/article/2596383/update--microsoft-stakes-future-on--net-strategy.html [Accessed 22 March 2020].

ECMA. (2012). *ECMA-335 Common Language Infrastructure (CLI)*. [online] ECMA International. Available from: https://www.ecma-international.org/publications/standards/Ecma-335.htm [Accessed 22 March 2020].

Eng, R. K. (2017). Chapter 3: What is Object-Oriented Programming? [online] Available from: https://medium.com/learn-how-to-program/chapter-3-what-is-object-oriented-programming-d0a6ec0a7615 [Accessed 22 February 2020].

Gregory, K. (2003). Managed, Unmanaged, Native: What Kind of Code Is This? [online] Available from: https://www.developer.com/net/cplus/article.php/2197621 [Accessed 22 March 2020].

Hemmendinger, D. (2020). ‘XML’, *Encyclopædia Britannica.* Encyclopædia Britannica, inc..

Hunter, S. (2019). .NET Core is the Future of .NET. Available from: https://devblogs.microsoft.com/dotnet/net-core-is-the-future-of-net/ [Accessed 22 March 2020].

Lander, R. (2019). Introducing .NET 5. [online] Available from: https://devblogs.microsoft.com/dotnet/introducing-net-5/ [Accessed 22 March 2020].

Legal Register Centre (2019). Sähköinen haastehakemus. [online] Available from: https://www.oikeusrekisterikeskus.fi/fi/index/loader.html.stx?path=/channels/public/www/ork/fi/structured_nav/asiakaspalvelu/sahkoinenhaastehakemus [Accessed 7 April 2020].

Microsoft. (2015). Introduction to the C# language and the .NET Framework. Available from: https://docs.microsoft.com/fi-fi/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [Accessed 22 March 2020].

Microsoft. (2017). Introduction. Available from: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/introduction [Accessed 22 March 2020].

Microsoft. (2019). Common Language Runtime (CLR) Overview. Available from: https://docs.microsoft.com/en-us/dotnet/standard/clr [Accessed 22 March 2020].

Microsoft. (2020). What is .NET? An open-source platform. [online] Available from: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet [Accessed 22 March 2020].

Ministry of Justice (2019). Summary civil cases. [online] Available from: https://oikeus.fi/tuomioistuimet/karajaoikeudet/en/index/riita-asiat_1/sahkoinen-haastehakemus.html [Accessed 7 April 2020].

Mono Project. (2020). The Mono Runtime. [online] Available from: https://www.mono-project.com/docs/advanced/runtime/ [Accessed 22 March 2020].

Oikeushallinnon tietotekniikkakeskus. (2010). *XML-muotoisen haastehakemuksen formaatti.* [online] Hämeenlinna: Oikeushallinnon tietotekniikkakeskus. Available from: https://www.oikeusrekisterikeskus.fi/material/attachments/ork/oikeusrekisterikeskus/santra/vKkwwN4rh/XML-muotoisen_haastehakemuksen_formaatti_2010.pdf [Accessed 30 August 2019].

Pobar, J. (2008). Alphabet Soup: A Survey of .NET Languages and Paradigms. *MSDN Magazine*, May 2008 [online]. Available from: https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/may/alphabet-soup-a-survey-of-net-languages-and-paradigms [Accessed 22 February 2020].

Quin, L. (2016) Extensible Markup Language (XML). [online] Available from: https://www.w3.org/XML/ [Accessed 29 March 2020].

Riel, A. J. (1996). *Object-Oriented Design Heuristics.* [online] Addison-Wesley, 2001. Available from: https://learning.oreilly.com/library/view/object-oriented-design-heuristics/020163385X [Accessed 22 February 2020].

Schwaber, K. and Sutherland, J. (2017). *The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game.* [online] ScrumGuides.org. Available from: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf [Accessed 22 February 2020].

Seibel, P. (2009). *Coders at Work. Reflections on the Craft of Programming.* [online] Apress. Available from: https://learning.oreilly.com/library/view/coders-at-work/9781430219484/ [Accessed 22 February 2020].

Sommerville, I. (2016). *Software Engineering.* 10th ed. Harlow: Pearson Education Limited.

Svea Group. (2020a). About us. [online] Available from: https://www.svea.com/gb/en/about/about-us/ [Accessed 7 April 2020].

Svea Group. (2020b). Perimme saatavasi hyvässä hengessä. [online] Available from: https://www.svea.com/fi/fi/yritykset/perinta/ [Accessed 7 April 2020].

