

Developing the ServiceNow App Accelerator

Entwicklung des ServiceNow App Accelerators

Master Thesis

Submitted in partial fulfilment of the requirements for the degree of

Master of Science in Engineering

to the University of Applied Sciences FH Campus Wien

Master Degree Program: IT-Security

Author:

Adrian Baginski, BSc (WU)

Student identification number:

1610537014

Supervisor:

ao.Univ.Prof. Mag. Dr.rer.soc.oec. Rony G. Flatscher

Secondary supervisor:

Priv.-Doz. Mag.rer.soc.oec. Dipl.-Ing. Dipl.-Ing. Dr.techn. Karl M. Göschka

Date:

May 7, 2018

Declaration of authorship:

I declare that this Master Thesis has been written by myself. I have not used any other than the listed sources, nor have I received any unauthorized help.

I hereby certify that I have not submitted this Master Thesis in any form (to a reviewer for assessment) either in Austria or abroad.

Furthermore, I assure that the (printed and electronic) copies I have submitted are identical.

Date:

Signature:

Preface

This master thesis was written under supervision of Professor Flatscher from WU Vienna and valuable support from the software engineering company and ServiceNow partner Wrangu, that allowed for exceptional insight into the workings of ServiceNow. They assisted by inviting me to a ServiceNow summit in Amsterdam to see the capabilities of cloud computing and how the Now Platform successfully integrates in that ecosystem, which helped a ton when getting started with ServiceNow Studio to develop the ServiceNow App Accelerator and writing the chapter "Exploring the ServiceNow Platform" of this master thesis.

Wrangu generally specializes in creating applications for the cloud such as the GDPR Data Scanner, which is an essential tool in ensuring that a ServiceNow instance does not fall foul of the impending European law. [1] Others are custom built solutions for various enterprises, which most often have the common problem that they are looking to migrate local shadow IT systems, typically built at the department or team level using languages such as PHP supported by MySQL databases to the ServiceNow cloud. This gives transparency and allows for oversight and governance as some of these shadow systems may contain sensitive data.

Kurzfassung

Diese praktische Arbeit beschäftigt sich mit der fehlenden Funktionalität des Cloud- Computing Anbieters "ServiceNow", bereits bestehende SQL Datenbanken automatisiert in die ServiceNow Plattform zu überführen. Die Relevanz ergibt sich durch das Bedürfnis vieler Organisationen mit tief etablierten, jedoch veralteten Datenbanken, ihre IT-Systeme in die Cloud zu migrieren. Um diese vollends abschließen zu können ist es notwendig, die wertvollen Daten selektiv aus den Datenbanken zu extrahieren und automatisiert in die ServiceNow Cloud hochzuladen. Mit dem jetzigen Stand der Technik ist dies jedoch nicht möglich. Die Plattform motiviert ihre Benutzer, die bereitgestellten Applikationen mit neuen Datenstrukturen zu verwenden, um einen optimalen Neuanfang in der Cloud zu gewährleisten.

Mit Hilfe der Programmiersprache Open Object Rexx wird eine Desktop-Applikation erstellt, welche mit SQL Konnektoren und eigens implementierter REST API diese Lücke schließt. Abschließend werden im Zuge eines Funktionstests zwei MySQL Tabellen einer Datenbank mit insgesamt 1.600 Datensätzen hochgeladen und auf Übereinstimmung überprüft.

Nebenbei werden sowohl ooRexx und das BSF4ooRexx Framework analysiert und dem Leser mit kurzen und zugleich einfachen Beispielen veranschaulicht. Die Erforschung von ServiceNow's aktueller und vergangener Marktsituation, den Möglichkeiten und seiner Konkurrenten sorgt für einen wissenschaftlichen Neuheitswert, da dieses Unternehmen trotz seines rasanten Wachstums in diesem Umfang noch nicht analysiert wurde.

Abstract

This practical thesis analyses the Platform-as-a-Service provider ServiceNow and documents the creation of a desktop application to mitigate the lack of functionality to import existing data in an automatic manner to the platform. Although ServiceNow tries to win its customers with performing applications provided by them, which handle data in non SQL related data structures, organisations presumably have valuable assets in existing yet most often legacy shadow database IT systems which they cannot leave behind while moving to the cloud. As a result, the migration causes those organisations to gain an additional system they need to keep track of instead of simplifying their IT infrastructure to just use the services provided by the cloud and leaving the old ones behind.

The ServiceNow App Accelerator provides a solution with the help of the programming language ooRexx and the BSF4ooRexx framework. In order to create new table structures programmatically in ServiceNow, a custom REST API is developed to serve this functionality. The Application uses JavaFX to build a complex graphical user interface with the focus on usability, user experience and fault tolerance. Afterwards, the functionality of the app is tested with 1,600 data sets that are successfully uploaded to a developer instance of ServiceNow.

List of Abbreviations

API	Application Programming Interface
ASP	Active Server Pages
App	Application
BSF4ooRexx	Bean Scripting Framework for ooRexx
CSO	Chief Security Officer
DES	Data Encryption Standard
DNS	Domain Name Service
FXML	JavaFX Extensible Markup Language
GNU	General Public License
GUI	Graphical User Interface
HR	Human Resources
IBM	International Business Machines
IT	Information Technology
JDBC	Java Database Connector
JFXAT	JavaFX Application Thread
JSON	JavaScript Object Notation
ooRexx	Open Object REXX
OS	Operating System
PaaS	Platform as a Service
REST	Representational State Transfer
REXX	Restructures Extended Executor
SAA	ServiceNow App Accelerator
SHA	Secure Hash Algorithm
SN	ServiceNow
SQL	Structured Query Language
SaaS	Software as a Service
URL	Uniform Resource Locator
XML	Extensible Markup Language

Keywords

JavaFX

ServiceNow

SQL Databases

Migration

Open Object Rexx

BSF4ooRexx

Contents

1. Introduction	1
2. Exploring the ServiceNow Platform	2
2.1. The Cloud Computing Market	2
2.2. The ServiceNow Approach to Cloud Computing	3
2.3. Architecture	5
2.4. Building Apps for the Cloud	6
2.5. Security Operations	7
2.6. Migration issues	8
2.7. ServiceNow Alternatives	10
2.8. Conclusion	11
3. Programming with Open Object Rexx	12
3.1. Nutshell Examples	13
3.2. The BSF4ooRexx Framework	15
4. Developing the ServiceNow App Accelerator	18
4.1. Possible Migration Strategies	18
4.2. Project Overview	20
4.3. Project Structure	22
4.4. Application Bootstrap	24
4.5. Configuration	34
4.6. Manage Credentials	51
4.7. The Import Task	58
4.8. Functional Testing	75
5. Related Work	80
6. Improvements and Outlook	81
Bibliography	85
List of Figures	85
Listings	86
A. Appendix	88

1. Introduction

The cloud computing market has experienced immense growth since 2010, reaching a volume of \$ 20.1 billion solely in Germany in 2017. It is dominating the IT industry worldwide today. [2] The concept of cloud computing in general makes use of huge data centre infrastructures, which can potentially cumulate their processing power for intensive calculations. However, the most common use is to store data in the cloud, process it in a predefined way and return it to the user so that he can further work with it. This enables large-scale distributed systems services without any notable up-front investments.

This master thesis provides an overview of the current cloud computing market and introduces the key concepts of the ServiceNow platform alongside its architecture in chapter 2. A brief description is given how users can create applications for the ServiceNow cloud and how to enforce Security Operations using the latest security technology provided by the platform. Then, the three main competitors in the IT service management and project management business are compared to ServiceNow.

The following chapter 3 describes the programming language Open Object Rexx alongside its core strengths that can be utilized to easily build desktop applications. The BSF4ooRexx framework is also analysed to extend the possibilities of Open Object Rexx by bridging Java and thereby using all of its classes and interfaces as if they were a huge third party library for ooRexx.

Chapter 4 is the core of this master thesis and builds on top of the previously gathered theoretical concepts to develop an application for the ServiceNow platform, that is going to be published to the Open-Source community eventually. This application titled "ServiceNow App Accelerator" allows an external SQL database to be replicated into a scoped application on the ServiceNow cloud platform. It is an application generator; examples of these tools that are non-ServiceNow related are SQL Maestro's PHPGenerator or Easy Code ASP Generator. The costs of these tools vary between 200 dollars for a developer license to 1200 dollars for a site license.

ServiceNow provides a Platform-as-a-Service and Service-as-a-Service solution used by enterprises globally to transform their internal processes from IT to Human Resources. The ServiceNow platform provides out of the box applications to support IT Service Management, Risk and Compliance, Security Operations, IT Operations and automation. It also provides a development environment for the creation of bespoke applications that can take full advantage of platform functionality. This allows lightening speed development of solutions to solve business problems in the cloud. ServiceNow markets their platform as prime for converting custom built departmental applications onto their platform to take advantage of the single source of record and other platform features.

Chapter 4 describes all relevant project files and the main application flow in a clear manner so that other developers can potentially use this to extend the feature base or to integrate the ServiceNow App Accelerator in other enterprise applications. This chapter concludes with a real world test scenario, where a MySQL database with two tables is successfully uploaded to a ServiceNow instance.

Chapter 5 describes the related work of this master's thesis. Finally, chapter 6 depicts a few improvements to the application that can be made to enhance user experience.

2. Exploring the ServiceNow Platform

ServiceNow provides a Platform-as-a-Service and Software-as-a-Service solution to business customers, who wish to neglect annoying IT administrative tasks so they can focus on their core businesses again. Thus, the deployment model decides the way of delivering the services. The Platform-as-a-Service paradigm allows to use associated services over the Internet without downloads or installation. [3] With the help of an extensive application market place and the ServiceNow Studio App, the Now Platform can be tailored to any enterprise's needs and redefine previously cumbersome IT systems.

2.1. The Cloud Computing Market

Cloud computing has a huge impact on the modern society as it has become very influential to the global economy. The total worldwide market of cloud computing was 2016 at \$209 billion and is expected to grow to an astonishing \$383 billion by 2020. [4] That is an annual increase of more than 20 percent, which makes cloud computing one of the fastest growing industries in the 21st century so far. But what is "cloud computing" actually? According to the official NIST definition, it involves a computing procedure in a shared pool of computing resources such as networks, servers and storage. It comes with minimal management effort or service provider interaction. [5] Moving a business' IT infrastructure to the cloud comes with lots of benefits. Cloud computing has the following key characteristics: [6]

- Cost efficiency
- Almost unlimited storage
- Backup and recovery
- Automatic software integration
- Easy access to information
- Quick deployment

Cloud computing is cost efficient, as customers only pay for the cloud resources they consume. There is practically no cap for storage space, so that customers can always upgrade if they need to. From a safety standpoint, backup and recovery plans are very easy to follow because the usual cloud computing infrastructure already includes some kind of redundancy. That is why the customer does not have to care about backup plans whilst being able to recover his whole infrastructure within seconds in case of a breach.

Software integration is something that occurs automatically. The customer only has to choose from a range of services and applications which suits his business best. The access to information is also very easy, as long as an internet connection is available. Generally speaking, the deployment phase is ought to be short.

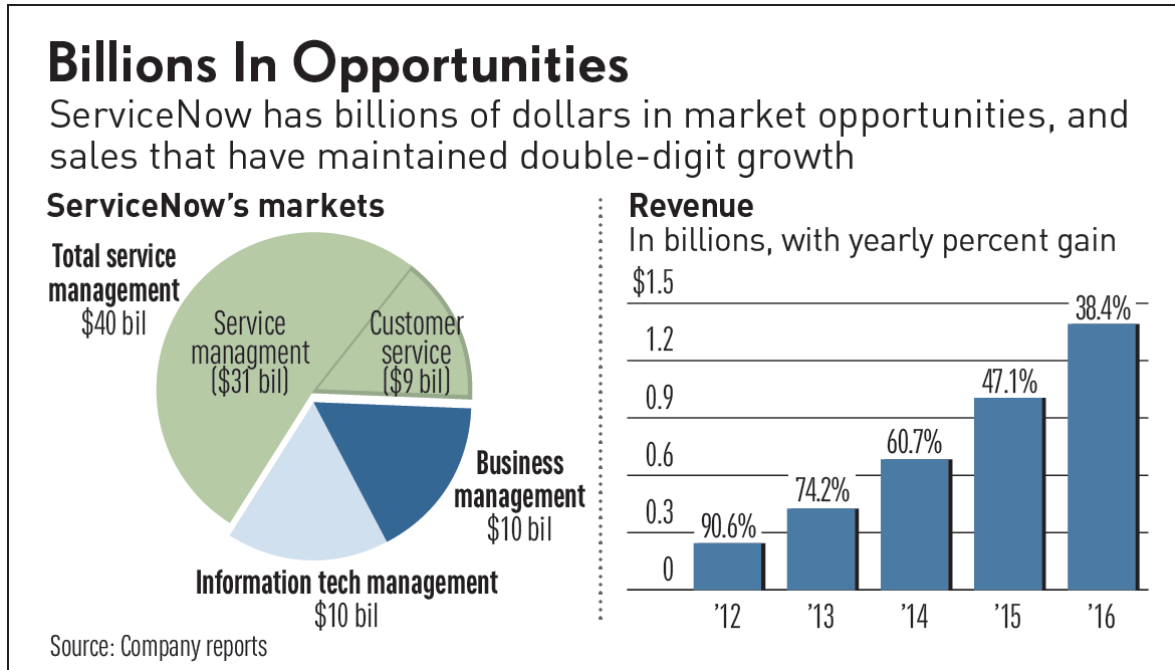


Figure 2.1.: ServiceNow market opportunities [7]

2.2. The ServiceNow Approach to Cloud Computing

ServiceNow is a new cloud computing provider with a focus on business-to-business solutions. Its revenue in 2016 exceeded \$1 billion for the first time in history and is expected to reach \$4 billion by 2020. According to the third quarter financial report in 2017, the year-over-year growth of subscription revenues is at 43 percent, while the gross profit accounts for astonishing 82 percent. Those operating numbers show just how fast this industry operates. Besides, ServiceNow is not even using all its market potential, as figure 2.1 shows. ServiceNow already serves the Service management market, Customer service market, Information technology and Business management market but could easily expand to others as well. [7]

ServiceNow offers a Platform-as-a-Service solution. This means, that its customers are able to use acquired, provided or self-programmed applications on the platform. Those apps make use of supported libraries, programming languages, services and tools. The customer is exempted from managing all the underlying systems including the servers, operating systems and the network. He only deploys and uses applications. That is essentially what a business needs to do in order to generate value. [8]

The ServiceNow platform provides the infrastructure needed to develop, run and manage applications. Being cloud based has the advantage that the data does not reside on only one device, therefore it is accessible from anywhere in the world using any device as long as it has a stable internet connection. The platform provides a variety of ready-to-use applications to automate the work-flow of everyday tasks across all departments of a business, as shown in Figure 2.2. The implemented applications can be divided into those three sections:

- Service Management
- Business Management
- Operations Management

Service Management is by far the biggest sections containing all service oriented compo-

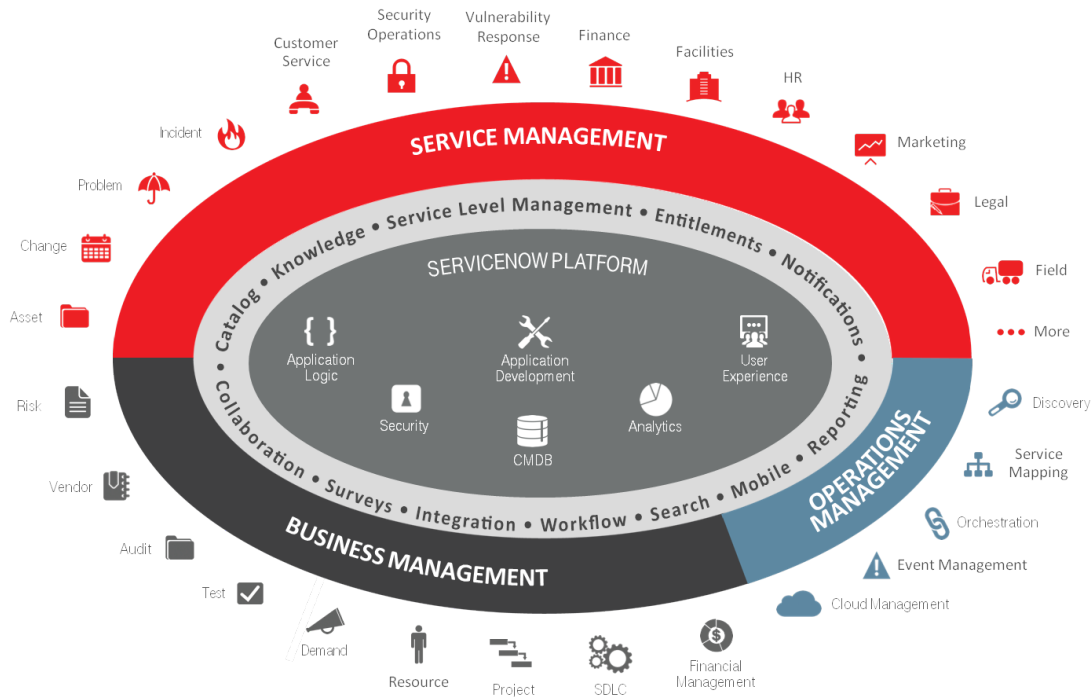


Figure 2.2.: Components of the ServiceNow Platform [10]

nents like Incident Management, Service Catalog, the Configuration Management Database and facilitating department applications. Operations Management is rather infrastructure oriented, so it involves managing Servers, Applications, Storage, Virtualization, Cloud and Network. Those two sections are able to interact with each other to maximize profitability of the system. This substantial communication happens when notifying Service Management about the latest events and alerts of the System or Operations Management reacting to changes based on Service Management actions.[9]

It is imperative to notice that all applications inside a section can also interact with each other - that is a key feature of the ServiceNow platform. This was designed based on the assumption that most business processes have a vibrant ecosystem and generally make use of several departments. Take the onboarding process of a new employee as an example. This is a typical work-flow that every business has to partake regularly, despite being always a headache for its participants. Onboarding an employee means to use the HR department for most of the process, IT for getting equipment and setting it up properly, Security for a system briefing, Facilities for a premises induction, Finance for the transactions and Legal department for legal matters. If the new employee has special requests like a specific Notebook that he would like to use, than the described work-flow might get really complicated, since he is not directly communicating with the executive department.

The ServiceNow platform offers a work-flow integration, where its customers can set up their most common processes to enable automation. This way, the platform does most of the necessary work like sending the right E-Mails to the right person from there on. That is how ServiceNow redesigns a process that has always consisted of mostly unstructured, slow and hard to optimize data flows to one where following business flows makes fun again, given that they initially sacrifice their time to configure the platform according to their needs.

2.3. Architecture

The ServiceNow platform uses a single system of records and a common data model to consolidate all business processes. Most other clouds are built on a multi-tenant architecture, which means that the hardware stack has the following composition:

1. Hardware
2. Virtual machine
3. Operating system
4. Middleware
5. Application

Multi-tenancy aims to provide isolation of the different cloud tenants with full independence while optimizing resource sharing capabilities. Due to the enormous popularity of this architecture it is well-known in the community, despite the fact that it is not the best solution for building clouds with multiple users. This traditional approach has the disadvantage that the data is commingled with other customers of the same cloud since it uses only one server instance to serve them all. Due to this property, there is a high risk of data being leaked or a possible unprivileged access. [3, p.55] Such architecture solution relies on large and complex databases that require hardware and software maintenance on a regular basis, which could lead to availability issues. Additionally, any action on the multi-tenant databases affects all customers since they strongly depend on it.

The ServiceNow architecture builds on an advanced technology they call "multi-instance". The composition is slightly different to the traditional approach, since every customer has his own server instance, that consists of the usual stack described in the multi-tenant model, but starting at the virtual machine as depicted in figure 2.3. Therefore the only layer all customers have in common is the Hardware layer - if one would allocate hardware to customers they would have a cloud on their own. That is why the multi-instance approach has the most advanced form of separation in every regard, at the expense of higher maintenance complexity and infrastructure costs.