The Editors of Encyclopaedia Britannica. (2011). ‘Markup language’, *Encyclopædia Britannica.* Encyclopædia Britannica, inc.

Troelsen, A. and Japikse, P. (2017). *Pro C# 7: With .NET and .NET Core.* 7th ed. Apress.

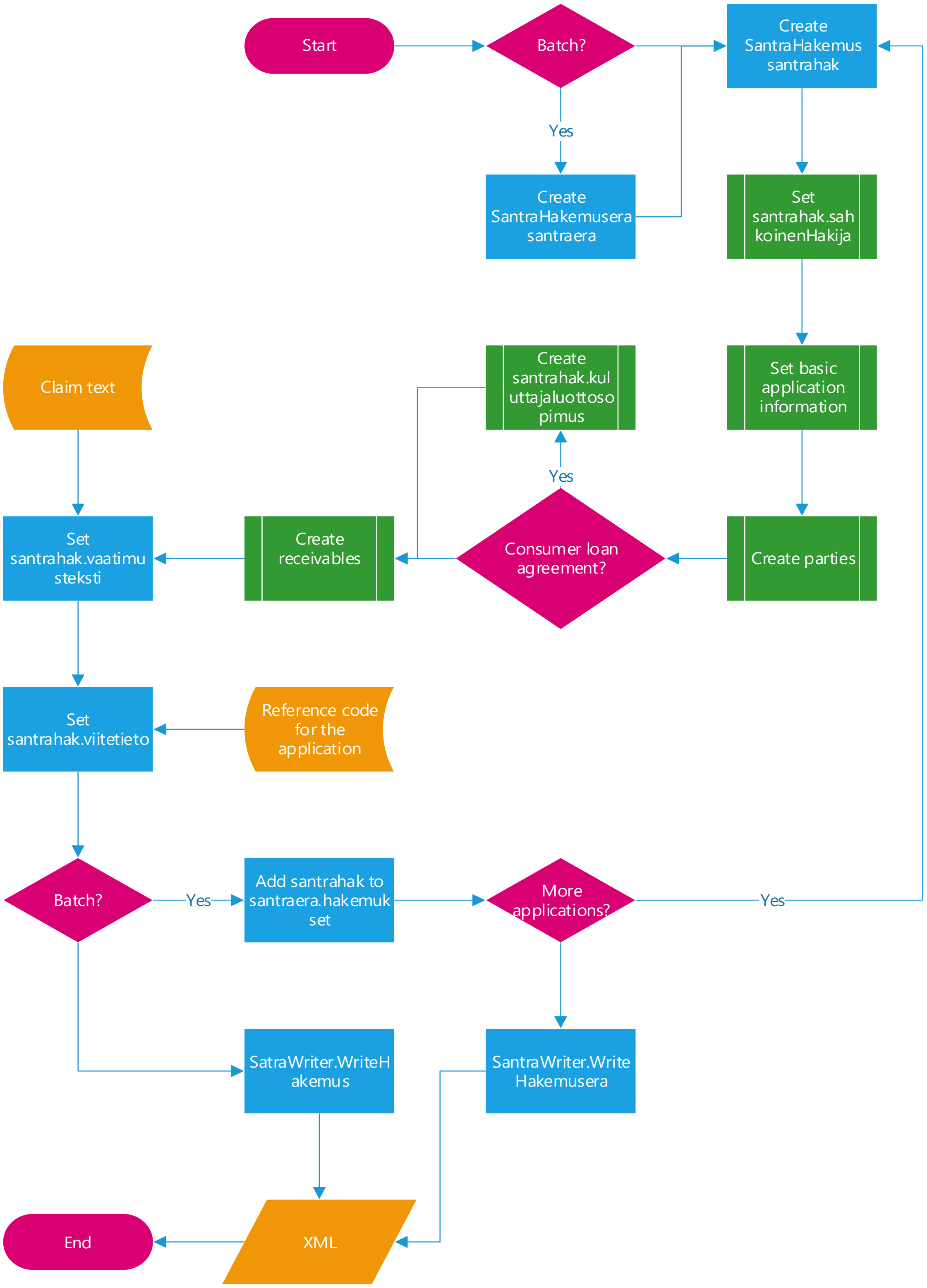
Virta J. (2019). *Kuluttajaluottouudistus 2019 Info-tilaisuus Santra-hakijoille*. Helsinki: Oikeat Oliot.

W3C. (1999). XML Schema Requirements. [online] Available from: https://www.w3.org/TR/NOTE-xml-schema-req [Accessed 29 March 2020].

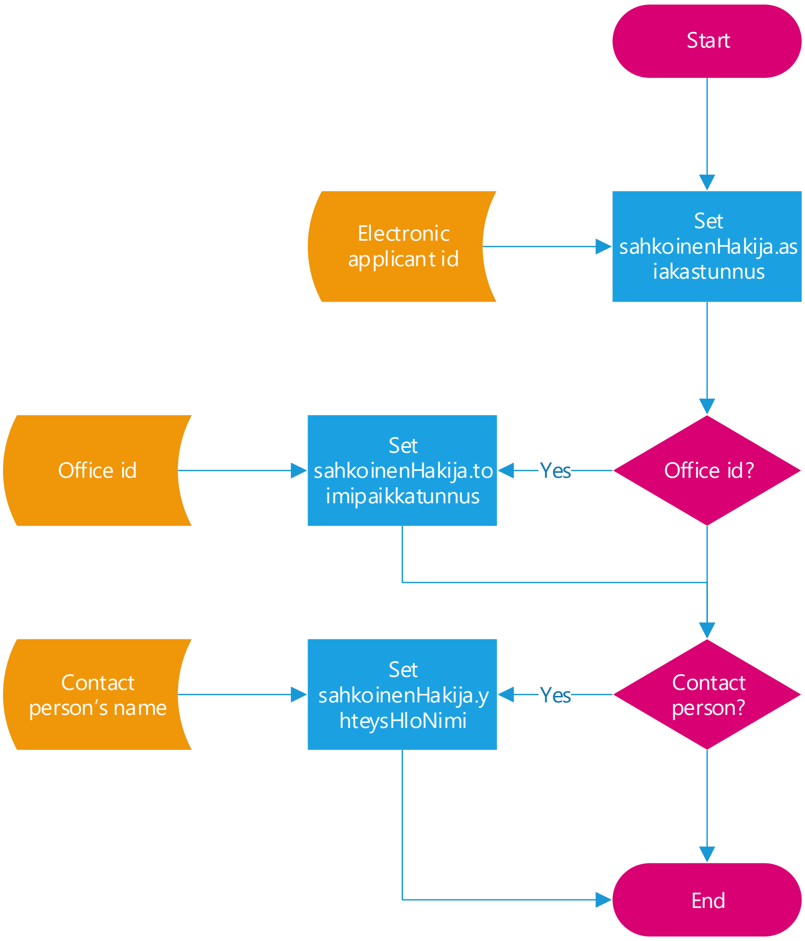
W3C. (2004). XML Schema Part 0: Primer Second Edition. [online] Available from: https://www.w3.org/TR/xmlschema-0/ [Accessed 29 March 2020].

W3C. (2012). W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. [online] Available from: https://www.w3.org/TR/xmlschema11-1/ [Accessed 29 March 2020].