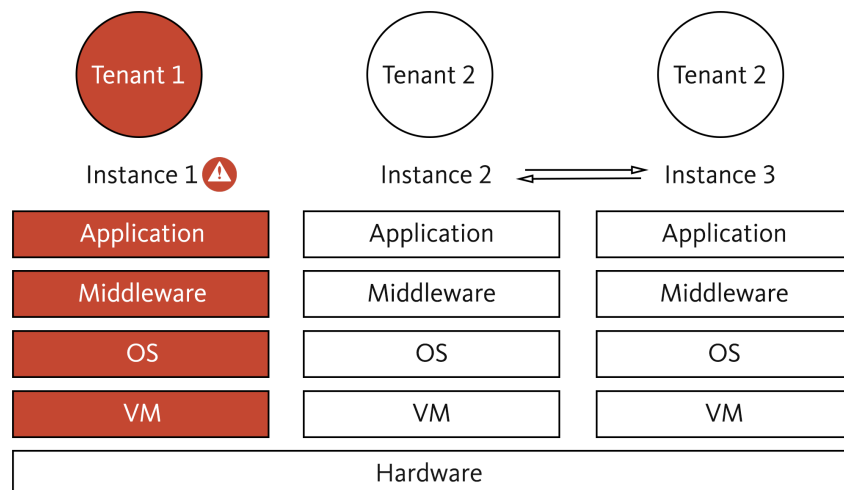


Figure 2.3.: Multi-instance architecture model

Each customer can have as many instances as he needs. All instances are isolated but can still communicate with each other on the Application layer. Nevertheless, this architecture

accomplishes true data separation which can be observed if an instance suffers a malfunction of some kind, then this only instance will be affected by that issue. This property provides the best quality of service for all tenants and makes locating and maintaining bugs of both hardware and software on every instance much easier to perform. [11]

2.4. Building Apps for the Cloud

Using applications is a key concept of the ServiceNow cloud. The platform is divided into many pre-installed applications grouped into modules. They have typically a designated functionality, which is built to serve a specific purpose for different types of enterprises. Despite the large quantity of those supplied applications, it could be necessary for a business owner to get a hold of an app tailored to his needs. In general, one can distinguish between the following types of applications:

- Configurable pre-built apps
- Apps bought from the store
- Open-source apps provided from the community
- Self-built apps

The app store offers more than 300 free and paid applications that can be installed with only a few clicks by a privileged user account. Whilst using pre-built apps is a valid option most of the time, the platform also includes a system application called "ServiceNow Studio", which can be utilized to build apps. This is especially useful if a business has very individual needs. Using ServiceNow Studio, one can create both frontend and backend applications with a graphical user interface, some worker scripts, databases, notifications and many more. Figure 2.4 shows the different types of files that can be created using ServiceNow Studio. The selected data model group holds tables and columns that can be used as a persistent data source for other application files.

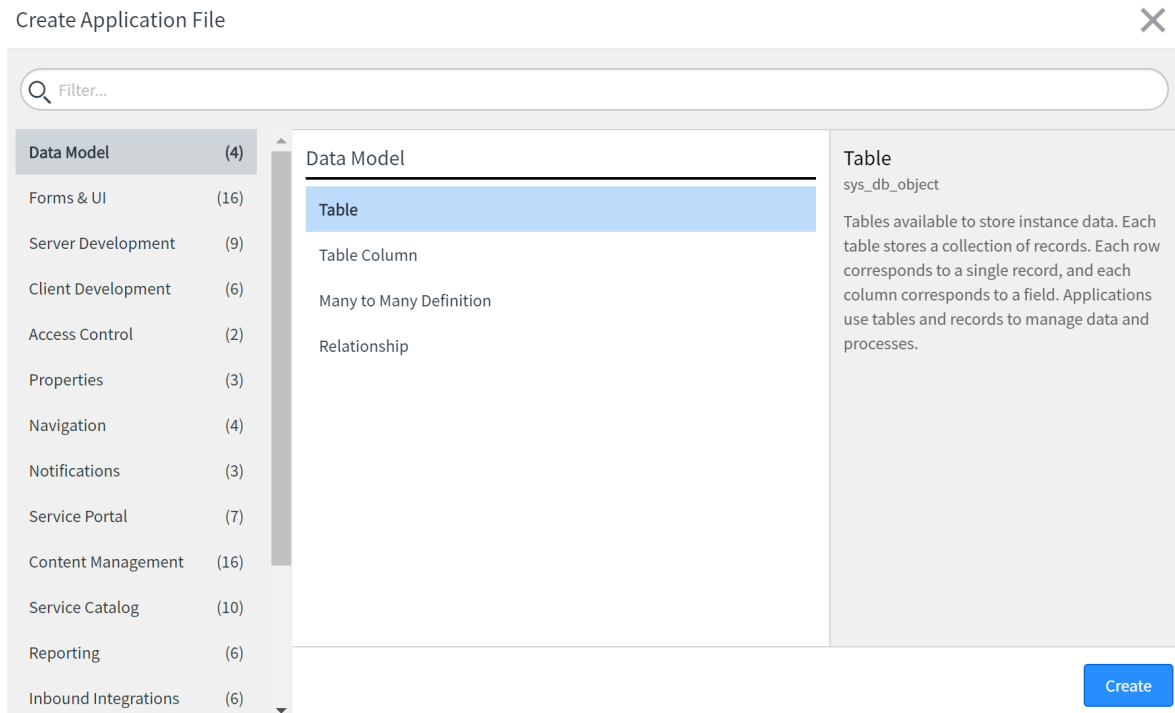


Figure 2.4.: "Create Application File" Mask (Source: Screenshot)

ServiceNow Studio enables teams to develop applications together using a source control system to apply remote changes or to create new branches and release tags. Once an app is ready to be published, it can be exported as a "Update Set", which is an XML file that can be easily imported in any ServiceNow instance to use the application it holds.

Managing businesses successfully requires engaging the competition with innovation and constantly searching for market opportunities, as well as responding quickly to customer needs. Especially in the cloud-first era, enterprise software is a great enabler of increased business velocity. ServiceNow Studio allows even inexperienced businesses to build advanced applications using resources the platform provides. [12]

2.5. Security Operations

Organizations invest in a lot of security products which are mainly for protection and detection of security threats. But when it comes to responding to an ongoing threat they all draw blank. A recent CSO study suggests that the average enterprise uses more than 70 different security products alongside security experts to control all the redundant products and manage useless alerts.[13] Despite that, it takes on average nearly three quarters of a year to spot a breach and another two months afterwards to contain it. The top ten exploited vulnerabilities in 2015 were over one year old, whereas 48 percent of them were even older than 5 years. The current situation shows that enterprises are having serious troubles securing them against digital threats.[14]

ServiceNow concentrates on providing a security response platform to maximise remediation efforts, streamline response processes and to provide intuitive dashboards that reflect the company's security disposition. The platform has a few tweaks to achieve that in the simplest possible way. First off, all pulled data from different sources is transferred into

a single system record, where it gets prioritized based on incident workload. This is only possible if the platform understands business criticality of all enterprise assets. ServiceNow enables IT and security teams to work from the same system so that information gets routed to the appropriate teams and people. Of course all basic tasks and processes are automated to minimise threats by human error. [15]

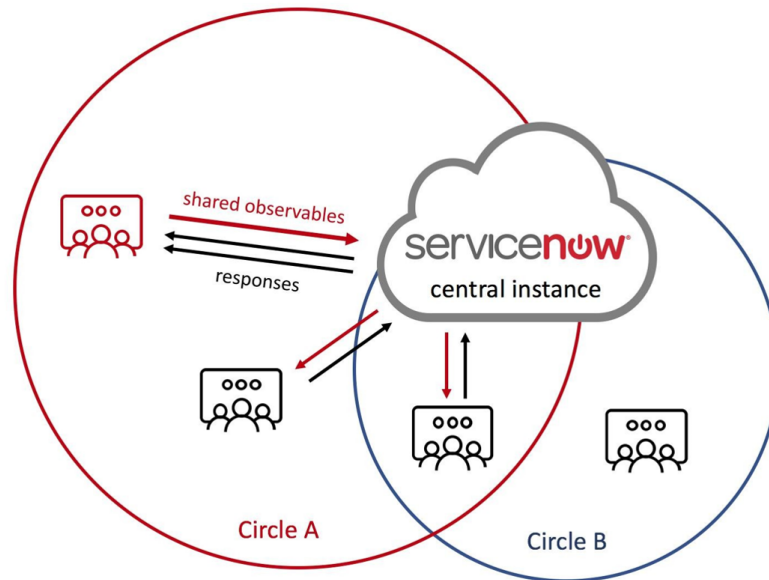


Figure 2.5.: Trusted Security Circles [16]

ServiceNow extended the threat intelligence business by introducing "Trusted Security Circles" in the latest build of their platform. Those have the purpose to share security relevant data with members of the supply chain or industry peers based on commonalities using current industry standards such as the Structured Threat Information eXpression (STIX) and the Trusted Automated eXchange of Indicator Information (TAXII). By sharing anonymous observations with other businesses that are also potentially targeted for an exploit, it makes it possible to get early warnings of an attack and react accordingly, as depicted in figure 2.5. The concept behind this functionality is that no enterprise truly exists in isolation. There are also distributors, outsourcers, IT software vendors, affiliates, consultants and resellers that every business depends on - if they go down, you do too! [16]

2.6. Migration issues

A legacy information system consists of one or many applications with its environments, whose technological components are outdated but still in use. The development of such systems was once a huge, long-term investment, but nowadays they suffer from low performance and non-extensibility. Migrating those well-established systems to the cloud can be a very exhausting process, especially for bigger organisations with diverse assets. Capturing legacy system data in a way that can support organisations into the future is thus an important research area. [17] They are critical for the organisation and their failure can have a serious impact on the business. [18]

The main issues with legacy information systems are, that they run on obsolete hardware

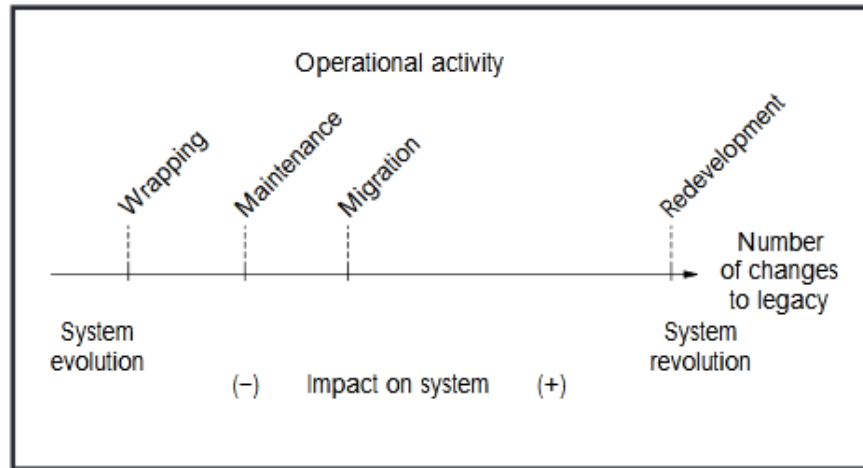


Figure 2.6.: Solutions to legacy information systems [17, p.104]

with high maintenance costs and low performance, and that the software that runs on them suffers also from those exact weaknesses. Thus, they can be difficult or even impossible to extend or integrate with other systems. Additionally, obsolete documentations cause another barrier so that organisations either leave such legacy systems running without maintaining them or migrate to an alternative.

There are several solutions to legacy systems that can be generally categorized into the following:

- Wrapping
- Migration
- Redevelopment

Wrapping the legacy information system in another system that has more and better interfaces to interact with is only a short-term solution that can even complicate maintenance in the long run and is therefore generally not recommended solution. Redevelopment on the other hand is a complete redo of the legacy system and requires a tremendous amount of resources such as time, money and personnel. It has the most impact on the system as it comes with the most number of changes to the legacy system.

The most reasonable approach is therefore migration, which essentially means to move a running system to a new platform while retaining the functionality and ideally causing as little disruptance as possible. [17] This solution sounds difficult to perform, but has some welcoming benefits if done correctly such as higher flexibility and reduced maintenance costs.

Migrating databases requires to first map the legacy information system's schema onto the target schema before working out the transformation. It can be necessary to also cleanse the data both before, during and after the migration if it is of poor quality.

The final phase of migration can be one of the following.

- The cut-and-run strategy
- The phased interoperability strategy
- The parallel operations strategy

The cut-and-run strategy performs a full switch of systems on a new feature-rich replacement. The phased interoperability strategy on the other hand uses the cut-and-run strategy

gradually for some parts of the application at a time, until the switch has been done. In the parallel operations strategy, both systems are running and responding simultaneously until the replacement system has been tested and trusted fully to shut the legacy information system down.

The decision upon which strategy to use depends on many factors bound to the organisation. In any case, if performed successfully, the migration to a cloud computing provider results in high performance boosts and other benefits as described in chapter 2.1 on page 2.1. Many organisations have to face the same issues at this moment of time until some day businesses will rise starting their venture directly in the cloud instead of creating - what will later become - legacy information systems.

2.7. ServiceNow Alternatives

ServiceNow might not always fulfill every company's needs and might even be too expensive for others. Certainly there are quite some competitors in the markets ServiceNow serves, especially in the expertise of Project Management Software. The most relevant alternatives are Mavenlink, Jira ServiceDesk and Redmine. None of them offer quite the same features as ServiceNow does, but they do have their own unique selling proposition on the market. [19]

Jira ServiceDesk is developed by the Atlassian Corporation, that also offers many additional products such as Confluence as knowledge base and Bitbucket as version control system. Jira is an issue tracking software with project management functionality and extensive bug tracking support with an integrated ticketing system. Its main strength lies in using all Atlassian products simultaneously, as they are designed to imperceptibly work together. A user can reference a wiki article from Confluence when creating a new issue in Jira and attach a Bitbucket project, so that all future changes will be also logged in Jira. Contrary to ServiceNow all those products have to be installed individually on the customer's hardware. This way they run fully isolated from the vendor, but the customer possibly has to outsource the server related processes to another enterprise or bother with that on his own risks and expenses. [20]

The most promising competitor Mavenlink provides collaborative resource planning and a project management software in a single application for delivering project-based services. It also includes project accounting and business intelligence features, so that it can work independently of the company's IT structure. All generated data is stored in the Mavenlink cloud with confidence and the pricing for professional teams starts at 39 dollars per user and month. [21]

Redmine is an open source project under the terms of the GNU General Public License v2. It is a cross-platform and cross-database project management web application that can be downloaded and set up on any server. It features a gantt chart and calendar generator, news, various feeds, per project wiki and forums, a time tracking system, Supply-Chain-Management integration, an issue management system and some more. Overall, Redmine does not have quite the rich feature base as the other mentioned competitors but it is a valid option for businesses with a tight budget to also manage projects and IT service management professionally. [22]

2.8. Conclusion

Once a venture reaches a certain growth it becomes very important to keep up with the competition by providing efficiency across all business divisions. One simple way to do so is to outsource IT related processes to the cloud. This way the cloud provider has to master the background work associated with such processes, like a high availability, security across all realms, backups and redundancy by middleware.

ServiceNow offers its customers a profound experience in automating most of the usual processes so that they can fully focus on their core tasks again. To achieve that, ServiceNow provides a platform which is accessible from any device through a web browser. To maximise efforts it is required to customize the necessary dashboards, applications and service portals to the business' needs.

The ServiceNow platform additionally offers a wide range of built-in security tools to analyse all relevant events. Using those tools, a business can detect and prevent attacks without the know-how of an IT-Security specialist. One such technology is called Trusted Security Circles, where threat information is shared with selected industry partners. This solution makes use of the fact that multiple businesses interconnected by a supply chain are most likely to be targeted by the same attack. Thus ServiceNow keeps publishing security applications, their success is highly dependent on customer acceptance. Nevertheless, they ultimately raise awareness for risks, vulnerabilities and security in general as those fields of expertise grow in importance with the business' value.

While ServiceNow takes good care of security and operations tasks, it naturally comes with privacy concerns, as all data is stored in their own data centres. Their customers have to heavily rely on ServiceNow not to leak or aid leakage of any sensitive information to the public or competitors. Additionally, the source code of the platform is closed-source, so no one really knows what happens behind the scenes of the front end application. It is possible that they grab more information about its users than they should. On the other hand, hackers have a hard time penetrating the system if they do not know the code they are facing.

All in all, ServiceNow provides a great solution for a selection of mostly international enterprises. Others will prefer to invest in private cloud solutions or huge internal networks for privacy reasons, despite the potentially higher costs. [23]

3. Programming with Open Object Rexx

Desktop applications are traditionally programmed using compiled languages, because they usually offer the best performance. One of the most widely used representatives is C, alongside its object oriented successor C++. The compiler of such languages transforms the source code into optimized binaries, that can only be understood by a specific operating system with an explicit processor architecture. This involves compiling multiple times for numerous platforms. Java is another widely used programming language with a different approach, as it enables its users to install a runtime engine holding a virtual machine for the programmers to compile their applications only once, targeting that exact system.

While compiled languages definitely offer the best performance, hardware limitations on modern computers have dropped to an insignificant level, so that performance is not that vital anymore. Naturally, when it comes to gaming and big data calculations, those solutions are still indomitable - but most applications only use a fraction of memory and processing power. They do not require a complex compiled language that decelerates the development process. This is where scripting languages are at their best. [24]

It is no coincidence that Python as the leading scripting language has gained massive popularity in the community over the past years, resulting in even outpacing Java as the most popular programming language in 2017. That is because scripting is generally easier for a programmer than compiling. It involves fewer steps to get an application running, because it does not have to be compiled before running. [25]

Open Object Rexx is a scripting language which tries to minimize the programming burden for the user. It is case-insensitive and human-centric in all its instructions. There is only punctuation involved if it is really necessary, like for mathematical operations or comparisons. Variable usage is very easy too, as they do not have to be declared or initialized. Every variable is a string by default, but if it can be interpreted as a decimal number, one can use it as if it were an integer in any strictly typed programming language. Of course, the programmer is not forced to use a semicolon at the end of every instruction, as long as there is only one per line. [26]

Object Rexx offers an object oriented approach with easy concurrency control and some innovative merits. Classes are defined using the `::CLASS` directive and can hold multiple methods and attributes. In case an object calls a method that is not implemented in the class, the message will be forwarded to the special *unknown* method alongside the original method name. This feature can be used to counteract runtime errors by reacting to unforeseen events.

When a Rexx program is about to launch, the Rexx Interpreter reads the file content twice. At the first time, all strings that are not enclosed in single or double quotes are transformed to upper case and only so called "Directives" are executed by the interpreter. Those can be external or internal resources, classes, methods, attributes or routines. They always start with two colons and are often placed at the bottom of an application to separate the main application flow from the definitions.

The Rexx language was originally invented in 1979 by IBM as "Restructured Extended Executor" to replace a legacy batch language that is to this day used for IBM mainframes.

Over the course of time, REXX became the leading scripting language for various operating systems such as Amiga OS and OS/2. In 2004, IBM released Object REXX as open source software, which got adapted to ooRexx by the Rexx Language Association. The latest official version is 4.2.0, although the developers are regularly releasing beta versions of ooRexx 5.0 since 2014. [26]

3.1. Nutshell Examples

Executing ooRexx applications is fairly easy, since the installation process does not require any dependencies. Sourceforge provides binaries for Windows, Mac and some Linux distributions to get ooRexx running without any issues. [27].

3.1.1. Loops and Arrays

Listing 3.1 shows a simple "Hello World" program written in ooRexx.

```
1 loop i = 1 to 20 by 2
2   say "Hello World from ooRexx:" i
3 end
4 -----
5 myArray = .Array~of("Array", "handling")
6 say myArray[1] myArray~at(2)
7
8 myArray~append("is nice!")
9 loop item over myArray
10   say item
11 end
```

Listing 3.1.: Loops and Arrays in ooRexx

This nutshell example shows a basic loop its first three lines. It repeats the loop content until the variable `i` reaches the value of `20`, increasing the variable value by `2` after each iteration, which results in an output of 10 lines of "Hello World from ooRexx".

Line 5 initiates a new instance of the array class with two items. The tilde character `~` is a message operator that can be used to invoke methods. If two tilde characters `~~` are used together, the output of the operation is neglected.