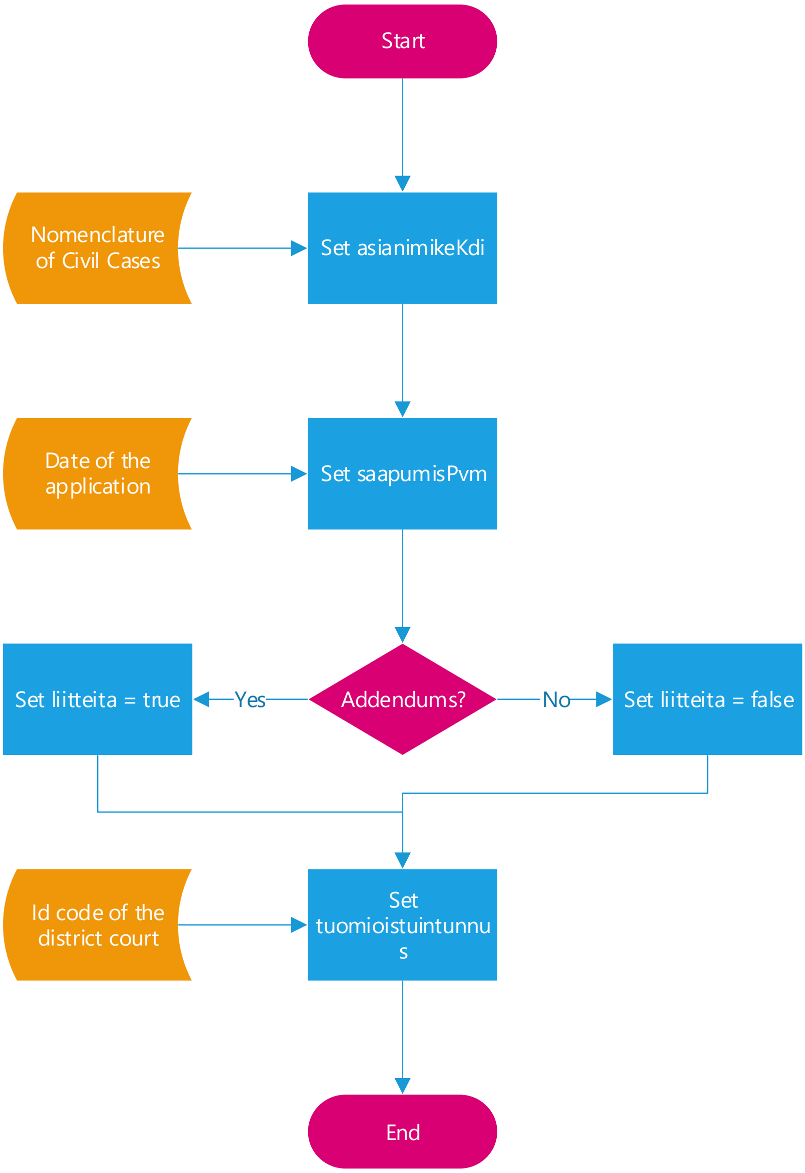
## Application Creation Process


Start
Batch?
Yes
Create SantraHakemus santrahak
Create SantraHakemusera santraera
Set santrahak.sahkoinenHakija
Set basic application information
Create santrahak.kuluttajaluottosopimus
Yes
Claim text
Set santrahak.vaatimusteksti
Create receivables
Consumer loan agreement?
Create parties
Set santrahak.viitetieto
Reference code for the application
Batch?
Yes
Add santrahak to santraera.hakemukset
More applications?
Yes
SatraWriter.WriteHakemus
SantraWriter.WriteHakemusera
End
XML