Line 6 demonstrates two equivalent ways of accessing array values. One can either use square brackets like as used in C related programming languages or send the `~at` message with the index as parameter. Interestingly, ooRexx always starts collection class indexing with 1 instead of 0. That is attributable to the language being human centric by design, and humans also start counting with the number 1. This unusual behaviour ultimately results in a simplification for the programmer where he uses the n^{th} index to retrieve the n^{th} value.

Another widely used type of loop is demonstrated in lines 8 to 11. It shows the ooRexx implementation of a `forEach` loop. The final output of the program is shown below:

Output:

```
Hello World from ooRexx: 1
Hello World from ooRexx: 3
[...]
Hello World from ooRexx: 19
Array handling
Array
```

```
handling
is nice!
```

3.1.2. Object Orientated Programming

The next nutshell example in listing 3.2 demonstrates the huge advantage of two globally accessible directory objects `.local` and `.environment` as well as custom class creation in general. First off, the `::CLASS` directive is used to create a class, followed by the constructor `::METHOD` `init` to save the provided parameter at the instantiation in the attribute "type". Then, it defines the methods "whoAmI?", "getBroadcast" and "sendBroadcast". Rexx allows to use certain punctuation characters such as explanation marks, dots and question marks for naming variables, routines and methods.

```
1 windows = .computer~new("Win10")
2 windows~~whoAmI? -
3     ~~getBroadcast -
4     ~~sendBroadcast("Is there anyone?")
5 mac = .computer~new("MacOS")
6 mac~sendBroadcast("Yeah, Mac rockz!")
7
8
9 ::class "Computer"
10 ::attribute type
11 ::method init
12     expose type
13     use arg type
14
15 ::method whoAmI?
16     expose type
17     say "I am" self~defaultName "of type" type
18
19 ::method getBroadcast
20     if .local~latestMessage = .nil then
21         say "No messages yet"
22     else
23         say .latestMessage
24
25 ::method sendBroadcast
26     expose type
27     use arg message
28     .local~latestMessage = type || ":" message
29     self~getBroadcast
```

Listing 3.2.: An advanced example demonstrating object-oriented functionality

Output :

```
I am a Computer of type Win10
No messages yet
Win10: Is there anyone?
MacOS: Yeah, Mac rockz!
```

The main application flow starting in line 1 creates two instances of the "Computer" class and calls all defined methods to send and receive messages. The keyword `expose` exposes the method to a specific attribute in the class, and the instruction `use arg` saves the argument in the attribute for other methods in the class to use it. The keyword `self` references the object and is equivalent to "this" in C related programming languages. The method `getBroadcast`

also shows an if expression in line 21 with the `.nil` class. Object Rexx allows to use one equals sign `"=`" instead of two to compare values. The dot can also be used to reference environment variables, as demonstrated in line 24.

This nutshell example shows just how easy it is to make technically separated objects communicate with each other. They could also depend heavily on variables set in the main flow of the application, without the programmer ever needing to think about variable scopes. The second available environment directory object `.environment` can even communicate with other ooRexx applications that run in the a different Rexx interpreter instance.

3.1.3. Routines

The last nutshell example in listing 3.3 shows routines in ooRexx, which are defined using the directive `::ROUTINE`. The keyword `strict` requires the arguments to be set when invoking that routine. Line 4 shows how to call routines with the `call` statement, in contrast to the well known function invocation in line 3 using brackets around the arguments.

```
1 a = "2"
2 b = 3
3 say add(a,b)
4 call multiply a,b
5
6 ::routine add
7   use strict arg a,b
8   return a + b
9
10 ::routine multiply
11   use strict arg a,b
12   say a * b
```

Listing 3.3.: Calling routines

Output :

5
6

Overall, Open Object Rexx is a great scripting language with unconventional distinctions that take time to get used to such as tilde characters to replace dots and array indexing starting at 1. Nevertheless, it is an easy yet powerful language, which supports the programmer with dynamically typed variables, object orientation and plenty collection classes suited for every purpose.

3.2. The BSF4ooRexx Framework

The Bean Scripting Framework for Open Object Rexx exposes the Java infrastructure to the ooRexx programming language. Using this framework, it is possible to interact with Java classes and interfaces through the ooRexx proxy BSF class and a few related routines.

The Bean Scripting Framework is a set of Java classes that provide scripting language support for Java applications. It was originally invented by IBM and later handed over to the Apache Software Foundation. The BSF4ooRexx framework in its first versions implemented a `RexxEngine` class to use Rexx inside Java applications. It was extended to even support the other way around, where Rexx applications make use of Java classes. [28] The following nutshell example provides an overview of the most relevant BSF4ooRexx functionality:

```

1  /* static class usage */
2  call bsf.import "java.lang.System", "System"
3  .System~out~println("Hello World from Java")
4  say "Java Version:" .System~getProperty("java.version")
5
6  /* save the System class for all rexx interpreter instances */
7  .environment~System = bsf.import("java.lang.System")
8
9  /* class instantiation */
10 url = .bsf~new("java.net.URL", "http://www.example.com:1080/docs/resource1.jpg")
11 say "File:" pp(url~getFile)
12 say "Host:" pp(url~getHost)
13 say "Port:" pp(url~getPort)
14
15 /* java interface handling */
16 rexxProxy = bsfCreateRexxProxy(.rexHandler~new, , "java.lang.Runnable")
17 .bsf~new("java.lang.Thread", rexxProxy)~bsf.dispatch("start")
18 say "Main Thread ID:" bsfGetTID() -- fetch the current thread id
19
20 /* rexx handler class definition */
21 ::class rexxHandler -- this class implements the Runnable
22   interface
23   ::method run
24     tID = bsfGetTID()
25     .System~out~println("Sub Thread ID:" tID) -- let Java handle
26     the output
27
28 ::requires "BSF.CLS" /* get Java support */

```

Listing 3.4.: BSF4ooRexx nutshell example

Output:

```

Hello World from Java
Java Version: 1.8.0_144
File: [/docs/resource1.jpg]
Host: [www.example.com]
Port: [1080]
Main Thread ID: 13220
Sub Thread ID: 16368

```

The `bsf.import()` routine is used to import Java classes to access their static fields and methods. This is done by providing the canonical class name and optionally a string identifier, which will serve as the index in the `.local` collection for the imported Java class. This example imports the `System` class and extracts information about the current version of Java using the static `getProperty` method. It also uses the `System.out.println()` method to print a simple "Hello World" message.

Line 10 creates a new instance of the `java.net.URL` class by simply instantiating the ooRexx `.BSF` class with the canonical class name and a number of arguments that have to be passed to the constructor of the Java class. The returned Rexx object acts as a proxy and can forward messages sent from ooRexx to the corresponding Java object. Lines 11 to 13 invoke the `~getFile()`, `~getHost()` and `getPort()` methods implemented in the `java.net.URL` class. The `pp()` routine is defined in the BSF4ooRexx package and encloses the argument in square brackets to promote readability.

The routine used in line 16 expects a Rexx object, an optional slot argument and the

canonical name of an abstract class or one or more Java interfaces. It returns a `RexxProxy` object representing the given Rexx object on the Java side. Using this functionality, it is possible for Java to send messages to the Rexx object as shown in the nutshell example. In line 17, a new Rexx object is created representing the `java.lang.Thread` class with the `RexxProxy` from the previous line as argument. After that, the `start()` method from the `Thread` class is invoked using the `bsf.dispatch()` method from the `ooRexx` Object class. Usually it is sufficient to just send the method name as message to the object for it to invoke the right method, but in that case "start" is one of the few methods that are implemented in the `.BSF` class - so simply sending "start" would result in invoking the wrong method and not start the thread after all. For comparison, here is how the Java equivalent of starting a thread would look like:

```

1 public class HelloRunnable implements Runnable {
2     public void run() {
3         System.out.println("Hello from a thread!");
4     }
5
6     public static void main(String args[]) {
7         (new Thread(new HelloRunnable())).start();
8     }
9 }

```

Listing 3.5.: Starting a thread in Java [29]

One needs to instantiate the `Thread` class with a Java class that implements the `Runnable` interface and invoke the `start()` method for the `run()` method to run in its own thread. Using `BSF4ooRexx`, it is also possible to implement this functionality in `ooRexx`. The `bsfGetTID()` routine returns the current thread ID in lines 18 and 23 and prints it out to illustrate that the `run()` method really runs in its own thread.

Apart from the functions shown there are also other routines that can be utilized for some special cases such as the `box()` and `unbox()` methods to create a Rexx object from a specific primitive data type. This is actually something that happens under the hood for `ooRexx` programmers, as they do not have to care about data types at all. Though in case that there are multiple methods implemented in Java, each with different argument types it might happen that the framework invokes the wrong method, so using the `box()` routine might be the best way to solve this issue.

Actually `BSF4ooRexx` already saves the most important Java class objects in the `.local` collection, so the import of `java.lang.System` in lines 2 and 7 from listing 3.4 was not necessary, as this class is already accessible through `.java.lang.System` or `.local~java.lang.System`. The `.local~bsf4ooRexx` directory also holds interesting values, that can be fetched to get more information about the operating system:

```

1 say .bsf4ooRexx~file.separator -- returns "\" on Windows
2 say .bsf4ooRexx~path.separator -- returns ";" on Windows

```

All in all the `BSF4ooRexx` framework offers an easy yet powerful way to use all of Java functionality in the dynamically typed, caseless and human-centric `ooRexx` language. Consequently it is possible to use even cryptographic functions such as hashes and encryption services without having to implement them oneself. This framework mitigates the core issue of creating complex `ooRexx` solutions - that there are not enough libraries available to use. Java is fast, handles concurrency well, scales with ease, is built for security and has a great ecosystem. The Java Virtual Machine is cross-platform and uses run-time information to manage itself and takes care of memory management automatically. For those reasons it is a wise choice to build applications with Java support using the `BSF4ooRexx` framework.

4. Developing the ServiceNow App Accelerator

As of time of this writing, the ServiceNow platform does not offer any possibilities to import SQL Databases of any kind to fill existing tables with data or to create them in the first place. Mostly big enterprises move their capacious IT infrastructure to the ServiceNow cloud with legacy systems that are still running using background databases that do not have to be created from scratch. Those could hold information about all kinds of user systems, the employees, events, newsletter subscriptions and many more. Migrating to the cloud results presents an opportunity to significantly reduce maintenance costs. Since ServiceNow promises to replace legacy systems with the performing cloud, it is unnatural that businesses are forced to hold onto their old systems because moving their databases is generally considered difficult. That is why this master's thesis focuses on the development of an application to solve this issue. [3, p.40]

There is a set of REST APIs for application developers to use, which already includes a Tables API. This utilizes the functionality to programmatically [30]

- retrieve multiple records from a specified table,
- insert one record into a table,
- or delete a record.

Naturally, using this APIs is only possible from within external or internal applications, as there are no graphical user interfaces for those APIs. Apart from that, there are no REST APIs to create or modify table structures, which makes it virtually impossible to perform that task for external applications that do not have access to the ServiceNow infrastructure.

4.1. Possible Migration Strategies

The right choice of a migration strategy is crucial for an organisation to survive the transition from using legacy information systems to the cloud. Typically, there are many approaches that each have their respective field of application. One could distinguish between the following strategies.

- The Big Bang Approach
- The Database First Approach
- The Database Last Approach
- The Composite Database Approach
- The Butterfly Methodology

The Big Bang Approach refers to a redevelopment of the legacy system as stated in described 2.6 on page 8. Although the risk of failure is quite high, it is sometimes best to reinvent the wheel and leave old techniques behind, especially if the discussed application is of small size and not mission crucial. [31]

The Database First Approach first transfers all data to the new systems, and then follows with incrementally migrating legacy applications. This methodology has the huge advantage of gradually migrating to the cloud with two running systems simultaneously. Transferring the database first forces the new systems to be able to work with that data. All in all, this is a quite simplistic migration strategy. [31]

The Database Last Approach is a similar concept that transfers the legacy database after all new applications are running in the cloud. This prevalent method comes with disadvantages regarding low performance of the so called "Reverse Gateway" entity, which is responsible for mapping the target database schema to the legacy database. Another issue is, that complex features of the new database may not be used due to the lack of support in the old database, such as consistency constraints, integrity checks and triggers. [31]

Both the Database First Approach and the Database Last Approach are only applicable for fully decomposable legacy systems, where all functional components are separable. Contrary, the Composite Database Approach has a wider field of application as it is also available for semi-decomposable and non-decomposable information systems. This migration strategy involves operating both the legacy and target platform in parallel and gradually rebuilding all functions of the archaic information system until it can be shut down. This involves a powerful transaction co-ordinator to be set up between the applications and the databases, who is responsible for intercepting all update requests and forwarding those calls to the database with the corresponding information, albeit it is possible that sometimes both legacy and target databases have duplicate data sets. If they do, the update is sent to both databases to ensure consistency throughout the migration process. Another variant of the Composite Database Approach is the Chicken-Little Strategy, which is an 11-step incremental migration plan using complex gateways, that differs in functionality and placement of those gateways. An overview of these 11 steps can be found at [31, p.27].

The Butterfly Strategy is a novel approach that replaces all gateways used in the previously mentioned strategies with a module called "Database-Access-Allocator", whose responsibility is to save all manipulation calls to the database in temporary data storages while migration takes place. A data transformer is then used to perform those changes in the real databases. This results in an unimpeded availability of the target database as long as the temporary stores do not exceed their data limits. Certainly, this approach can only work in environments, where the speed of updating the data stores is greater than filling them, otherwise the target database would finish the migration. Therefore, it is possible that this is not the best strategy for high-traffic information systems. [31]

The ServiceNow App Accelerator can be used for the Database Last Approach for migrating legacy information systems, since ServiceNow already offers a wide variety of pre-built applications. However, ensuring that the data will work with them is a difficult problem and definitely requires some configuration work to do, even after the data tables were successfully transferred to the platform. Still, a Database First Approach is also feasible since the platform allows custom applications to be created on top of the data.

4.2. Project Overview

Since the ServiceNow App Accelerator is a complex application that involves many routines, functions, classes and methods in multiple files, the following tree graph provides an abstract overview of all relevant files and noteworthy directives in their order of appearance.

Legend:

- **c**: class
- **m**: method
- **a**: attribute
- **r**: routine

```
ServiceNowAppAccelerator.rxj
├── c RxApplication
│   ├── m start
│   ├── m handle
│   ├── m setUpDefaultValues
│   ├── m setUpValidators
│   └── m setUpListeners
├── c formValidator
│   └── m changed
├── c manageComboboxChangeListener
│   └── m changed
├── c configurationTabPaneChangeListener
│   └── m changed
├── c mainTabPaneChangeListener
│   └── m changed
├── c ServiceNow
│   ├── a url
│   ├── a username
│   ├── a password
│   └── m init
├── c Database
│   ├── a name
│   ├── a type
│   ├── a host
│   ├── a url
│   ├── a port
│   ├── a username
│   ├── a password
│   ├── a tables
│   ├── m init
│   └── m addTable
└── c DbTable
    ├── a name
    ├── a columns
    ├── m init
    └── m addColumn
```

```

├─ c Column
│   ├── a name
│   ├── a field
│   ├── a type
│   ├── a null
│   ├── a default
│   ├── a extra
│   └─ a key
├─ c JFXTableManager
│   ├── m init
│   ├── m setUpCellValueFactoryAndRootNode
│   ├── m showAndFill
│   ├── m showTablee
│   ├── m clearDatae
│   └─ m fillTablee
├─ c PropertyValueFactory
│   ├── m init
│   └─ m call
├─ c GetChildrenCallback
│   └─ m call

```

```

SAA-controller.rxj
├─ r openMail
├─ r openManual
├─ r goToConfiguration
├─ r testDatabaseConnection
├─ r saveDatabase
├─ r saveServiceNow
├─ r testServiceNow
├─ c TestServiceNowCallback
│   ├── m completed
│   ├── m failed
│   └─ m run
├─ r validateForm
├─ r dbConnect
├─ r showPopup
├─ r setLoadingSymbolToButton
├─ r setCheckGraphicToButton
├─ r saveRexxObjToJson
├─ r getRexxObjFromJson
├─ r promptMasterPassword
├─ r prepareImport
├─ r selectAllCheckboxes
├─ r startImport
├─ c ImportTask
│   ├── m init
│   ├── m getBasicAuthe
│   ├── m connectToDatabase
│   └─ m call

```

```

├─ m unlockNextStep
├─ m calculateTotalApiCalls
├─ m extractTables
├─ m addTable
├─ m log
├─ m logAndProgress
├─ m progress
├─ m getTableSchema
├─ m getColumnSchema
├─ m createServiceNowTable
├─ m uploadData
├─ c DataInsertionCallback
│   └─ m completed
│   └─ m failed

```

4.3. Project Structure

The ServiceNow App Accelerator consists of those two main parts: firstly, an internal ServiceNow application written in ECMAScript 5 as an scoped application using ServiceNow Studio. It features a self-built REST API for creating tables and filling it with so-called "fields", that represent table columns in SQL-like databases. The exact implementation is discussed in listing 4.49 on page 74. Secondly, the ooRexx application which offers a graphical user interface (GUI) with the help of BSF4ooRexx and JavaFX. The application structure is shown in the following graph.

```

ServiceNowAppAccelerator
├─ java
│   └─ Java 8
│       ├── fontawesomefx-commons-8.15.jar
│       ├── fontawesomefx-materialicons-2.2.0-5.jar
│       └─ jfoenix-8.0.1.jar
│   └─ Java 9
│       ├── fontawesomefx-commons-9.1.2.jar
│       ├── fontawesomefx-materialicons-2.2.0-9.1.2.jar
│       └─ jfoenix-9.0.1.jar
├─ commons-codec-1.10.jar
├─ commons-logging-1.1.2.jar
├─ fontawesomefx-fontawesome-4.7.0-5.jar
├─ fontawesomefx-icons525-3.0.0.jar
├─ httpasyncclient-4.0.2.jar
├─ httpclient-4.3.6.jar
├─ httpcore-4.4.6.jar
├─ httpcore-nio-4.4.6.jar
├─ httpmime-4.3.6.jar
├─ jasypt-1.9.2-lite.jar
├─ json-20170516.jar
├─ mysql-connector-java-5.1.43-bin.jar
├─ postgresql-42.1.4.jar
├─ unirest-java-1.4.9.jar
└─ licenses

```

```

├── Apache License 2.txt
├── Unirest MIT License.txt
├── ooRexx
│   ├── credentials.json
│   ├── JFXAlert-controller.rxj
│   ├── json-rgf.cls
│   └── SAA-controller.rxj
├── resources
│   ├── fonts
│   │   ├── palanquin-regular.ttf
│   │   └── palanquindark-regular.ttf
│   ├── installation-guide
│   │   ├── installation-guide.md
│   │   ├── retrieved-update-sets.png
│   │   └── sys_remote_update_set_e10d82a4db020300a3b07d9ebf9619b4.xml
│   ├── JFXAlert.fxml
│   ├── put_FXID_objects_into.my.app.rex
│   ├── SAA.css
│   └── SAA.fxml
├── installation-guide.html
├── ServiceNowAppAccelerator.rxj
└── startup.rxj

```

The root directory holds the installation guide in form of an html file, the "ServiceNowAppAccelerator.rxj" file and the "startup.rxj" file as an entry point to start the application. The filename extension "rxj" stands for "Rexx with Java" and references Rexx files that use BSF4ooRexx to access Java functionality.

The *java* directory holds all necessary Java archives for this BSF4ooRexx application. Its subdirectories "Java 8" and "Java 9" provide Java version specific JAR files to also support Java 9 and higher. The *java* directory includes the following libraries:

- **FontAwesomeFX** comes with several jars and provides an extensible icon font that allows the programmer to use high quality icons without handling raw images. All icons are vector graphics and can be easily resized or coloured. [32]
- **Jasypt** stands for "Java Simplified Encryption" and removes the burden of handling low level methods for iterating over arrays filled with bytes. Using Jasypt, the programmer simply initializes the algorithms, defines a clear text and a password, and receives a base64 cipher text from the library. [33]
- **JFoenix** implements the Google Material Design using JavaFX components. [34]
- **Unirest** is a lightweight library for HTTP requests, which is used for REST API calls in the ServiceNow App Accelerator project. [35]
- **Apache Commons** comes with a lot of different jar files which are just dependencies of the *Jasypt* library.
- **Database connectors** for MySQL and PostgreSQL, so that Java can access those databases.

All of those libraries are Open-Source. Their licenses can be found in the *licenses* directory. Apart from Unirest, which is MIT licensed, all libraries use the Apache License version 2.0 (APL 2.0).

The *ooRexx* directory includes the controller files "SAA-controller.rxj" and "JFXAlert-

controller.rxj". They are used for interaction with the JavaFX view components to handle every event that occurs in the GUI. The file "json-rgf.cls" implements the *JSON* class for ooRexx to read and write json files. This is used for the body of various REST API calls and for the credentials.json file, which is also located in this directory. It is encrypted, so the file only contains a base64 representation of the cipher credentials.

The *resources* directory holds all custom fonts, the Markdown resource files of the installation guide from the root directory, the FXML files, one cascading style sheet and an ooRexx script that puts all used components into a Rexx directory for easy referencing. FXML files contain information on the hierarchical component structure of the GUI. See A.1, A.2 and A.3 for the contents of the FXML and CSS files.

4.4. Application Bootstrap

The *startup.rxj* file in the root directory bootstraps the application. It starts with the license agreement according to the Apache License Version 2.0 and a brief summary of the purpose of this file, its author and version number alongside the date of creation.

```

1  #!/usr/bin/rexx
2  /**
3   * Inserts the Java archives from the "java" directory and Java-Version
4   * dependent subdirectories
5   * to the classpath to make the classes of those libraries accessible and start
6   * the main application
7   *
8   * @author Adrian Baginski, BSc (WU)
9   * @version 1.0, 2018-03-14
10  *
11  *----- Apache Version 2.0 license -----
12  *
13  *      Copyright 2018 Adrian Baginski
14  *
15  *      Licensed under the Apache License, Version 2.0 (the "License");
16  *      you may not use this file except in compliance with the License.
17  *      You may obtain a copy of the License at
18  *
19  *          http://www.apache.org/licenses/LICENSE-2.0
20  *
21  *      Unless required by applicable law or agreed to in writing, software
22  *      distributed under the License is distributed on an "AS IS" BASIS,
23  *      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
24  *      implied.
25  *      See the License for the specific language governing permissions and
26  *      limitations under the License.
27  *-----
28  */
29  signal on syntax
30
31  parse source os . absoluteFilepath
32
33  /* fetch path to this Rexx program */
34  appDirectory = filespec("Location", absoluteFilepath)
35  javaDir      = appDirectory || "java" || .file~separator
36
37  /* determine java version */
38  'java -version 2>&1 | rxqueue' -- fetch output into RexxQueue
39  majorVersion = ""
40  do while queued() > 0          -- make sure that all queued lines get read
41      parse pull line

```



```

37   if majorVersion="" then
38       parse var line "java version "" majorVersion "." minorVersion "." .
39   end
40
41   pathExtension = ""
42   call addJarsToPathExtension
43
44   /* now also add java version dependent jars from the right subdirectory */
45   subDir = (majorVersion >= 9)~?("Java 9", "Java 8")
46   javaDir ||= subDir || .file~separator
47   call addJarsToPathExtension

```

Listing 4.1.: License and Bootstrap

Line 24 turns on signal messages for the entire application, which in case of a syntax error triggers the internal routine "syntax". This way, custom error handling can be achieved. It is a best practice for fetching extended error reports and locating bugs quickly. In this case, additional information about the nature of the error is printed out and the application shuts down in lines 74 to 77. This technique can be seen multiple times throughout all parts of the application.

```

74 syntax:
75     co = condition("object")
76     say ppCondition2(co)
77     exit -1

```

Listing 4.2.: Internal Routine "Syntax"

Line 26 parses the source of the Rexx file which exposes it to some information about the environment it runs in. This call returns three bits of information: [36]

- Operating System
- Command it was invoked with
- Absolute file path including file name

With the help of ooRexx string parsing we save the operating system in the local variable *os*, drop the second information, but save the third one in the variable *absoluteFilepath*.

The built-in function *filespec()* in line 29 extracts the file path from the variable *absoluteFilepath*, and therefore drops the unnecessary file name. Line 30 concatenates the file path of the invoked file with the "java" word using the `\mintinline{ooRexx}{||}` symbol and adds a file separator at the end using a static method of the *FILE* class. The file separator depends on the operating system of the application environment - Windows uses a backslash while Unix based platforms typically use a simple slash.

After that, the code in lines 32 to 39 parses the Java version to later include the right dependencies by sending the command "java -version" to the console and parsing its response.

```

62 addJarsToPathExtension:
63     call SysFileTree javaDir, "jars", "F" -- search for files in java/ and save
        as a .stem in `jars`
64     loop i = 1 to jars.0 -- the 0th entry of a stem holds the
        number of items
65         parse var jars.i . . . . jar -- remove the first 4 bits of
        information (date, time, size, access control)
66         extension = filespec("extension", jar) -- fetch the extension of the
        file
67         if extension = "jar" then do

```

```

68     pathExtension ||= jar~strip
69     pathExtension ||= .file~pathSeparator      -- the pathseparator of the
        File class holds ";" or ":" depending on the OS
70     end
71 end
72 return

```

Listing 4.3.: Internal Routine "addJarsFromDirToClasspath"

The internal routine `addJarsFromDirToClasspath()` is invoked with the `call` instruction in lines 42 and 47. It iterates over all files in the given directory of the variable `javaDir`, searches for all Java archives and concatenates the paths to those files with the contents of the variable `pathExtension`. This routine is called twice, as the `javaDir` variable is adjusted according to the Java version in line 46 right before the second call. This procedure results in first adding all general Java archives to the `pathExtension` variable, followed by the version specific ones either from the subdirectory "Java 8" or "Java 9".

```

48
49 /* get the current value of the CLASSPATH environment variable */
50 classpath = value("CLASSPATH", "ENVIRONMENT")
51
52 say time() "updating environment"
53 newClasspath = classpath || .file~pathSeparator || pathExtension      -- append
        our paths to be looked up last
54 call value "CLASSPATH", newClasspath, "ENVIRONMENT"      -- change the classpath
55
56 /* run the Application again making use of the new classpath */
57 'rexxj "ServiceNowAppAccelerator.rxxj"'      -- this statement will be sent
        directly to the terminal/console, which will invoke "rexxj"
58 /* change PATH back to original and return "true" to exit the application */
59 call value "CLASSPATH", classpath, "ENVIRONMENT"
60 exit

```

Listing 4.4.: Adjusting the Classpath environment variable

Afterwards, the current classpath is extracted from the environment variables. If it does not yet contain the computed `pathExtension`, the application sets a new classpath and starts the ServiceNow App Accelerator application with the new configuration in place. The instruction in line 57 is quite interesting, as it directly sends a command to the console. This command is blocking the further execution of wrapping startup.rxxj program - it will return only if the the second application exits.

This approach ensures, that the ServiceNow App Accelerator can use all dependencies included in the "java" directory. Of course, if the user decides to permanently set each and every jar file to the classpath, he would not even need the startup application. Given the sheer amount of jar files involved, it is not feasible to expect that from all users.

The main application `ServiceNowAppAccelerator.rxxj` prints the current time just before loading all resources in line 38. Lines 40 to 46 import various static Java classes that are used later on. Those are saved in the `.local` directory for easy accessing. As the application grows in lines of code, it is wise to gather those class imports together at this position instead of scattering it all over the document. The instructions in lines 51 and 52 add the controller file to the package list, so that this main application file can also use the respective routines and classes.

```

1 #!/usr/bin/rexx
2 /**

```

```

3  * The main application which loads all Fonts and starts the GUI
4  *
5  *
6  * @author Adrian Baginski, BSc (WU)
7  * @version 1.0, 2017-08-27
8  *----- Apache Version 2.0 license -----
9  *      Copyright 2017 Adrian Baginski
10 *
11 *      Licensed under the Apache License, Version 2.0 (the "License");
12 *      you may not use this file except in compliance with the License.
13 *      You may obtain a copy of the License at
14 *
15 *      http://www.apache.org/licenses/LICENSE-2.0
16 *
17 *      Unless required by applicable law or agreed to in writing, software
18 *      distributed under the License is distributed on an "AS IS" BASIS,
19 *      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
20 *      implied.
21 *      See the License for the specific language governing permissions and
22 *      limitations under the License.
23 *-----
24 */
25 /* in case an error occurs, jump to `Syntax:` label for custom error handling
26 */
27 signal on syntax
28 /* parse operating system and filepath of this script */
29 parse source os . absoluteFilepath
30
31
32 /* add all jars from the java directory to the classpath to make use of the
33    provided classes
34    determine path to this Rexx program */
35 appDirectory = filespec("Location", absoluteFilepath)
36
37 say time() "loading resources"
38
39 /* static class imports */
40 call bsf.import "javafx.beans.value.ChangeListener", "ChangeListener"
41 call bsf.import "javafx.scene.text.Font", "Font"
42 call bsf.import "javafx.util.Callback", "fxCallback"
43 call bsf.import "javafx.collections.FXCollections", "FXCollections"
44 call bsf.import "javafx.beans.property.SimpleStringProperty",
45    "SimpleStringProperty"
46 call bsf.import "javafx.beans.property.SimpleIntegerProperty",
47    "SimpleIntegerProperty"
48 call bsf.import "java.util.concurrent.Executors", "Executors"
49
50 /* adding the controller file as a package to the context allows to use its
51    public routines
52    ::REQUIRES will not work here because the file would be loaded before calling
53    `addJarsFromDirToClasspath`
54    so the static class imports at the top of the file which reference external
55    libraries will cause exceptions */
56 package = .package~new("ooRexx/SAA-controller.rxj")
57 .context~package~addPackage(package)

```

Listing 4.5.: Static class import

Lines 55 to 57 define the *appClzLdr* entry in the *.environment*. This is a working instance of the Java class *URLClassLoader*, which can be utilized to point at certain files and resources inside the project. This is due to some classes requiring this kind of referencing of files, like the *Font* class, which was originally defined in line 41. Both lines 72 and 73 include a font using this class and they require the resource to be a stream rather than a file path or similar.

The instructions in lines 60 to 66 put some additional Java classes to the *.environment* object for having the possibility of also accessing them through the controller files. The *Executor* is a JavaFX concurrency feature for creating and managing new threads. The algorithm used for encrypting and decrypting the credentials is also set here to password based encryption with SHA1 and tripple DES. Later in the execution cycle, the master password is used with this algorithm. Jasypt does not offer any algorithms whatsoever - one has to choose an algorithm based on a list of available algorithms, which highly depends on the installed providers. One of them could be Bouncycastle, Sun or Oracle. [37] Since each computer could have a different list of providers it is best to stick to an algorithm that comes with the Java installation - and *PBEWithSHA1AndDESede* is one the most secure ones in that category. Finally, the JavaFX application is launched in line 77. This is yet again a blocking call, so the next line of code will only start if the GUI is closed, which will result in exiting the ServiceNow App Accelerator and possibly returning to the wrapping application *startup.rnxj* which will also shut down immediately.

```

54 /* prepare the URL class loader for the root directory of this application */
55 jfile = .bsf~new("java.io.File", appDirectory)
56 urlAppDir = jfile~toUri~toUrl
57 .environment~appClzLdr = .bsf~new("java.net.URLClassLoader",
    bsf.createJavaArrayOf("java.net.URL", urlAppDir))
58
59 /* setup environment variables */
60 .environment~jsonLocation = "ooRexx/credentials.json"
61 .environment~FXMLLoader = bsf.import("javafx.fxml.FXMLLoader")
62 .environment~ValidationFacade =
    bsf.import("com.jfoenix.validation.ValidationFacade")
63 .environment~Executor = .Executors~newSingleThreadExecutor -- create a new
    executor for concurrency operations
64 .environment~encryptor =
    .bsf~new("org.jasypt.encryption.pbe.StandardPBEStrngEncryptor")
65 .encryptor~setAlgorithm("PBEWithSHA1AndDESede") -- use triple DES algorithm
    for encryption and decryption operations
66 .environment~masterPassword = .nil
67
68 /* load fonts
69 one could also use @font-face in css to load fonts, but as of 2017-08-24
    there is a bug in JavaFX 8 if the
70 path to the fonts has spaces in it. That is why it is safer to use this
    method
71 see https://stackoverflow.com/questions/33973921/javafx-font-face-css-error-j
    loadstylesheetunprivileged
    */
72 .Font~loadFont(.appClzLdr~getResourceAsStream("resources/fonts/palanquindark-regu_
    gular.ttf"),
    14)
73 .Font~loadFont(.appClzLdr~getResourceAsStream("resources/fonts/palanquin-regula_
    r.ttf"),
    14)
74
75 /* launch the JavaFX Application Thread */
76 .environment~Application =
    bsfCreateRexxProxy(.RxApplication~new,, "javafx.application.Application")

```

```

77 .Application~launch(.Application~getClass,.nil)           -- invoke the "start"
    Method of .RxApplication
78 exit 0

```

Listing 4.6.: JavaFX application start

When starting the application, Java sends a "start" message to the *.RxApplication* class and provides the stage as an argument. JavaFX conceptually uses stages and scenes to build the graphical user interface. A stage is the window that is displayed and the scene represents its content. Therefore, starting new stages would result in opening new windows, and changing the scene would refresh its content.

After receiving the stage, line 97 saves the stage object in an environment variable to make it accessible outside this class. After that, its attribute *title* is set to the name of the application, so that the window has an appropriate title.

```

91 /**
92  * The main GUI class
93  */
94 ::class RxApplication
95 ::method start    -- will be invoked by the "launch" method
96   use arg stage   -- we get the primary stage to use for our UI
97   .environment~stage = stage
98   stage~title = "ServiceNow App Accelerator"
99   .Platform~setImplicitExit(.false)
100  /* if the user closes the stage, call the "handle" method of this class by the
    EventHandler interface */
101  rexxEventHandler = bsfCreateRexxProxy(self,,"javafx.event.EventHandler")
102  stage~setOnCloseRequest(rexxEventHandler)
103  /* load and attach FXML file */
104  sceneFXMLUrl = .appClzLdr~findResource("resources/SAA.fxml")
105  sceneFXML    = .FXMLLoader~load(sceneFXMLUrl)    -- load the fxml document
106  say time() "starting app"
107  /* create a scene from the FXML DOM and assign it to the stage */
108  scene = .bsf~new("javafx.scene.Scene", sceneFXML) -- create a scene for our
    document
109  self~setUpDefaultValues
110  self~setUpListeners
111  self~setUpValidators
112  /* create a new instance of the JFXTableManager Class in scope "local"
    to make it accessible from anywhere in this file */
113  .local~TableManager = .JFXTableManager~new
114

```

Listing 4.7.: The .RxApplication Class

Line 102 activates an event handler for close requests to intervene when the user tries to close the stage manually. The generic Java class *EventHandler* is used many times in JavaFX to handle all kinds of events by invoking the *handle* method in the provided interface class, in that case *self*, which is the *RxApplication* class.

The *handle* method causes the Unirest and Executor background event loops to shut down, before exiting the application. This is compulsory because the main Java application will not be able to stop otherwise. [38]

```

120 /**
121  * Shuts down all event loops and exits the platform
122  */
123 ::method handle
124  /* Unirest starts a background event loop until you manually shutdown all the
    threads by invoking `shutdown` */

```

```

125 .Unirest~shutdown
126 /* the same applies to the ExecutorService */
127 .Executor~shutdown
128 /* shutdown the Application */
129 .Platform~exit
130 say time() "shutting down"

```

Listing 4.8.: CloseRequest Event Handler

Lines 104 to 108 use the static *FXMLLoader* class to load the main fxml file "SAA.fxml" into a variable and create a new instance of the *Scene* class using the previously defined fxml content as parameter. Afterwards, the three methods *setUpDefaultValues*, *setUpListeners* and *setUpValidators* of the *RxApplication* class are called to initialise some functionality of the JavaFX application as shown in listing 4.9.

```

132 /**
133  * Points each combo box to the first value as this is not possible in FXML
134  */
135 ::method setUpDefaultValues private
136   comboBoxes = .my.app~SAA.fxml~databaseType, .my.app~SAA.fxml~manageCombobox
137   loop comboBox over comboBoxes
138     comboBox~getSelectionModel~select(0)
139   end
140
141 /**
142  * Creates change listeners on the "focused" property of some predefined
143  * mandatory textfields
144  */
145 ::method setUpValidators private
146   IDs = "databaseHost", "databaseDatabase", "databaseUsername", -
147         "servicenowURL", "servicenowUsername", "servicenowPassword", -
148         "importDatabase", "importServiceNow"
149   loop ID over IDs
150     rexxProxy = bsfCreateRexxProxy(.formValidator~new,, .ChangeListener)
151     control = .my.app~SAA.fxml[ID~upper]
152     control~focusedProperty~addListener( rexxProxy)
153   end
154
155 /**
156  * Sets change listener to various controls of the FXML file
157  */
158 ::method setUpListeners private
159   /* add a change listener to the "manage" combobox */
160   rexxProxy =
161     bsfCreateRexxProxy(.manageComboboxChangeListener~new,, .ChangeListener)
162   .my.app~SAA.fxml~manageCombobox~valueProperty~addListener( rexxProxy)
163   /* add a change listener to the TabPane in the "Configuration" Tab
164   so we can update the data in the "Manage" Sub-Tab every time the user
165   visits this Tab */
166   rexxProxy = bsfCreateRexxProxy(.configurationTabPaneChangeListener~new,, .ChangeListener)
167   configurationTabPane = .my.app~SAA.fxml~configurationTabPane
168   configurationTabPane~getSelectionModel~selectedItemProperty~addListener( rexxProxy)
169   /* add a change listener to the main TabPane
170   so we can update both ComboBoxes in the "Import Data" Tab each time the
171   user visits that Tab */
172   rexxProxy =
173     bsfCreateRexxProxy(.mainTabPaneChangeListener~new,, .ChangeListener)
174   mainTabPane = .my.app~SAA.fxml~mainTabPane

```

```
170 | mainTabPane~getSelectionModel~selectedItemProperty~addListener(rexxProxy)
```

Listing 4.9.: Completing the Stage initialisation

The method *setUpDefaultValues* iterates over an predetermined array of combo boxes and selects each first element. This is a valuable user experience addition to the application, because the combo boxes would have an empty value otherwise. The method *setUpValidators* is very similar, as it attaches a change listener to the *focused* property of some selected controls. In case of this event, the *changed* method of the given interface class *formValidator* is invoked with the following three arguments:

- **observable:** An *ObservableValue* object that holds the respective property
- **oldValue:** The value of the property before it was changed
- **newValue:** The value after the change

The *formValidator* simply invokes the *validate* method of the input fields or uses a *ValidationFacade* to do so for combo boxes since they need special handling. [39] The validators are located throughout the FXML file and simply check if a value is set, otherwise the bottom border is coloured red.

```
173 /**
174  * An implementation of the ChangeListener interface.
175  */
176 ::class formValidator private
177 /**
178  * Validates the control when it loses focus
179  *
180  * @param observable - <code>javafx.beans.value.ObservableValue</code>
181  * @param oldValue   - The value of the control before the change
182  * @param newValue   - The current value after it has changed
183  */
184 ::method changed
185   use arg observable, oldValue, newValue
186   if \newValue then do
187     control = observable~getBean
188     if control~getClass~getSimpleName = "JFXComboBox" then
189       .ValidationFacade~validate(control)      -- JFXComboBox
190     else
191       control~validate                          -- JFXTextField
192   end
```

Listing 4.10.: The formValidator class

The *setUpListeners* method in line 157 attaches a Rexx proxy class called *manageComboboxChangeListener* as a change listener handler to the *manageCombobox*. This class simply invokes the *showAndFill* method of the *TableManager* class with the new value of the combo box, which is described in listing 4.34.

```
196 /**
197  * A change listener class for the control `manageCombobox` in the `Manage` Tab
198  */
199 ::class manageComboboxChangeListener private
200 /**
201  * This method gets invoked when the value of the `manageCombobox` has changed
202  * It clears the values of the the right table depending on the selection and
203   fills it with fresh data
204  *
205  * @param observable - <code>javafx.beans.value.ObservableValue</code>
```

```

205 * @param oldValue - The value of the control before the change
206 * @param newValue - The current value after it has changed
207 */
208 ::method changed
209   use arg observable, oldValue, newValue
210   .TableManager~showAndFill(newValue)

```

Listing 4.11.: The manageComboboxChangeListener class

Then, it creates two independent Rexx proxies *configurationTabPaneChangeListener* and *mainTabPaneChangeListener* to attach them to the respective tab pane as change listener handlers. The first mentioned class reloads the displayed table when the user selects the "Manage" tab by invoking the *showAndFill* method in line 229, whose implementation is discussed in listing 4.33.