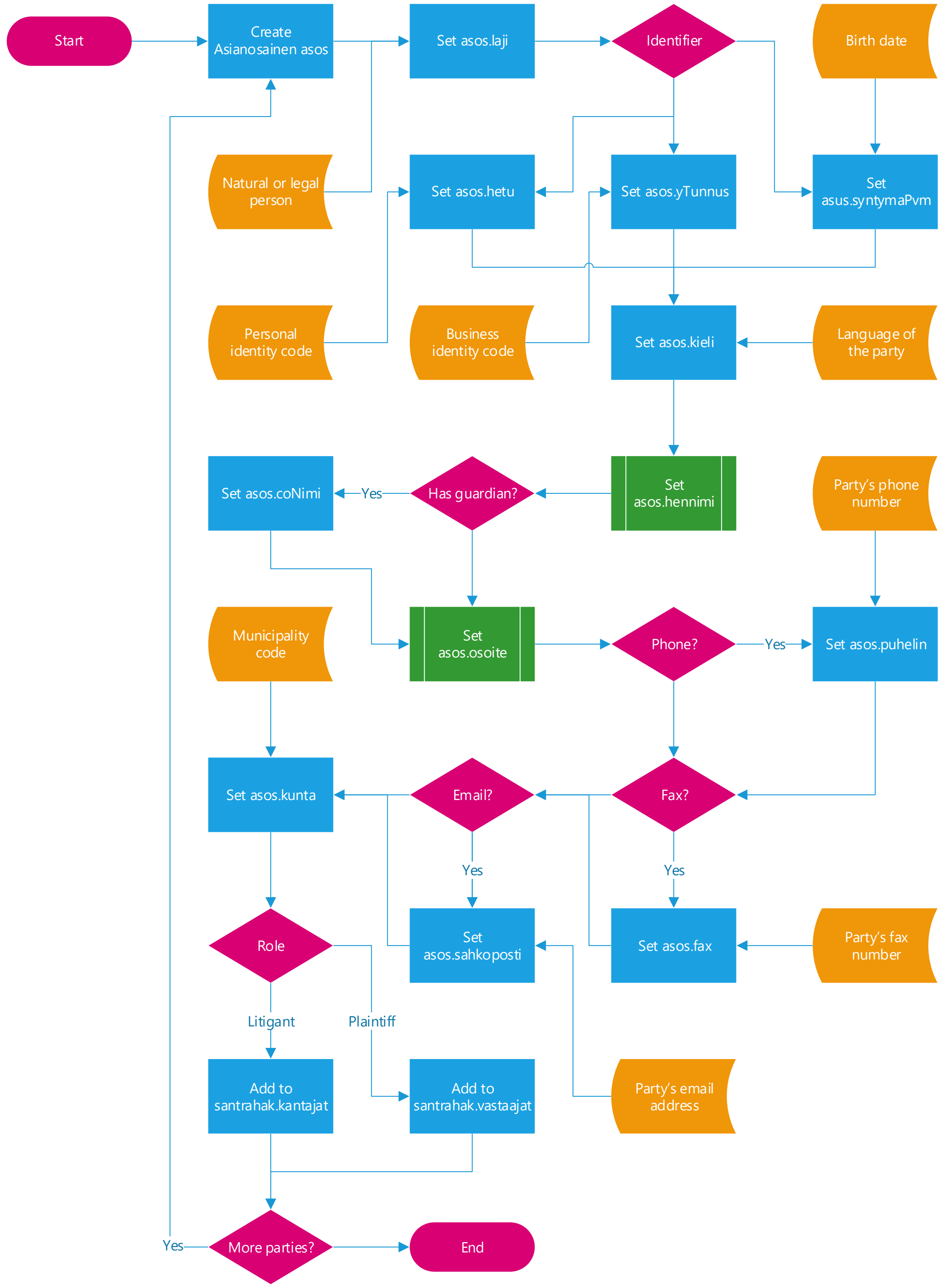
## Set santrahak.sahkoinenHakija


Start
Electronic applicant id
Set sahkoinenHakija.asiakastunnus
Office id
Set sahkoinenHakija.toimipaikkatunnus
Yes
Office id?
Contact person's name
Set sahkoinenHakija.yhteysHloNimi
Yes
Contact person?
End