The *mainTabPaneChangeListener* is a complex class that observes the main menu on the left side. If the user clicks on "Import data", implemented in line 247, the script clears all previous Database and ServiceNow credential combo box values in line 253 and refills them with fresh data retrieved from the encrypted json file in the next lines. The implementation of the *getRexxObjFromJson()* routine is located in the controller file and described in listing 4.22. Line 260 assembles the text for one combo box item using information about the database type, host, and its name. Moreover the ServiceNow combo box is supposed to show only the URL. After adding all items to an observable array list, they are attached to the respective combo boxes in lines 263 and 270. This results in refreshing the data, so that the user is always presented current data.

```

214 /**
215  * A change listener class for the JFXTabPane `configurationTabPane`
216  */
217 ::class configurationTabPaneChangeListener private
218 /**
219  * This class will reload the displayed Table when the user selects the "Manage"
220  * Tab
221  *
222  * @param observable - <code>javafx.beans.value.ObservableValue</code>
223  * @param oldValue - The Tab before the change
224  * @param newValue - The current Tab after it has changed
225  */
226 ::method changed
227   use arg observable, oldValue, newValue
228   if newValue~getText = "Manage" then do
229     selectedComboboxItem =
230       .my.app~SAA.fx.xml~manageCombobox~getSelectionModel~getSelectedItem
231     .TableManager~showAndFill(selectedComboboxItem)
232   end
233
234 /**
235  * A change listener class for the main Tab Pane
236  */
237 ::class mainTabPaneChangeListener private
238 /**
239  * Refresh the items of both ComboBoxes on the "Import data" Tab when the user
240  * visits it
241  *
242  * @param observable - <code>javafx.beans.value.ObservableValue</code>
243  * @param oldValue - The Tab before the change
244  * @param newValue - The current Tab after it has changed

```



```

244 */
245 ::method changed
246 use arg observable, oldValue, newValue
247 if newValue~getText~word(1) = "Import" then do
248     /* get a reference to the combo boxes */
249     databaseCombobox = .my.app~SAA.fxml~importDatabase
250     servicenowCombobox = .my.app~SAA.fxml~importServicenow
251     /* clear the data */
252     loop combobox over databaseCombobox, servicenowCombobox
253         combobox~getItems~clear
254     end
255     jsonObj = getRexxObjFromJson()
256     /* set items for databaseCombobox */
257     items = .FXCollections~observableArrayList
258     if jsonObj~hasEntry("Database") then
259         loop data over jsonObj~Database
260             text = data~type || ":" || data~host || "/" || data~dbName
261             items~add(text)
262         end
263     databaseCombobox~getItems~addAll(items)
264     /* set items for databaseCombobox */
265     items = .FXCollections~observableArrayList
266     if jsonObj~hasEntry("ServiceNow") then
267         loop data over jsonObj~ServiceNow
268             items~add(data~url)
269         end
270     servicenowCombobox~getItems~addAll(items)
271 end
272 else if oldValue~getText = "Import data" & .my.app~hasEntry("importTask") then
273     do
274         /* if the user switches from last tab to another tab and has started to
275         prepare the import task, abort it */
276         if .environment~hasEntry("jTask") then
277             .jTask~cancel(.true) -- interrupt the Thread if necessary
278         /* show the first screen for the user to select a database and servicenow
279         instance */
280         .my.app~SAA.fxml~importTabPane~getSelectionModel~selectFirst
281     end

```

Listing 4.12.: Tab pane change listener implementations

After all the previous steps have been completed, the application finally attaches the scene to the stage at line 116 and invokes the *show* method, causing the graphical user interface to display, thus concluding the bootstrap phase of the ServiceNow App Accelerator.

```

115 /* show the GUI */
116 stage~~setScene(scene) -
117     ~~show
118 say time() "app ready"

```

Listing 4.13.: Completing the Stage Initialisation

4.5. Configuration

The graphical design of this application is depicted in figure 4.1 running on Windows 10. It consists of a menu on the left and the main content area to the right. The overall design relies heavily on Google Material Design implemented with JFoenix' custom controls such as most used tab panes, input text fields, combo boxes, buttons and the progress bar at the last step of the import data process. At application startup, the user receives an introductory message that includes the email address of the author to file bugs and a link to the installation guide located in the root directory. He is then prompted to install an update set inside his ServiceNow instance for the obligatory REST API support. The main code of the update set can be found in listing 4.49, whereas the content of the installation guide is depicted in figure 4.2.

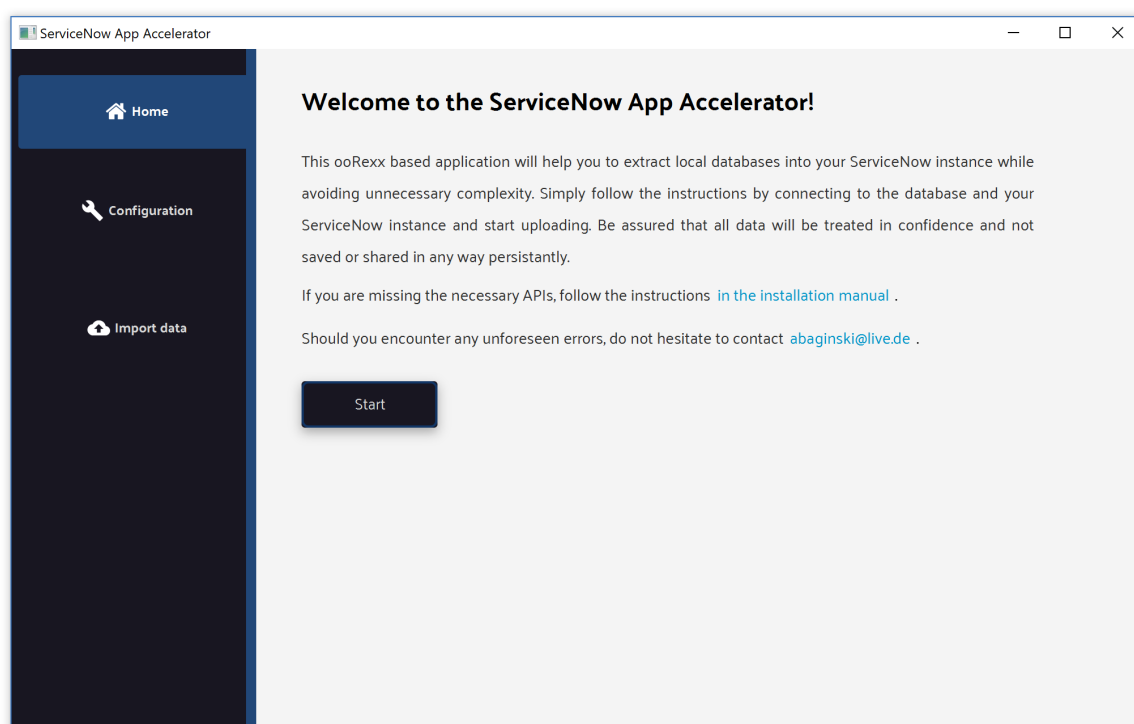


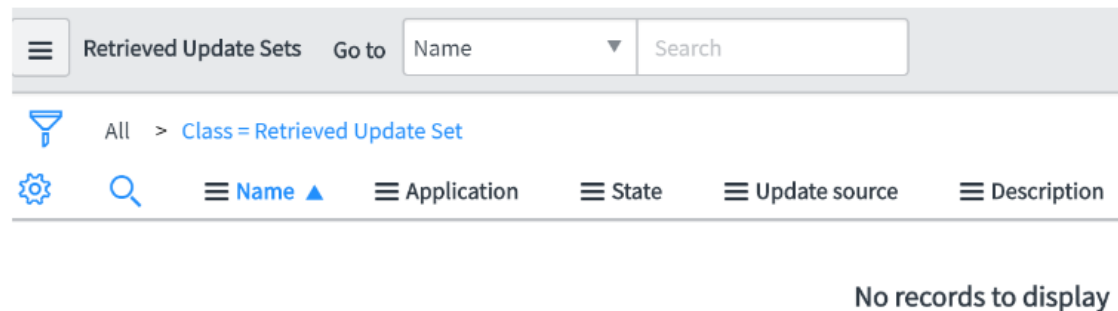
Figure 4.1.: Application Start Screen

ServiceNow App Accelerator API installation guide

This document guides you through the API installation process in ServiceNow. First off, download the following file to your preferred location:

- [Download Update Set](#)

Now go to your ServiceNow instance and navigate to `Update Sets > Retrieved Update Sets`. On that page, click on `Import Update Set from XML` as shown in the screenshot below.



Related Links

[Import Update Set from XML](#)

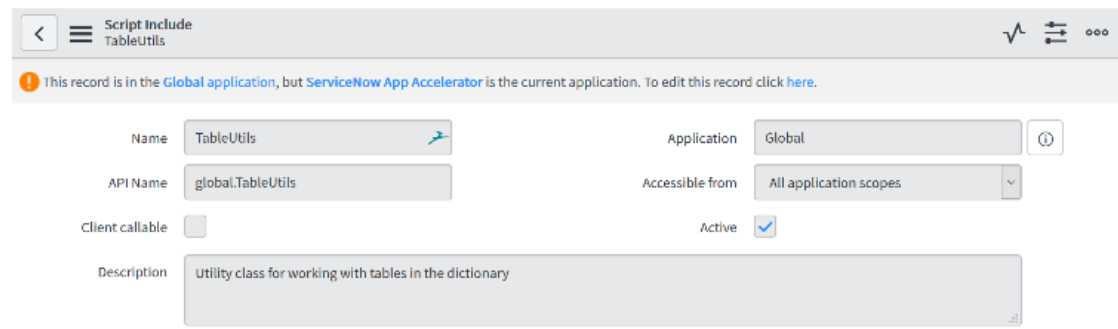
There you can upload the [Update Set](#) and confirm by clicking on the `OK` button. After that you will get prompted to commit the Update Set to finally import it.

There are three private Script Includes, that need to be allowed for the ServiceNow App Accelerator application in order to work properly. Those are

- TableUtils
- TableDescriptor
- FieldDescriptor

Simply switch to `System Definition > Script Includes`, edit either one of these and change the `Accessible from` attribute to "All application scopes" as shown in the picture below.

generated by haroopad



Congratulations, you are all set up!

Figure 4.2.: Installation Manual

By pressing the "Start" button, the routine *goToConfiguration* from the controller file in line 43 gets invoked, which results in selecting the next item of the main tab pane on the left. Listing 4.14 shows some static class imports used in the controller file in lines 9 to 17, as well as the implementation of all routines used in the "Home" tab.

The *openMail* routine fetches the supplied *slotDir* as argument. This information is always attached by BSF4ooRexx as a function call's last argument and can be used to access the ScriptContext and its Bindings [40]. Doing so, the routine extracts the value of the click event target and tries to open an URI using the *Application* class, so that it is up to the operating system to decide which program to use for that purpose. In the case of an E-Mail address as URI starting with the string "mailto:", it will open a pre-installed User Mail Agent. The *openManual* routine applies the exact same technique to open the file installation-guide.html in the Browser.

```

1  /**
2   * The FXML Controller File communicates directly with the FXML controls from
3   * "SAA.fxml".
4   * All Button or HyperLink clicks in FXML controls result in invoking a public
5   * Routine located in this file.
6   *
7   * @author Adrian Baginski, BSc (WU)
8   * @version 1.0, 2017-08-27
9   */
10  /* Java static class imports */
11  call bsf.import "java.sql.DriverManager", "DriverManager"
12  call bsf.import "javafx.scene.control.Alert$AlertType", "AlertType"      --
13  call bsf.import "de.jensd.fx.glyphs.materialicons.MaterialIcon", "MaterialIcon"
14  call bsf.import "com.mashape.unirest.http.async.Callback", "Callback"
15  call bsf.import "com.mashape.unirest.http.Unirest", "Unirest"
16  call bsf.import "javafx.application.Platform", "Platform"
17  call bsf.import "java.lang.Runnable", "Runnable"
18
19  /**
20   * Fetches the E-Mail address from slotDir and writes a new Mail using the
21   * default Mail client
22   *
23   * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
24   */
25  ::routine openMail public
26  use arg slotDir
27  scriptContext = slotDir~scriptContext
28  event = scriptContext~getAttribute("event")
29  hyperlink = event~target
30  emailAddress = hyperlink~text
31  .Application~getHostServices~showDocument("mailto:" || emailAddress)
32
33  /**
34   * Opens the file `installation-guide.md` in the root directory
35   */
36  ::routine openManual public
37  .Application~getHostServices~showDocument("installation-guide.html")
38
39
40  /**
41   * Selects the next Tab ('configuration')

```

```

42 */
43 ::routine goToConfiguration public
44   TabPane = .my.app~SAA.fxml~mainTabPane
45   SelectionModel = TabPane~getSelectionModel
46   SelectionModel~selectNext

```

Listing 4.14.: Static Imports and "Home" Routines from SAA-controller.rxj

The "Configuration" Tab consists of another tab pane to choose between "Database", "ServiceNow" and "Manage". By default, "Database" is selected and offers a form to provide database related information as depicted in figure 4.3. It has the following input fields:

- **Database type:** an editable combo box with two predefined values "mysql" and "postgresql". Adding new database types is possible, albeit they have to follow a strict naming convention, since the ServiceNow App Accelerator will use this string as identifier for connecting to the database. [41]
- **Host:** The IP address of the database host. This information can also be a domain name, as long as the DNS can resolve it.
- **Port:** The optional Port of the database. If it is left out, Java will try to contact the database on the usual ports - 3306 for MySQL for example.
- **Database:** The unique name of the database.
- **Username:** The name of the database user.
- **Password:** The password of the given user for the database. The input text is masked by big black bullets.

The fields "Database type", "Host", "Database" and "Username" are mandatory and protected by a validator. If the value is empty, the control's black bottom border will turn red and prevent the form from submitting.

Figure 4.3.: Configuration - Database

After the user inserts all values properly, he can either save without verification or test the connection first as described in listing 4.15.

```

49 /**
50  * Tries to connect to the Database and shows the result as a Popup or Button
   image, depending on the result
51  *
52  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
53  */
54 ::routine testDatabaseConnection public
55  use arg slotDir
56  /* validate the form fields first */
57  IDs = "databaseHost", "databaseDatabase", "databaseUsername",
        "databasePassword"
58  if \validateForm(IDs) then return -- leave the routine if .false returned
59  scriptContext = slotDir~scriptContext
60  /*@get(databaseType databaseHost databasePort databaseUsername
   databasePassword databaseDatabase testDatabaseConnectionButton)*/
61  call setLoadingSymbolToButton testDatabaseConnectionButton
62  call SysSleep .5 -- wait 500ms for Java to update its GUI
63  /* now connect to the Database, since the arguments from the scriptContext
   reference Java objects, we need
64  to extract the values of them before passing to the routine "dbConnect" */
65  selectedType = databaseType~getSelectionModel~getSelectedItem
66  signal on syntax
67  call dbConnect selectedType, databaseHost~getText, databaseUsername~getText,
        databasePassword~getText, -
68        databaseDatabase~getText, databasePort~getText
69  if result~isA(.bsf) then do -- a Connection returned, i.e. dbConnect was
        successful
70    call setCheckGraphicToButton testDatabaseConnectionButton
71  end
72  return
73
74  syntax:
75    co = condition('o')
76    message = co['ADDITIONAL'][1]
77    call showPopup message
78    testDatabaseConnectionButton~graphic = .nil -- remove the loading
        spinner, test connection failed

```

Listing 4.15.: testDatabaseConnection Routine

The routine *testDatabaseConnection* initially calls the *validateForm* function on all the previously mentioned mandatory form fields, which simply loops over all items and either calls the *~validate* method or uses the *ValidationFacade* class for combo boxes. Additionally, it focuses on the first faulty control to for user experience reasons. This function will return *.false* if at least one control did not match the validation pattern defined in the FXML file. The ServiceNow App Accelerator uses solely *RequiredValidators* to check for value existence.

After all fields were validated, the *testDatabaseConnection* routine proceeds with extracting the ScriptContext entry of the slotDir argument. It uses a decorator provided by BSF4ooRexx in line 60 to fetch the controls from the fxml file using their respective fx:id attribute values and saves them in local variables.

```

235 /**
236  * Calls the "validate" method of every JFXTextField supplied in the argument
237  *
238  * @param IDs - an Array of fx:id attributes of JFXTextField controls

```

```

239 * @return <code>.false</code> if the validator finds an error, otherwise
      <code>.true</code>
240 */
241 ::routine validateForm private
242   parse upper arg IDs
243   loop ID over IDs
244     control = .my.app~SAA.fxml[ID]
245     if control~getClass~getSimpleName = "JFXComboBox" then
246       result = .ValidationFacade~validate(control)
247     else
248       result = control~validate          -- JFXTextField
249     if \result then do      -- control has an error
250       /* we need to ensure that all controls get validated before exiting this
          function */
251       if firstFaultyControl~isA(.string) then
252         firstFaultyControl = control
253     end
254   end
255   /* focus on firstFaultyControl if it is a JavaFX control */
256   if firstFaultyControl~isA(.bsf) then do
257     firstFaultyControl~requestFocus
258     return .false
259   end
260   return .true

```

Listing 4.16.: validateForm Routine

Line 61 calls the routine *setLoadingSymbolToButton* with the button that was activated as argument. This routine simply sets a progress indicator control as graphic with a fixed size of 20 by 20 pixels and a value of -1.0, which is an indeterminate value and is thus displayed as a spinning circle. This is also a user experience measure to indicate that the script is running and he is expected to receive a result soon.

```

309 /**
310  * Sets an indeterminate ProgressIndicator as graphic to the specified Button
311  *
312  * @param button - The reference to a JavaFX Button
313  */
314 ::routine setLoadingSymbolToButton
315   use arg button
316   loadingSymbol = .bsf~new("javafx.scene.control.ProgressIndicator", -1.0)
317   -- indeterminate progress
317   loadingSymbol~prefHeight = 20.0          -- resize the loading symbol
318   loadingSymbol~prefWidth  = 20.0
319   button~graphic = loadingSymbol          -- set the progress indicator as
      image for the Button

```

Listing 4.17.: setLoadingSymbolToButton Routine

After this function call, the wrapping routine *testDatabaseConnection* calls a built-in function called "SysSleep" in line 62 to suspend the progress of the application for a specific amount of time, in this case half a second. This is done for Java to update the GUI before proceeding with the application flow.

Line 67 calls the routine *dbConnect* with a set of values and expects to return a *Connection* object. The mentioned routine assembles a connection string using the provided information in the arguments. It creates an instance of the Driver class for the database type, registers it using the static class *.DriverManager* and tries to get a connection with the credentials. Line 66 defines *signal on syntax*, so that if anything causes an error, the label *syntax* in

line 74 gets activated and fetches an error message based on the condition object in line 75. The syntax subroutine further calls the *showPopup* routine with the stated error message, and clears the loading symbol off the *testDatabaseConnectionButton*.

```

262 /**
263  * Connects via JDBC to a Database
264  *
265  * @param type - Database type, e.g. mysql or postgresql
266  * @param host - Host or IP address of the Database
267  * @param user - Username of the Database
268  * @param password - Password for the given User
269  * @param dbName - The name of the Database
270  * @param [port] - The Port number for this connection
271  * @return <code>java.sql.Connection</code>
272  */
273 ::routine dbConnect private
274   use arg type, host, user, password, dbName, port
275   /* Assemble the Connection String from the given information */
276   url = "jdbc:" || type || "://" || host
277   if port <> "PORT" & port <> "" then -- if the port was not provided as
       argument, ooRexx will assign uppercase PORT to variable port
278     url ||= ":" || port
279   url ||= "/" || dbName
280   /* Get the JDBC-Driver */
281   driver = .bsf~new("com." || type || ".jdbc.Driver")
282   .DriverManager~registerDriver(driver)
283   /* Build the Connection*/
284   connection = .DriverManager~getConnection(url, user, password)
285   return connection

```

Listing 4.18.: dbConnect Routine

The *showPopup* routine generates a new custom Alert with an expendable text area and shows it, therefore blocking the application flow until it has been closed. This provides a generic way to show all kinds of error messages that can be utilized throughout the entire application. The *~wrapText* attribute of the *Alert* class in line 301 causes the text area to not overflow its bounds, so that the user can always read the entire error message.

```

288 /**
289  * Displays a JavaFX Popup with the error message as expandable content
290  *
291  * @param message - The printed error message
292  */
293 ::routine showPopup
294   use strict arg message
295   alert = .bsf~new("javafx.scene.control.Alert", .AlertType~ERROR)
296   alert~setTitle("An Error occurred")
297   alert~setHeaderText("An Error occurred");
298   alert~setContentText("Ooops, something went wrong! Have a look at the error
       message below:" .endOfLine)
299   expContent = .bsf~new("javafx.scene.layout.GridPane")
300   textArea = .bsf~new("javafx.scene.control.TextArea", message)
301   textArea~wrapText = .true
302   expContent~add(textArea, 0, 1)
303   /* Set expendable Exception into the dialog pane */
304   alert~getDialogPane~~setExpendableContent(expContent) -
305     ~~setExpanded(.true)
306   alert~showAndWait

```

Listing 4.19.: showPopup Routine

Finally, if there was no error in the *dbConnect* execution, the *testDatabaseConnection* routine invokes *setCheckGraphicToButton* with the "Test connection" button as argument, which simply sets a material icon tick mark as graphic, overriding the previously used loading symbol.

```

323 * Sets a Tick mark as graphic to the specified Button
324 *
325 * @param button - The reference to a JavaFX Button
326 */
327 ::routine setCheckGraphicToButton
328   use arg button
329   /* create a tick mark using FontAwesome */
330   checkSymbol = .bsf~new("de.jensd.fx.glyphs.materialicons.MaterialIconView",
331   .MaterialIcon~CHECK)
331   button~graphic = checkSymbol

```

Listing 4.20.: setCheckGraphicToButton Routine

The user is also able to save his database connection permanently by invoking the *saveDatabase* routine through the "Save" button. Lines 89 to 93 from listing 4.21 validate the user input first to prevent saving empty values. Then the app creates a Rexx *.stringTable* object to save all information from the form and calls the *saveRexxObjToJson* routine with the just created Rexx object and the "Database" string identifier. If that works, the *saveDatabaseButton* gets a tick mark too. Figure 4.5 shows how this form would look like after testing and saving the database connection.

```

80 /**
81  * Saves the provided Database information in an ooRexx StringTable and
82  * invokes `saveRexxObjToJson` to save the data persistantly
83  * It is recommended to test the connection first, as this routine merely saves
84  * the data!
85  *
86  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
87  */
88 ::routine saveDatabase public
89   use arg slotDir
90   /* validate the form fields first */
91   IDs = "databaseHost", "databaseDatabase", "databaseUsername",
92   "databasePassword"
93   if \validateForm(IDs) then return -- leave the routine if .false returned
94   scriptContext = slotDir~scriptContext
95   /*@get(databaseType databaseHost databasePort databaseUsername
96   databasePassword databaseDatabase saveDatabaseButton)*/
97   /* create a Rexx Object to hold all information */
98   rxObj = .stringTable~new
99   rxObj~type = databaseType~getSelectionModel~selectedItem
100   rxObj~host = databaseHost~getText
101   rxObj~port = databasePort~getText
102   rxObj~user = databaseUsername~getText
103   rxObj~pass = databasePassword~getText
104   rxObj~dbName = databaseDatabase~getText
105   success = saveRexxObjToJson(rxObj, "Database")
106   if success then
107     call setCheckGraphicToButton saveDatabaseButton

```

Listing 4.21.: saveDatabase Routine

Saving the database connection conceptually requires to open the json file with the credentials, decrypt it, extract only the database information, add the new credentials and then

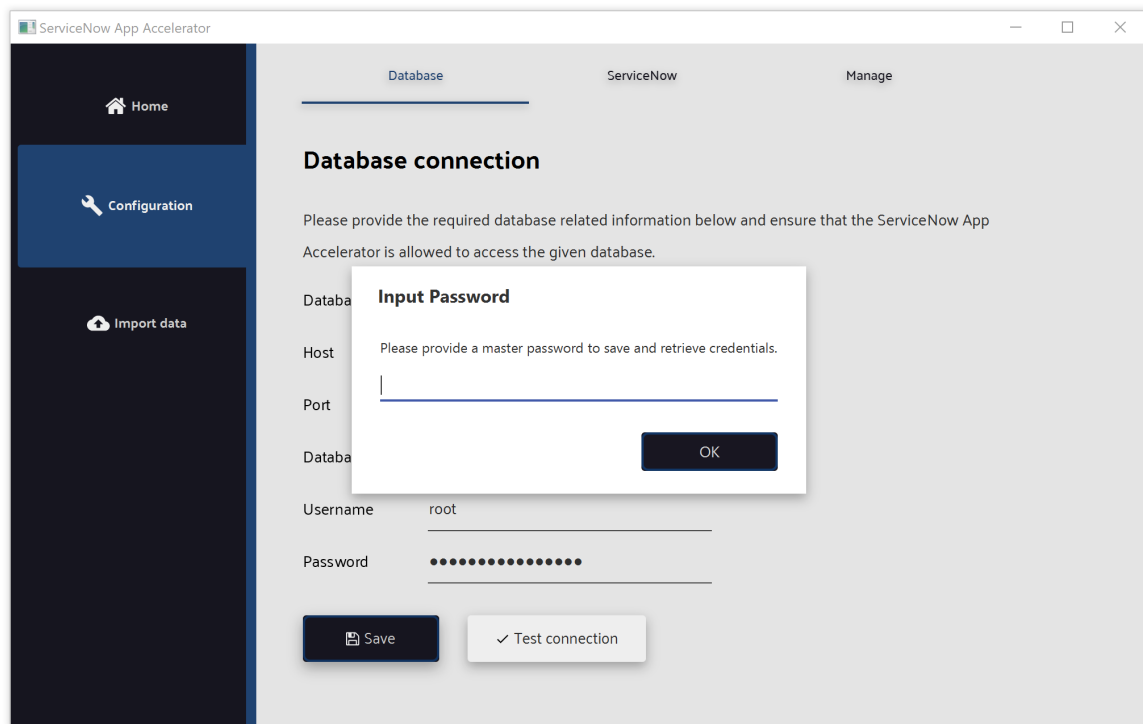


Figure 4.4.: Master Password Prompt

encrypt and rewrite it back to its initial location. The *getRexxObjFromJson* routine creates a Rexx object in lines 376 to 378, that has a similar structure to that of a JSON string. A Rexx *.queue* is a collection class that offers the "push" method where new items are stored at the beginning of the queue. This is exactly what is needed in this case, because the latest credentials are the most important ones, thus they should be displayed first. After making sure that a master password is set, the application opens the file "credentials.json" in line 384, reads its content in line 387 and tries to decrypt it using Jasypt in line 390. If anything goes wrong, the syntax subroutine gets invoked, closing the stream if necessary and returning the empty *jsonObj*. In case of successful decryption, the Database and ServiceNow information of the decrypted credentials file is appended to the *jsonObj* before returning. Note that Rexx executes all instructions after the *syntax:* label because there is no *exit* or *return* instruction before that.

The routine *saveRexxObjToJson* appends the *rxObj* from the argument to the *jsonObj* from the *getRexxObjFromJson* function call. Then it encrypts the Rexx object holding the old and new credentials and writes it back line by line to the "credentials.json" file in lines 361 to 363.

```

334 /**
335  * Reads the contents of .jsonLocation defined in the main app and appends the
336  * provided Rexx Object
337  *
338  * @param rxObj - A Rexx Object which will be converted to Json
339  * @param type - Can be either "Database" or "ServiceNow"
340  */
341 ::routine saveRexxObjToJson
342   use strict arg rxObj, type
343   jsonObj = getRexxObjFromJson()
344   /* we cannot save without a password, so leave the routine if there is no
345      password set */

```

```

344 if .masterPassword = .nil then
345     return .false
346 /* prepend our argument `rxObj` based on the type to the `jsonObj` */
347 select
348     when type = "Database" then
349         jsonObj~Database~push(rxObj)
350     when type = "ServiceNow" then
351         jsonObj~ServiceNow~push(rxObj)
352     otherwise
353         raise syntax 40.900 array("Bad Type. Use either `Database` or
354             `ServiceNow`!")
355 end
356 /* convert `jsonObj` back to a string and write it to the jsonStream */
357 jsonEngine = .json~new
358 jsonString = jsonEngine~toJson(jsonObj)
359 encryptedJsonString = .encryptor~encrypt(jsonString)
360 jsonStream = .stream~new(.jsonLocation)
361 jsonStream~open("write replace") -- open the stream with write access and
362     replace its content when writing
363 loop line over encryptedJsonString
364     jsonStream~lineOut(line)
365 end
366 jsonStream~close -- release the lock on this file
367 return .true
368
369 /**
370  * Reads the JSON String from the hard drive and replaces the used Array with an
371  * Queue
372  *
373  * @return jsonObj - A Rexx Object which holds all information from the Json
374  * file located in .jsonLocation
375  */
376 ::routine getRexxObjFromJson public
377 /* create a new jsonObj with queues instead of the generated arrays and import
378 the existing data from tmpObj
379 queues offer the "push" method, with which new items land on top of the
380 collection */
381 jsonObj = .stringTable~new
382 jsonObj~Database = .queue~new
383 jsonObj~ServiceNow = .queue~new
384 /* make sure that the master password is set or return empty jsonObj */
385 passwordIsSet = promptMasterPassword()
386 if \passwordIsSet then
387     call syntax
388 /* load a fresh copy of the json file as a Rexx string */
389 jsonStream = .stream~new(.jsonLocation)
390 jsonStream~open("read")
391 jsonLength = jsonStream~chars
392 encryptedData = jsonStream~charIn(1, jsonLength)
393 /* try decrypting the data, the app throws a Signal if the password is wrong
394 */
395 signal on syntax -- go to syntax: in case of exception
396 jsonString = .encryptor~decrypt(encryptedData)
397 /* convert the Rexx string with json syntax to a Rexx object */
398 jsonEngine = .json~new
399 tmpObj = jsonEngine~fromJson(jsonString)
400 jsonObj~Database~appendAll(tmpObj~Database)
401 jsonObj~ServiceNow~appendAll(tmpObj~ServiceNow)
402 syntax:
403     if jsonStream~isA(.stream) then

```

```

398     jsonStream~close
399     return jsonObj

```

Listing 4.22.: saveRexxObjToJson and getRexxObjFromJson Routines

The *promptMasterPassword* routine displays a Google Material Design styled custom alert box as depicted in figure 4.4 on page 42. Unfortunately there is no out-of-the-box component to use here, so all controls have to be assembled by the programmer individually to create such an alert box. To save some lines of code, this application uses the *.FXMLLoader* class to import the FXML file shown in A.3. The function *bsf.createJavaArrayOf()* in line 412 creates a Java array of type *Node* and length 1 with the fxml content. The application then adds an OK button to the Alert window and *~setOverlayClose* is used to turn overlay close off, so that the user is not allowed to leave this alert without entering a password when clicking outside the alert box area. Since Jasypt does not allow empty passwords, lines 421 and 422 test if the result of the alert box is a string or empty. After that, the master password is saved in the environment object and fed to Jasypt's password based key derivation function using the *~setPassword()* method.

This popup will only show once given that the user inputs a password properly. If it matches the previously used password for encrypting credentials, the *getRexxObjFromJson* routine will succeed in retrieving them from the encrypted "credentials.json" file and be able to restore them.

```

401 /**
402  * Displays a text input dialog if the master password is empty
403  *
404  * @return <code>.true</code> if password is set
405  */
406 ::routine promptMasterPassword
407     if .masterPassword = .nil then do
408         JFXAlert = .bsf~new("com.jfoenix.controls.JFXAlert", .stage)
409         .environment~JFXAlert = JFXAlert          -- make it accessible for
JFXAlert-controller.rxj
410         alertFXMLUrl = .appClzLdr~findResource("resources/JFXAlert.fxml")
411         alertFXML     = .FXMLLoader~load(alertFXMLUrl)  -- load the fxml document
412         JFXAlert~setContent(bsf.createJavaArrayOf("javafx.scene.Node", alertFXML))
-- setContent is expecting a java array of type Node
413         okButtonType = .bsf~new("javafx.scene.control.ButtonType", "OK",
bsf.import("javafx.scene.control.ButtonBar$ButtonData")~OK_DONE)
414         JFXAlert~getDialogPane~getButtonTypes~add(okButtonType)
415         /* usually, if one clicks anywhere in a JFXAlert, it closes.
416         Disable that because we want to encourage the user to use a password */
417         JFXAlert~setOverlayClose(.false)
418         /* show the alert and wait for its answer */
419         Optional = JFXAlert~showAndWait
420         /* if the user clicks on ESC, Optional~get holds the invisible "Close"
421         Button, return .false in that case */
422         if \Optional~get~isA(.string) then
423             return .false
424         /* save the master password in the environment */
425         .environment~masterPassword = Optional~get
426         /* feed the password based key derivation function */
427         .encryptor~setPassword(.masterPassword)
428     end
429     return .true          -- password was set before

```

Listing 4.23.: promptMasterPassword Routine

ServiceNow App Accelerator

Home Configuration Import data

Database ServiceNow Manage

Database connection

Please provide the required database related information below and ensure that the ServiceNow App Accelerator is allowed to access the given database.

Database type:

Host:

Port:

Database:

Username:

Password:

Figure 4.5.: Database Connection saved

Once the user has tested and saved his database connection details, he can switch to the second tab "ServiceNow". There he gets presented with a short text and just the those three input fields, as described below and shown in figure 4.6 on page 46:

- ServiceNow instance URL
- Username
- Password

Since ServiceNow's REST APIs are secured with basic authentication, username and password alongside the URL of the ServiceNow instance are mandatory to provide. In this view the user also gets presented with two buttons to test and save his credentials. Testing invokes *testServiceNow*, which works very similar to the *testDatabaseConnection* routine, just with less controls to deal with.

After validating all form controls, a loading symbol is attached to the button in line 143. Then, the API URL is extended with the string `"/api/x_146620_serviceno/create_table/%7BtestApiExistence%7D"`. This calls the custom REST API for creating a table, which was created for to this project and can be imported with an XML Update Set following the instructions in the manual. The exact ServiceNow-side JavaScript implementation of this API is discussed in 4.49 on page 74. Basically, it fetches the table name of the URL and validates it against the string *testApiExistence* and eventually returns just a message that the API exists and works properly. Additionally, ServiceNow obviously inspects the basic authentication details and returns an error if these are wrong.

Line 150 creates a new Rexx proxy class `.TestServiceNowCallback` that implements the Java *Callback* interface. In line 152 to 154 Unirest is used to create an asynchronous PUT request to the previously assembled URL of the API authenticated with the given username and password. In case an error occurs, lines 158 and 159 remove the loading symbol of the

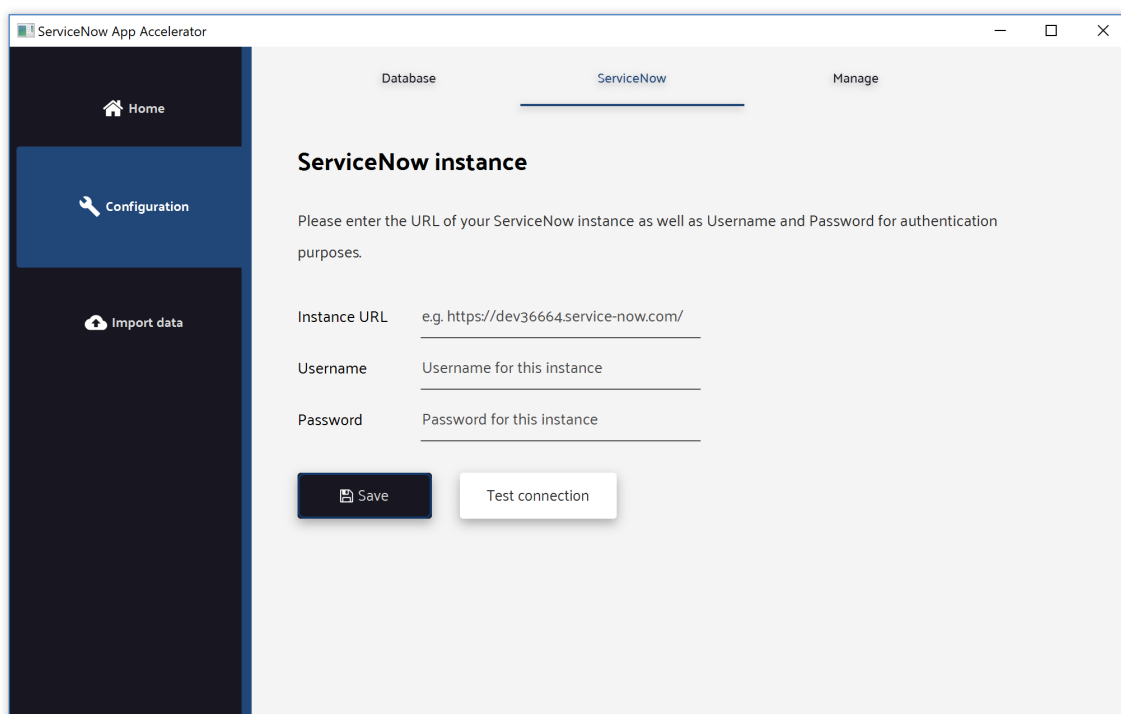


Figure 4.6.: Configuration - ServiceNow

button and raise the condition again in the caller of the routine which ultimately causes the error message to show. [42, pp. 69–73]

```

129 /**
130  * Connects to the URL specified in the control in the `ServiceNow` tab using
131  * basic authentication with
132  * username and password that are also supplied in this tab.
133  *
134  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
135  */
136 ::routine testServiceNow public
137 use arg slotDir
138 /* validate the form fields first */
139 IDs = "servicenowURL", "servicenowUsername", "servicenowPassword"
140 if \validateForm(IDs) then return -- leave the routine if .false returned
141 /*@get(servicenowURL servicenowUsername servicenowPassword)*/
142 /* build the REST request using the `Unirest` API */
143 button = .my.app~SAA.fxml~testServiceNowButton
144 call setLoadingSymbolToButton button
145 call SysSleep .5 -- wait 500ms for Java to update the GUI
146 url = servicenowUrl~getText
147 user = servicenowUsername~getText
148 pass = servicenowPassword~getText
149 /* for testing we use a PUT request on the custom rest api with a magic table
150  name containing { } */
151 API.URL = url || "/api/x_146620_serviceno/create_table/%7BtestApiExistence%7D"
152 rexxProxy = bsfCreateRexxProxy(.TestServiceNowCallback~new,, .Callback)
153 signal on syntax
154 .Unirest~put(API.URL) -
155     ~basicAuth(user, pass) -
156     ~asJsonAsync(rexxProxy)

```

```

155     return
156
157     syntax:
158         button~setGraphic(.nil)           -- remove the loading symbol
159         raise propagate                   -- show the error message

```

Listing 4.24.: testServiceNow Routine

The *TestServiceNowCallback* class implements the three methods

- completed
- failed
- run

Completed and *failed* are needed for the *Callback* Java interface, while *run* is invoked by the *Runnable* interface. JavaFX handles all GUI-related operations in the so called "JavaFX Application Thread". If any other threads want to change GUI controls, they need to use the *~runLater()* method of the static *Platform* Java class, as used in lines 179 and 193, which will invoke the *run* method.

Both the *completed* and *failed* methods create a *userData* directory with either the response or the exception and provide that to the Rexx proxy class as second argument of the *bsfCreateRexxProxy()*. This causes the invoked methods through that proxy to hold the *userData* in the *slotDir* argument, which is automatically supplied as last argument anyway.