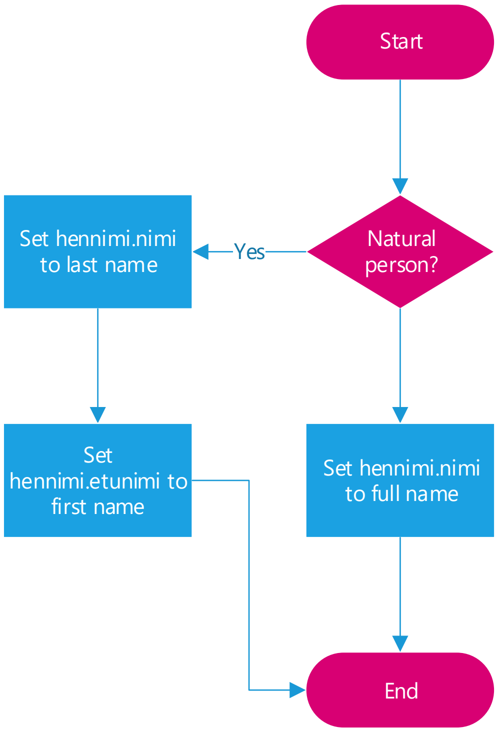
## Set Basic Application Information



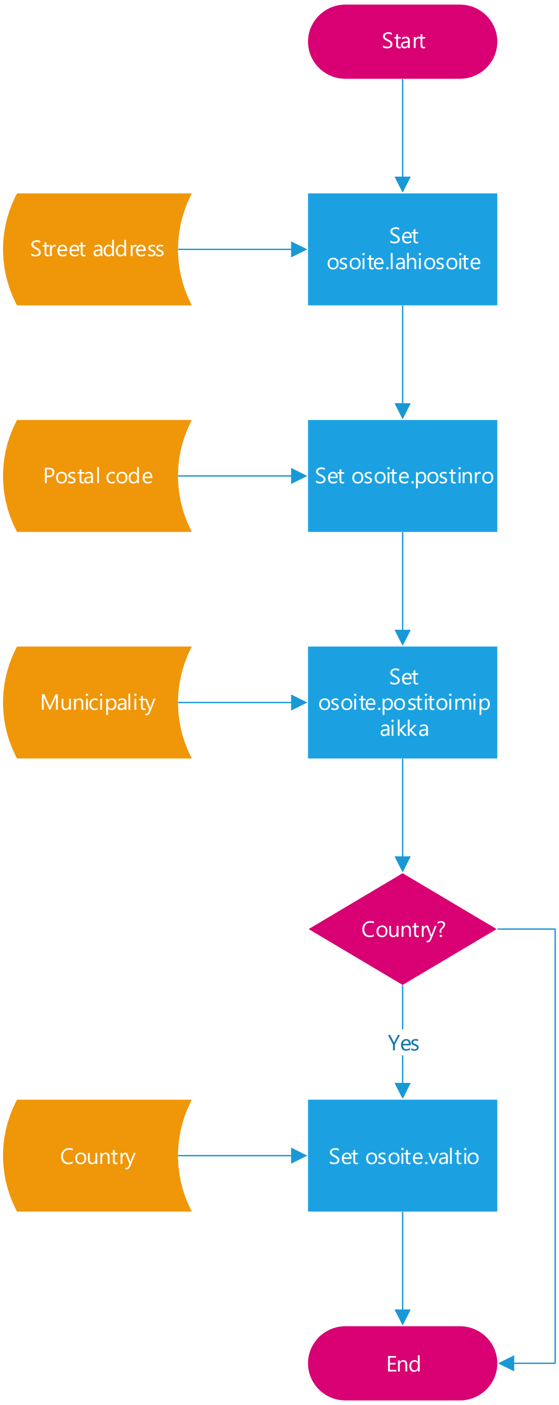
## Create Parties



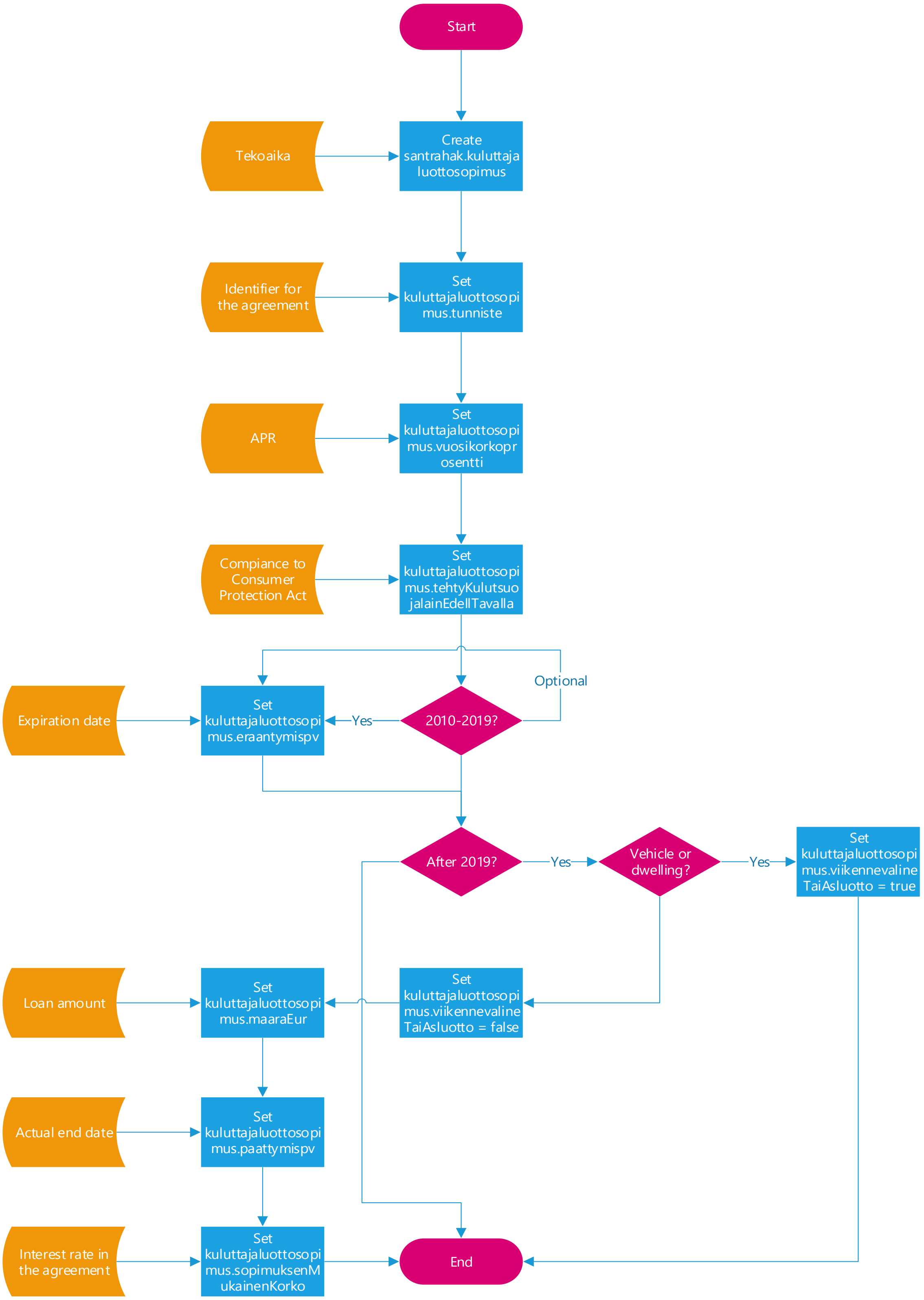
### Set asos.hennimi


Start
Natural person?
Yes
Set hennimi.nimi to last name
Set hennimi.etunimi to first name
Set hennimi.nimi to full name
End