The *run* method extracts the status code in case the API call returned with a response and not an error. If it was 200, the REST API works well and the button receives a tick mark using the *setCheckGraphicToButton* routine. Otherwise an error message with the status code, the exact server response and a hint what could have gone wrong is displayed in the expendable content area of JFXAlert using the *showPopup* routine in lines 213 to 219, which is discussed in listing 4.19 on page 40. In case of an exception, the message, cause and stack trace are extracted from it and displayed accordingly. The method "failed" is invoked when the host cannot be reached, so the ServiceNow URL is probably wrong. Either way, this class provides very exact information on the nature of an error.

```

161  /**
162   * A Rexx Proxy Class. Java can invoke methods in this class
163   */
164  ::class TestServiceNowCallback
165  /**
166   * The REST API call was successful and returns a response
167   * Examines the status code and displays the result to the user
168   *
169   * @param response - <code>com.mashape.unirest.http.HttpResponse</code>
170   */
171  ::method completed
172      use arg response
173      userData = .stringTable~new
174      userData~response = response
175      /* create another Proxy - let this class carry out the invoked method "run" */
176      runnable = bsfCreateRexxProxy(self, userData, .Runnable)
177      /* since this Async call operates in a separate Thread, we cannot change
178         controls in the user interface
179         so we need to use Platform.runLater() to queue the GUI changes in the main
180         JavaFX Application Thread */
181      .Platform~runLater(runnable)

```

```

182 * The REST API call was not successful, probably because the host is not
    reachable
183 * Displays an error message with the stack trace
184 * Platform~runLater() is used to execute GUI changes
185 *
186 * @param exception -
    <code>com.mashape.unirest.http.exceptions.UnirestException</code>
187 */
188 ::method failed
189     use arg exception
190     userData = .stringTable~new
191     userData~exception = exception
192     runnable = bsfCreateRexxProxy(self, userData, .Runnable)
193     .Platform~runLater(runnable)
194
195 /**
196 * Called by a java.lang.Runnable interface
197 * Decides based on the userData in the slotDir argument, whether the response
    is 200 OK and a tick mark has to be
198 * displayed, otherwise shows a popup with some information about the error
199 *
200 * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
201 */
202 ::method run
203     use arg slotDir
204     userData = slotDir~userData
205     button = .my.app~SAA.fxml~testServiceNowButton
206     if userData~hasEntry("response") then do
207         response = userData~response
208         statusCode = response~getStatus
209         message = ""
210         if statusCode = 200 then
211             call setCheckGraphicToButton button
212         else do
213             if statusCode = 400 then
214                 message ||= "You probably forgot to import the custom REST APIs. Please
                    have a detailed look at the information in the `Home` tab."
                    .endOfLine~copies(2)
215             else if statusCode = 401 then
216                 message ||= "Wrong credentials!" .endOfLine~copies(2)
217             message ||= "Status Code:" statusCode .endOfLine
218             message ||= "Server Response:" response~getBody~toString
219             call showPopup message
220             button~setGraphic(.nil) -- remove the loading graphic
221         end
222     end
223 else do
224     /* no response, so method "failed" invoked this runnable */
225     exception = userData~exception
226     message = "The Server could not be reached!" .endOfLine
227     message ||= "Message:" exception~getMessage .endOfLine
228     message ||= "Cause:" exception~getCause .endOfLine
229     message ||= "Stack Trace:" exception~getStackTrace .endOfLine
230     call showPopup message
231     button~setGraphic(.nil)
232 end

```

Listing 4.25.: TestServiceNowCallback Class

Saving the ServiceNow credentials works just like saving Database information. After validating the three input fields, their values are saved in the *rxObj stringTable* and supplied

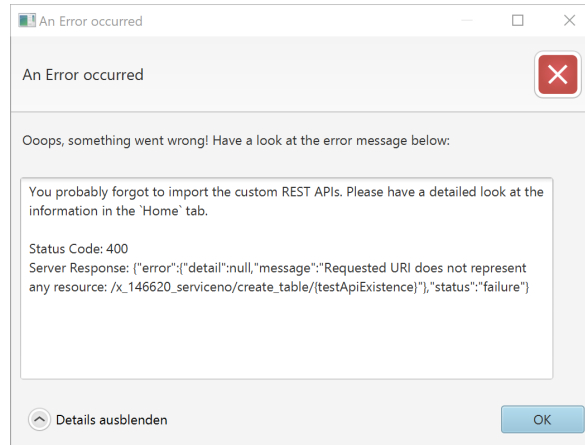


Figure 4.7.: Test ServiceNow returns an error

to the `saveRexxObjToJson()` routine with the "ServiceNow" string identifier. If this call returns `.true`, a tick mark is added to the left button showing the user that all went well.

```

106 /**
107  * Saves the content of the three input controls in the `ServiceNow` tab
108  *
109  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
110  */
111 ::routine saveServiceNow public
112   use arg slotDir
113   /* validate the form fields first */
114   IDs = "servicenowURL", "servicenowUsername", "servicenowPassword"
115   if \validateForm(IDs) then return -- leave the routine if .false returned
116   scriptContext = slotDir~scriptContext
117   /*@get(servicenowURL servicenowUsername servicenowPassword
118       saveServiceNowButton)*/
119   /* create a Rexx Object to hold all information */
120   rxObj = .stringTable~new
121   rxObj~url = servicenowURL~getText
122   rxObj~user = servicenowUsername~getText
123   rxObj~pass = servicenowPassword~getText
124   success = saveRexxObjToJson(rxObj, "ServiceNow")
125   /* check if there was an error while saving, caused by lack of password */
126   if success then
127       call setCheckGraphicToButton saveServiceNowButton

```

Listing 4.26.: saveServiceNow Routine

Figures 4.7 and 4.8 show the error message with the status code 400 and the ServiceNow App Accelerator after successfully testing and saving ServiceNow credentials.

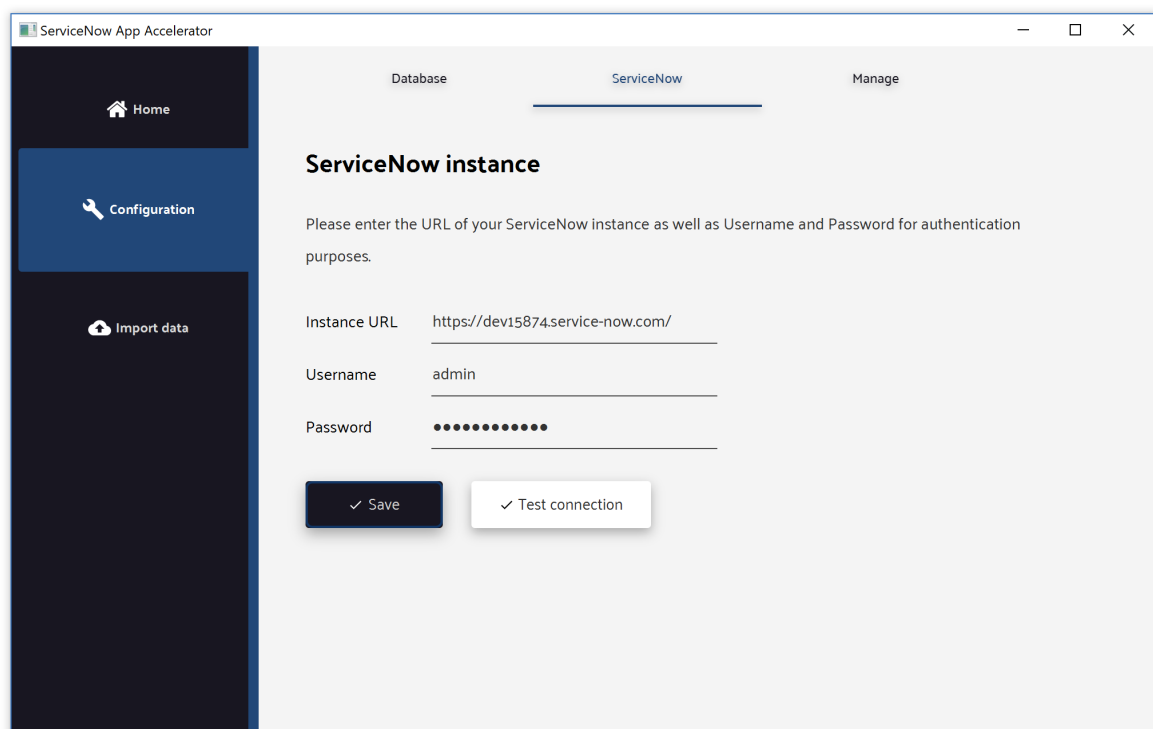


Figure 4.8.: ServiceNow Connection tested and saved

4.6. Manage Credentials

The last configuration tab "Manage credentials" shows a combo box to switch between Database and ServiceNow credentials, as well as a table beneath to display the data as depicted in figure 4.9 There are actually two tables, one for each credentials type - they each use the *visible* and *managed* attributes to hide the currently non-active table. The FXML source code of the database table shows, that a *JFXTreeTableView* is used. This is the material design implementation of a traditional *TreeTableView* which additionally offers a grouping mechanism. [43]

```

383 <JFXTreeTableView fx:id="configurationManageDatabases" managed="true"
    visible="true" prefHeight="260.0">
384     <VBox.margin>
385         <Insets top="15.0" />
386     </VBox.margin>
387     <columns>
388         <JFXTreeTableColumn text="Type" prefWidth="100.0" />
389         <JFXTreeTableColumn text="Host" prefWidth="100.0" />
390         <JFXTreeTableColumn text="Port" prefWidth="60.0" />
391         <JFXTreeTableColumn text="Name" />
392         <JFXTreeTableColumn text="Username" />
393         <JFXTreeTableColumn text="Password" />
394     </columns>
395     <columnResizePolicy>
396         <TreeTableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
397     </columnResizePolicy>
398 </JFXTreeTableView>

```

Listing 4.27.: JFXTreeTableView

Tables are quite difficult to manage in JavaFX, because they offer lots of built-in features for the correlated data such as sorting or auto-update functionality. It is not as simple as in html, where one only includes the data row by row - instead, so called "data models" have to be applied. Those are simple classes that hold some attributes which will be eventually displayed in the respective table columns. Therefore, each object has its own row and the programmer does not have to manipulate the table or its content, but the objects it represents.

The data model for the ServiceNow credentials has three attributes "url", "username" and "password". The constructor method optionally fetches the argument of class *.stringTable* and extracts all information about the ServiceNow connection.

```

282 /**
283  * The data model class "ServiceNow" holds all attributes that are to be shown
    in the affiliated Table in the `Manage` Tab
284  */
285 ::class ServiceNow
286 ::attribute url
287 ::attribute username
288 ::attribute password
289 /**
290  * Constructor method
291  * Extracts the information of the supplied stringTable and saves them as class
    attributes
292  *
293  * @param informationDirectory - a <code>stringTable</code> which holds all
    saved information about
294  *
    a ServiceNow connection as a result from
    calling <code>getRexxObjFromJson</code>

```

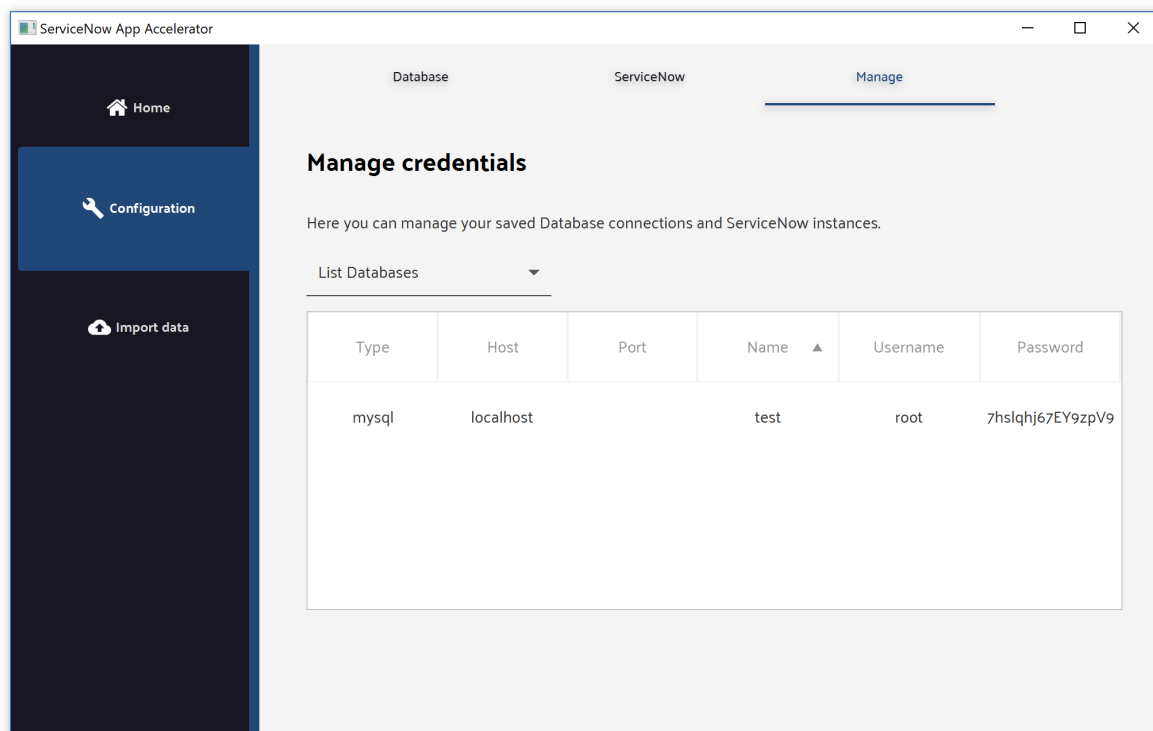


Figure 4.9.: Manage credentials

```

295 */
296 ::method init
297   expose url username password
298   use arg informationDirectory
299   if informationDirectory~isA(.stringTable) then do
300     url = informationDirectory~url
301     username = informationDirectory~user
302     password = informationDirectory~pass
303   end

```

Listing 4.28.: ServiceNow data model in ServiceNowAppAccelerator.rxd

The database data model is very similar, albeit it holds more attributes and the option to save and add tables. The *addTable* method will be used in the final Import Task to add tables to a given database.

```

307 /**
308  * The class "Database" holds connection and schema information about one
309  * database
310  */
311 ::class Database
312 ::attribute name
313 ::attribute type
314 ::attribute host
315 ::attribute url
316 ::attribute port
317 ::attribute username
318 ::attribute password
319 ::attribute tables
320 /**
321  * Constructor method

```

```

321 * Extracts the information of the supplied stringTable and saves them as class
    attributes
322 *
323 * @param informationDirectory - a <code>stringTable</code> which holds all
    saved information about a Database
324 *                               as a result from calling
    <code>getRexxObjFromJson</code>
325 */
326 ::method init
327     expose tables name type host url port username password
328     use arg informationDirectory
329     tables = .array~new
330     if informationDirectory~isA(.stringTable) then do
331         name = informationDirectory~dbName
332         type = informationDirectory~type
333         host = informationDirectory~host
334         port = informationDirectory~port
335         url = informationDirectory~url
336         username = informationDirectory~user
337         password = informationDirectory~pass
338     end
339
340 /**
341 * Adds a <code>DbTable</code> to the `tables` collection
342 *
343 * @param table - An instance of the DbTable class defined in this file
344 */
345 ::method addTable
346     expose tables
347     use arg table
348     tables~append(table)

```

Listing 4.29.: Database data model

The *start* method of the *RxApplication* class created a new instance of the *JFXTableManager* just before showing the graphical user interface in line 114 of listing 4.7 on page 29. This class is supposed to connect the data models with the tables through cell value factories. The constructor simply calls the *setUpCellValueFactoryAndRootNode* method with a reference to each table individually. The mentioned method loops over all columns of the table and extracts the column name from the fxml file. This information is used to create a Factory that implements the Java *Callback* interface of an instance of the *PropertyValueFactory* class. This Rexx proxy is set as cell value factory of the column and will answer to calls about the expected content of the cell.

```

395 /**
396 * The class `JFXTableManager` manages both tables from the "Manage" Tab
397 */
398 ::class JFXTableManager
399 /**
400 * Constructor method
401 */
402 ::method init
403     expose databaseTable servicenowTable
404     /* get a reference of both tables and save them in the class */
405     databaseTable = .my.app~SAA.fxml~configurationManageDatabases
406     servicenowTable = .my.app~SAA.fxml~configurationManageServiceNowInstances
407     self~setUpCellValueFactoryAndRootNode(databaseTable)
408     self~setUpCellValueFactoryAndRootNode(servicenowTable)
409

```

```

410 /**
411  * Sets up the CellValueFactory for every column in the supplied table
412  * After that, it creates an empty root node for the Table
413  *
414  * @param table - A <code>JFXTreeTableView</code>
415  */
416 ::method setUpCellValueFactoryAndRootNode private
417   use strict arg table
418   loop column over table~getColumns
419     heading = column~getText
420     factory = .PropertyValueFactory~new(heading)
421     rexxProxy = bsfCreateRexxProxy(factory,,.fxCallback)
422     column~setCellValueFactory(rexxProxy)
423   end
424   /* create an empty TreeObject */
425   RecursiveTreeObject =
426     .bsf~new("com.jfoenix.controls.datamodels.treetable.RecursiveTreeObject")
427   callbackFunction = bsfCreateRexxProxy(.GetChildrenCallback~new,,.fxCallback)
428   root = .bsf~new("com.jfoenix.controls.RecursiveTreeItem", RecursiveTreeObject,
429     callbackFunction)
430   /* expand the root node to automatically display all its child nodes */
431   root~expanded = .true
432   /* set the root node in the table and hide it, because it is empty anyway */
433   table~root = root
434   table~showRoot = .false

```

Listing 4.30.: JFXTableManager Class

After the set up of each column, an empty *RecursiveTreeItem* is created and set as root of the table in lines 425 to 427, whose constructors require a *RecursiveTreeObject* as first argument and a callback function as second.[44] The *GetChildrenCallback* class is used here to simply return all child nodes of the root node on call. The root node itself is hidden, because it contains no data and just acts as a hook for the objects holding real data.

```

549 /**
550  * Another Callback function for retrieving the children of the data model
551  * RecursiveTreeObject
552  */
553 ::class GetChildrenCallback
554 /**
555  * Supplies the caller with the child nodes of RecursiveTreeObject
556  *
557  * @param RecursiveTreeObject - supplied by the caller on Java side. Holds the
558  * proxied extended java class `jObj`
559  * @return The children of the parameter
560  */
561 ::method call
562   use arg RecursiveTreeObject
563   return RecursiveTreeObject~getChildren

```

Listing 4.31.: GetChildrenCallback Class

The method *setUpCellValueFactoryAndRootNode* used the *.PropertyValueFactory* class in line 420 by supplying the constructor with the column name. Since this class implements the Java *Callback* interface, it will invoke the *call* method whenever the table requests new data. The supplied argument *CellDataFeatures*, which is used in line 540, is a nested class of the *TableColumn* class and is bound to a specific table, a column and a value. [45]

By extracting the value of the *RecursiveTreeItem* in line 542, *jObj* represents the *RecursiveTreeObject* and is used to retrieve the underlying Rexx proxy class. Its *~sendMessage0()*

method is utilized to send a message with the column name to the Rexx object to receive the attribute value which is then returned and used as cell value. Again, each column has its own *PropertyValueFactory*, which retrieves a *RecursiveTreeItem*, that is an equivalent of a table row out of the Callback interface call. Having both column and row, it makes it possible to fetch the cell value.

```

517 /**
518  * implements javafx.util.Callback<P,R>(P o)
519  * This class allows instances, that remember the message to be sent to the data
    model instances to
520  * return the attribute that should be shown in the table cell.
521  */
522 ::class PropertyValueFactory
523 /**
524  * This constructor method saves the attribute name in the class
525  *
526  * attributeName - The name of the Attribute in the data model class
527  */
528 ::method init
529     expose attributeName
530     use strict arg attributeName
531
532 /**
533  * Is called by the Callback Interface whenever the Table needs new Data
534  *
535  * @param CellDataFeatures - is a nested class of <code>TreeTableColumn</code>
    and holds information about the Table
536  * @return The attribute value from the data model camouflaged as the java proxy
    class
537  */
538 ::method call
539     expose attributeName
540     use arg CellDataFeatures          -- an observable value for the ooRexx object
        boxed in a Java RexxProxy object
541     RecursiveTreeItem = CellDataFeatures~getValue
542     jObj = RecursiveTreeItem~getValue
543     rexxProxy = jObj~getTargetRexxProxy
544     attributeValue = rexxProxy~sendMessage0(attributeName)
545     return .SimpleStringProperty~new(attributeValue)

```

Listing 4.32.: PropertyValueFactory Class

At this point, the infrastructure for the table is installed including a root node as hook for all table rows. Now the data model objects have to be created. The *configurationTabPaneChangeListener* class observes the selected tabs "Database", "ServiceNow" and "Manage" in listing 4.12 on page 33. In case the user switches to the last tab, the method *~showAndFill()* of the self-built *JFXTableManager* class gets invoked with the value of the combo box as argument, which can be either "ServiceNow" or "Database".