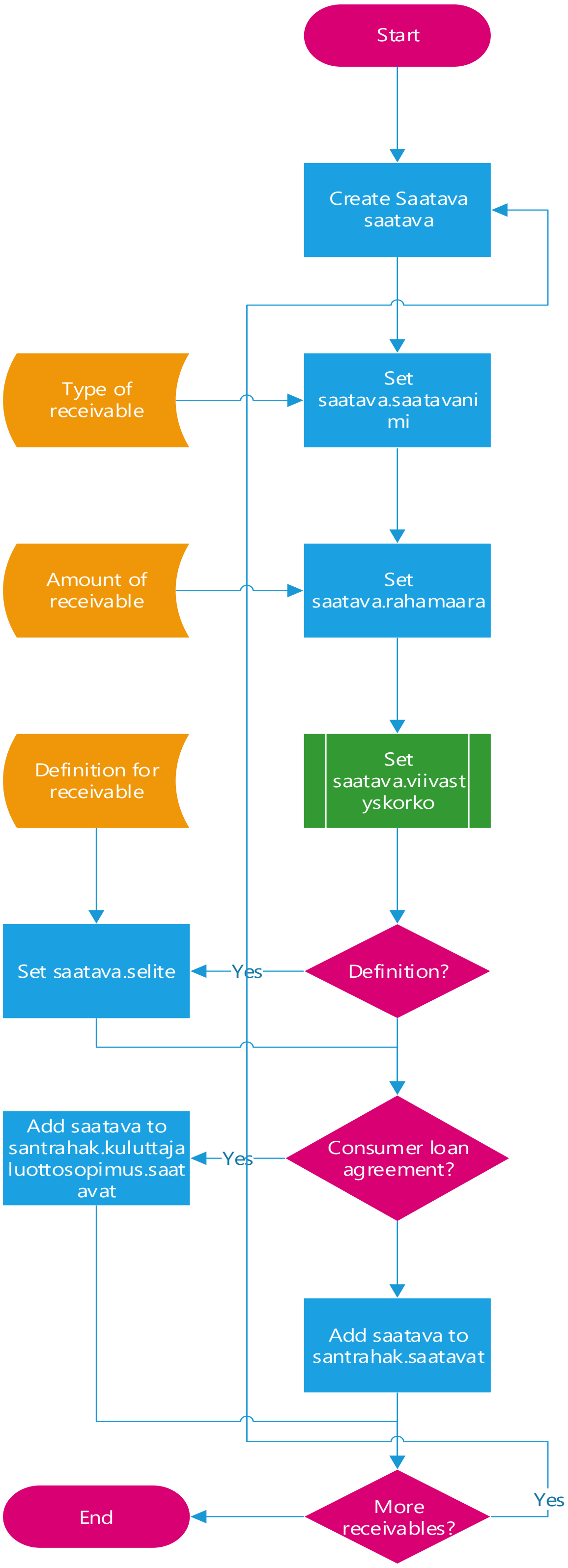
## Set asos.osoite



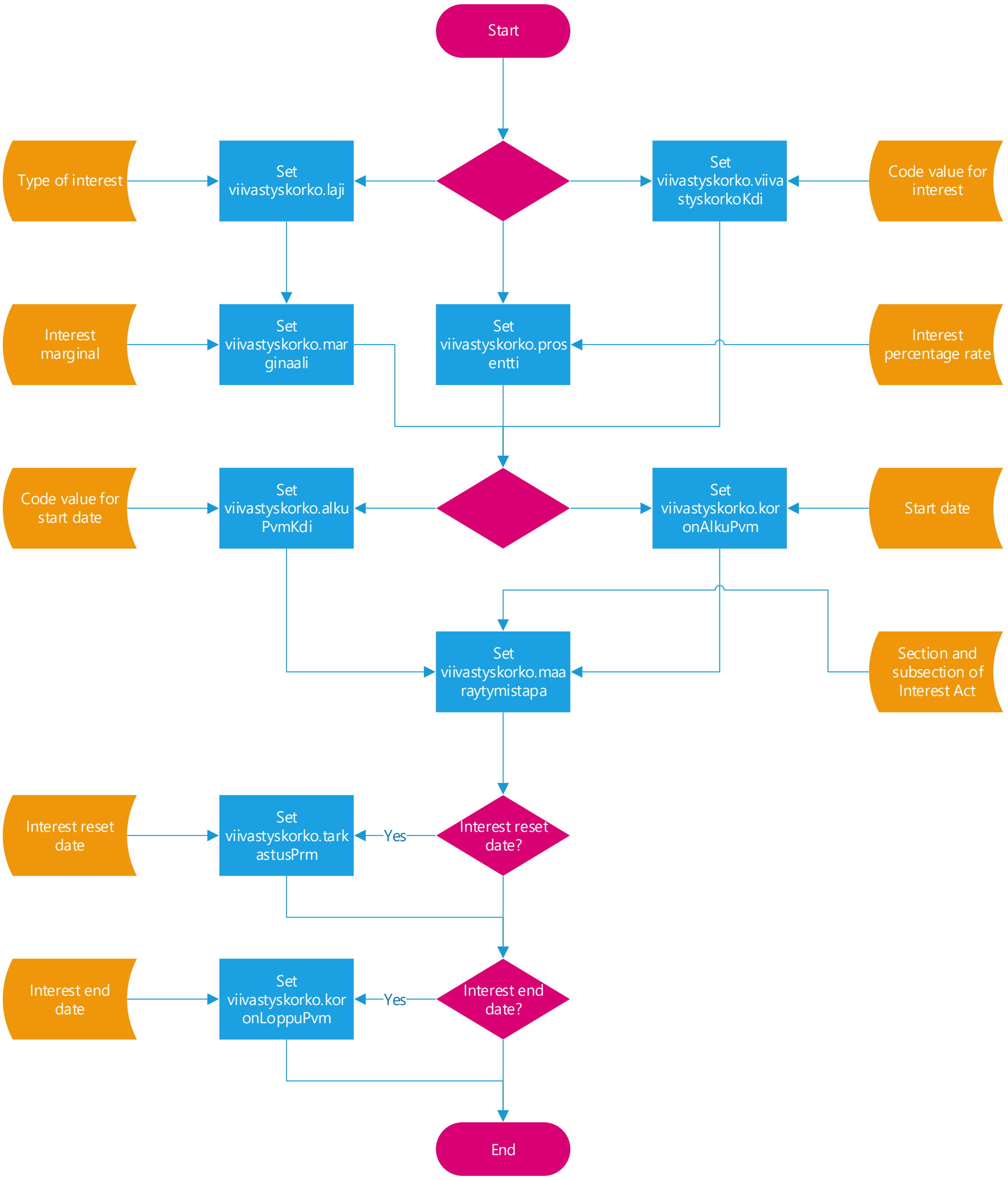
## Create santrahak.kuluttajaluottosopimus


Start
Tekoaika
Create santrahak.kuluttajaluottosopimus
Identifier for the agreement
Set kuluttajaluottosopimus.tunniste
APR
Set kuluttajaluottosopimus.vuosikorkoprosentti
Compiance to Consumer Protection Act
Set kuluttajaluottosopimus.tehtyKulutsuojalainEdellTavalla
Optional
Expiration date
Set kuluttajaluottosopimus.eraantymispv
Yes
2010-2019?
After 2019?
Yes
Vehicle or dwelling?
Yes
Set kuluttajaluottosopimus.viikennevalineTaiAsluotto = true
Loan amount
Set kuluttajaluottosopimus.maaraEur
Set kuluttajaluottosopimus.viikennevalineTaiAsluotto = false
Actual end date
Set kuluttajaluottosopimus.paattymispv
Interest rate in the agreement
Set kuluttajaluottosopimus.sopimuksenMukainenKorko
End

## Create Receivables



## Set saatava.viivastyskorko


Start
Type of interest
Set viivastyskorko.laji
Set viivastyskorko.viivastyskorkoKdi
Code value for interest
Interest marginal
Set viivastyskorko.marginaali
Set viivastyskorko.prosentti
Interest percentage rate
Code value for start date
Set viivastyskorko.alkuPvmKdi
Set viivastyskorko.koronAlkuPvm
Start date
Set viivastyskorko.maaraytymistapa
Section and subsection of Interest Act
Interest reset date
Set viivastyskorko.tarkastusPrm
Yes
Interest reset date?
Interest end date
Set viivastyskorko.koronLoppuPvm
Yes
Interest end date?
End