The invoked *showAndFill* method makes use of references to both tables which were already saved in the object when instantiating the *JFXTableManager* class. The compulsory argument of this method determines which table is saved in the *currentTable* class attribute and which one should be hidden by calling the *showTable* method with the second argument set to *.false*. The methods *showTable*, *clearData* and *fillTable* are finally called to update the data in the *JFXTreeTableView*.

The method *showTable* checks the existence of the first argument, otherwise it defaults to the class attribute *currentTable* in line 471. Then, it shows or hides the mentioned table based on the second argument, which defaults to *.true*. So this method basically shows

the *currentTable* if no arguments were specified. The method *clearData* on the other hand removes all rows of the table.

```

434 /**
435  * Is called by value change listeners whenever the user visits the `Manage` Tab
436  * or changes the value of the manageComboBox.
437  * This method calls a number of functions to show the right table, to fill it
438  * with data and to hide the other table
439  *
440  * @param tableDescription - The value of the ComboBox in the `Manage Tab`
441  */
442 :method showAndFill
443   expose databaseTable servicenowTable currentTable
444   use strict arg tableDescription
445   /* select the right table based on the second word of the supplied argument */
446   select case tableDescription~word(2)
447     when "Databases" then do
448       currentTable = databaseTable
449       self~showTable(servicenowTable, .false) -- hide the servicenowTable
450     when "ServiceNow" then do
451       currentTable = servicenowTable
452       self~showTable(databaseTable, .false)
453     end
454     otherwise
455       raise syntax 40.900 array("Bad value. Cannot refresh the Data in the
456         Table")
457   end
458   self~showTable -- show currentTable
459   self~clearData -- clear currentTable
460   self~fillTable -- fill it with fresh data
461 /**
462  * Hides or shows the supplied table. If there is none, it will use the current
463  * table from the class
464  *
465  * @param table - The <code>JFXTreeTableView</code> to show/hide
466  * @param [show]- .true to show, .false to hide. Default: .true
467  */
468 :method showTable private
469   expose currentTable
470   use arg table, show = .true
471   if \table~isA(.bsf) then
472     table = currentTable
473   table~~setManaged(show) -
474     ~~setVisible(show)
475 /**
476  * Clears the Data from the current Table
477  */
478 :method clearData private
479   expose currentTable
480   currentTable~getRoot~getChildren~clear

```

Listing 4.33.: The JFXTableManager methods showAndFill, showTable and clearData

The *fillTable* method retrieves the current data from the encrypted json file as described in listing 4.22 on page 44. Then, it uses the ternary operator *~?()* to decide if it holds Database or ServiceNow related items, based on the amount of columns of the *currentTable*. This information is used to retrieve the right data set from the json object in line 490.

Then, the `.GetChildrenCallback` class is instantiated as Rexx proxy that implements the `Callback` interface. This will later be needed as second argument for the construction of `RecursiveTreeItem` in line 512.

Line 497 uses a new `BSF4ooRexx` routine `bsf.createProxyClass()` that creates a new Java class on the fly and extends the supplied Java class name "RecursiveTreeObject". Unfortunately, JFoenix requires all data models to do that.

In lines 499 to 513, the application iterates over all data from the json object. It creates a new data model object with a directory holding information about one row of credentials as argument in line 506. Then, it boxes this Rexx object as a Java proxy class in line 508 by not supplying any interfaces to the `bsfCreateRexxProxy` routine. Line 510 finally extends this class with the "RecursiveTreeObject" class, creates a tree item and attaches it to the root node of the table, which fills the table with data.

```

482 /**
483  * Retrieves fresh saved Data from the JSON file and displays it in the current
484  * Table
485  */
486 ::method fillTable private
487     expose currentTable
488     jsonObj = getRexxObjFromJson()
489     /* decide based on the number of columns whether this is a Database or
490      * ServiceNow table */
491     kind = (currentTable~getColumns~size > 5)~?("Database", "ServiceNow")
492     jsonData = jsonObj[kind~upper]
493     root = currentTable~getRoot
494     /* this callback function retrieves the children of the tree item on call */
495     callbackFunction = bsfCreateRexxProxy(.GetChildrenCallback~new,,.fxCallback)
496     /* extend the Java class RecursiveTreeObject, which is the data model that is
497      * used in JFXTreeTableView
498      * cf. <b>Note:</b> the data object used in JFXTreeTableView <b>must</b>
499      * extends this class
500      * https://github.com/jfoenixadmin/JFoenix/blob/master/jfoenix/src/main/jav_
501      * a/com/jfoenix/controls/
502      * /datamodels/treetable/RecursiveTreeObject.java */
503     extendRecursiveTreeObject = bsf.createProxyClass("com.jfoenix.controls_
504     .datamodels.treetable.RecursiveTreeObject")
505     /* iterate over the queue - `informationTable` will be a stringTable */
506     loop informationTable over jsonData
507     /* create a Rexx Object of a Rexx Class based on the kind of Table we are
508      * trying to fill with data */
509     if kind = "Database" then
510         useClass = .Database
511     else
512         useClass = .ServiceNow
513     rxObj = useClass~new(informationTable)
514     /* box this rexx object as a java proxy class */
515     rexxProxy = bsfCreateRexxProxy(rxObj)
516     /* extend our java class with `RecursiveTreeObject`, as this is required by
517      * JFoenix */
518     jsonObj = extendRecursiveTreeObject~new(reeXProxy)
519     /* create a new instance of the RecursiveTreeItem with our data model and a
520      * callback function as parameters */
521     RecursiveTreeItem = .bsf~new("com.jfoenix.controls.RecursiveTreeItem", jsonObj,
522     callbackFunction)
523     /* attach it to the root node */
524     root~getChildren~add(RecursiveTreeItem)

```

Listing 4.34.: JFXTableManager's method fillTable

4.7. The Import Task

After the user saves all his credentials in the "Configuration" tab, he can switch to the last option "Import data" on the main tab pane to the left. He is then presented with two combo boxes and a "Connect" button as shown in figure 4.10. The *mainTabPaneChangeListener* class is responsible for refreshing the values of those two combo boxes every time the user enters the "Import data" tab, as shown in listing 4.12 on page 33.

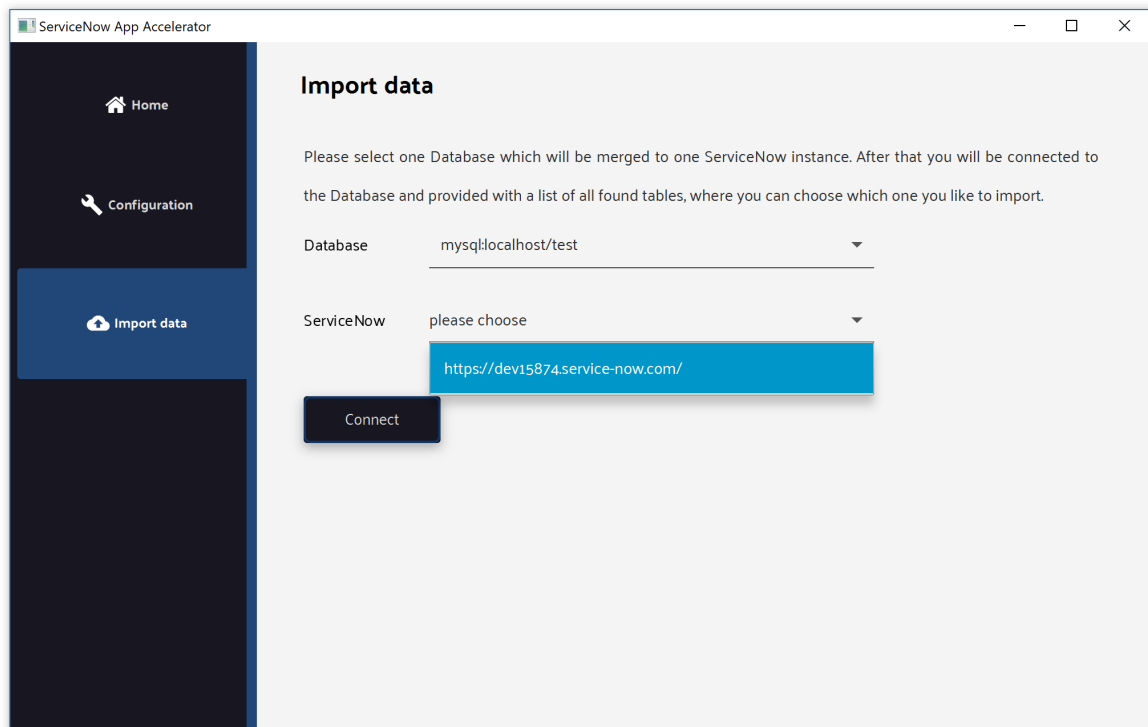


Figure 4.10.: Application Start Screen

By clicking on the "Connect" button, the *prepareImport* routine gets called. First, both combo boxes are checked for validity using the *validateForm* routine to make sure that a value is selected. After that, the application fetches all references to both combo boxes and two currently invisible user interface controls:

- **databaseAfterConnect** is a short text to let the user know, that the connection to the database went well.
- **databaseCheckboxPane** is a layout component that holds all table names wrapped in check boxes. This way, the user can select which tables he would like to import to his ServiceNow instance.

After extracting the table names from the selected database, the application removes all check boxes except for the first one in line 459. The first check box has the predefined label "Select all" and is described in listing 4.36. The loop in line 461 to 464 creates new *JFXCheckBox* items with the table name as constructor argument which will be used as control label by the check box and adds it to the layout pane. This results in a list of check boxes holding solely the table names of the database.

```

431 /**
432  * Initialises an object of the .importTask class with the database and
    servicenow credentials
433  * Shows the Pane with the fx:id "databaseAfterConnect" including its content
434  * Displays a list of tables in the database in form of checkboxes
435  *
436  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
437  */
438 ::routine prepareImport public
439     use arg slotDir
440     signal on syntax
441     scriptContext = slotDir~scriptContext
442     /* validate the form fields first */
443     IDs = "importDatabase", "importServiceNow"    -- create an array with those
        two items
444     if \validateForm(IDs) then return -- leave the routine if .false has returned
445     /* @get(importDatabase importServiceNow databaseAfterConnect
        databaseCheckboxPane) */
446     /* fetch the indices of the selected ComboBox items and supply them to the
        .ImportTask class */
447     databaseIndex = importDatabase~getSelectionModel~getSelectedIndex
448     servicenowIndex = importServiceNow~getSelectionModel~getSelectedIndex
449     /* interrupt the last Thread if necessary */
450     if .environment~hasEntry("jTask") then
451         .jTask~cancel(.true)
452     .my.app~importTask = .ImportTask~new(databaseIndex, servicenowIndex)
453     /* call a method to load tables from the database */
454     tableNames = .my.app~importTask~extractTables
455     checkboxes = databaseCheckboxPane~getChildren
456     /* first checkbox should always be "Select all" - delete the rest if available
        */
457     items = checkboxes~size
458     if items > 1 then
459         checkboxes~remove(1, items)    -- (from,to) 0 based indices
460     /* now append each table name as a checkbox */
461     loop tableName over tableNames
462         checkbox = .bsf~new("com.jfoenix.controls.JFXCheckBox", tableName)
463         checkboxes~add(checkbox)
464     end
465     databaseAfterConnect~visible = .true
466     return
467     syntax:
468         call showPopup "Connection failed!"

```

Listing 4.35.: prepareImport Routine

The "Select all" check box calls the *selectAllCheckboxes* routine, which propagates the value of the mentioned check box to all other check boxes in the *databaseCheckboxPane*, selecting or de-selecting all of them at once.

```

470 /**
471  * Propagates the selected property of the "Select all" checkbox to all other
    checkboxes in the `databaseCheckboxPane`
472  *
473  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
474  */
475 ::routine selectAllCheckboxes public
476     use arg slotDir
477     checkboxes =
        slotDir~scriptContext~getAttribute("databaseCheckboxPane")~getChildren

```

```

478   selectAllCheckbox = slotDir~scriptContext~getAttribute("event")~target
479   loop checkbox over checkboxes
480     checkbox~selected = selectAllCheckbox~isSelected
481   end

```

Listing 4.36.: selectAllCheckboxes Routine

The *prepareImport* routine created a new instance of the *ImportTask* class using the indices of the Database and ServiceNow credentials in lines 447 to 452. Since the combo boxes show the values according to their sequence in the json file, passing the indices to the constructor of this class suffices. It is not necessary to access the json file for connection details here.

The *ImportTask* class is very complex as it offers all methods used for both extracting database data and sending it to the ServiceNow instance via REST API calls. The constructor method saves some important components as attributes in the class so that subsequent methods can easily access them. Those components are:

- **jsonEngine** used for reading and writing json files
- **importProgressbar** is the JFXProgressBar at the bottom of the progress view after starting the import process. Figure 4.12 on page 65 shows the structure of the progress view.
- **steps** is an array with fixed values of all four steps the application is going to go through in the import task. More specifically, this array holds the *fx:id* attributes of the panes in the next view which will show the progress of the task.
- **step** is the current index of the *steps* array. The import task will obviously begin with the first step, that is why it is set to 1.
- **databaseTables** is an array that will be filled in the *extractTables* method with table names of this database.
- **databaseCredentials** and **serviceNowCredentials** store information about the credentials, extracted from the encrypted json file using the routine *getRexxObjFromJson* in line 537.
- **serviceNowURL** is the URL of the ServiceNow instance.

The method *~setDefaultHeader("Authorization", basicAuth)* from the *Unirest* class in line 542 sets a basic authentication string as default authorization header to every Unirest call from now on. The credentials are not going to change for this *.ImportTask* object at all, that is why this will result in a simplification for all future API calls. Furthermore, the *~setTimeouts(0,0)* method in line 544 removes all timeouts, so that Unirest never stops trying to connect to the API. The default value for a connection is 10 seconds. However if the host has a slow internet connection, it appears possible that Unirest would throw connection errors, which we want to avoid at this point. Finally, the *connectToDatabase* method is called to establish a connection with the database.

```

515  /**
516   * The main heart of the application: does the extracting and uploading job in a
      separete thread
517   */
518  ::class ImportTask
519  /**
520   * This constructor method saves the database and servicenow credentials in the
      class
521   *
522   * @param databaseIndex - The 0 based index of the selected database connection
523   * @param databaseIndex - The 0 based index of the selected servicenow instance

```

```

524 */
525 ::method init
526   expose databaseCredentials serviceNowCredentials serviceNowURL databaseTables
527   importProgressbar jsonEngine step steps
528   use strict arg databaseIndex, servicenowIndex
529   jsonEngine = .json~new
530   importProgressbar = .my.app~SAA.fx.xml~importProgressbar
531   importProgressbar~setProgress(-1.0) -- indeterminate progress
532   /* create an array which holds all fx:ids of panes to get a hook for GUI
533      manipulation in the `unlockNextStep` method */
534   steps = .array~of("readingDatabasePane", "creatingTablesPane",
535                     "uploadingDataPane", "allDonePane")
536   step = 1
537   databaseTables = .array~new
538   databaseIndex += 1
539   servicenowIndex += 1
540   jsonObj = getRexxObjFromJson()
541   databaseCredentials = jsonObj~Database[databaseIndex] -- only extract
542   information about this Database connection
543   serviceNowCredentials = jsonObj~ServiceNow[servicenowIndex] -- only extract
544   information about this ServiceNow instance
545   serviceNowURL = serviceNowCredentials~url
546   basicAuth = self~getBasicAuth(serviceNowCredentials)
547   .Unirest~setDefaultHeader("Authorization", basicAuth)
548   /* remove both timeouts for connection and socket (default values are 10s and
549      1m) */
550   .Unirest~setTimeouts(0,0)
551   self~connectToDatabase(databaseIndex)

```

Listing 4.37.: ImportTask Constructor

The *getBasicAuth* method creates an authentication string in the format "username:password" in lines 556 to 558 and encodes it with base64 encoding string using the method *~encodeBase64()* of the Rexx' *.string* class and returns the result.

```

547
548
549 /**
550  * Base64 encodes a string with the username and password of the ServiceNow
551  * instance for the REST API calls
552  *
553  * @param credentials - a .stringTable with the entries "user" and "pass"
554  */
555 ::method getBasicAuth private
556   use arg credentials
557   user = credentials~user
558   pass = credentials~pass
559   authString = user ":" pass
560   basicAuth = "Basic" authString~encodeBase64 -- encodeBase64 is a method of
561   the Rexx' string class
562   return basicAuth

```

Listing 4.38.: ImportTask's getBasicAuth Method

The *connectToDatabase* method uses the *dbConnect* routine with the database information of the class to get a connection. Thus, it creates a new default *Statement* object with the use of the established connection and saves it in the class as *databaseStatement*. This variable can now be used to interact with the database by executing SQL commands within the context of a connection, which will be exploited after the user starts the import task. This concludes the constructor method of the *ImportTask* class.

```

562 /**
563  * Calls the `dbConnect` function in its own thread
564  *
565  * @param databaseIndex - the 1-based index of the database to connect to
566  */
567 ::method connectToDatabase
568   expose databaseCredentials databaseConnection databaseStatement
569   use arg databaseIndex
570   d = databaseCredentials
571   databaseConnection = dbConnect(d~type, d~host, d~user, d~pass, d~dbName,
572     d~port)
573   databaseStatement = databaseConnection~createStatement

```

Listing 4.39.: ImportTask's connectToDatabase Method

Previously, the *prepareImport* routine in listing 4.35 invoked the *extractTables* method of the *importTask* object in line 454. This method extracts only the available table names from the database with help of the now established *databaseStatement* attribute.

```

454 tableNames = .my.app~importTask~extractTables

```

The Java Database Connector generally offers a lot of functionality regarding meta data. A connection object provides information about the database's tables, version, all stored procedures and more. This information is obtained with the *getMetaData* method in line 667. Whereas the *getTables* method can be utilized to extract all tables by supplying the wildcard operator *%* as third argument. [46]

```

660 /**
661  * Extracts the table names with the help of meta data
662  *
663  * @return tableNames - an array of table names in the database
664  */
665 ::method extractTables
666   expose databaseConnection tableNames
667   metaData = databaseConnection~getMetaData
668   resultSet = metaData~getTables(.nil, .nil, "%", .nil)
669   tableNames = .array~new
670   loop while resultSet~next
671     tableName = resultSet~getString("TABLE_NAME")
672     tableNames~append(tableName)
673   end
674   return tableNames

```

Listing 4.40.: ImportTask's extractTables Method

Back to the user interface, the *startImport* routine is called when clicking on the "Start" button as shown in figure 4.11. The routine extracts all table names and returns with an error message in lines 496 and 497 if no tables are present. Otherwise, it saves all selected table names in the *importTask* object with help of the *addTable* method.

The "Import data" tab actually consists of another JFXTabPane with hidden tab controls. That is why the instruction in line 507 selects the next tab in order to navigate to the import view as depicted in figure 4.12. This is because the JFXTabPane has a nice sliding animation for switching tabs, which is bound to this class and could not be used otherwise. After waiting for 500 milliseconds for the graphical user interface to complete the animation, the already initialized *importTask* object is used to create a Rexx proxy that implements the Java *Task*

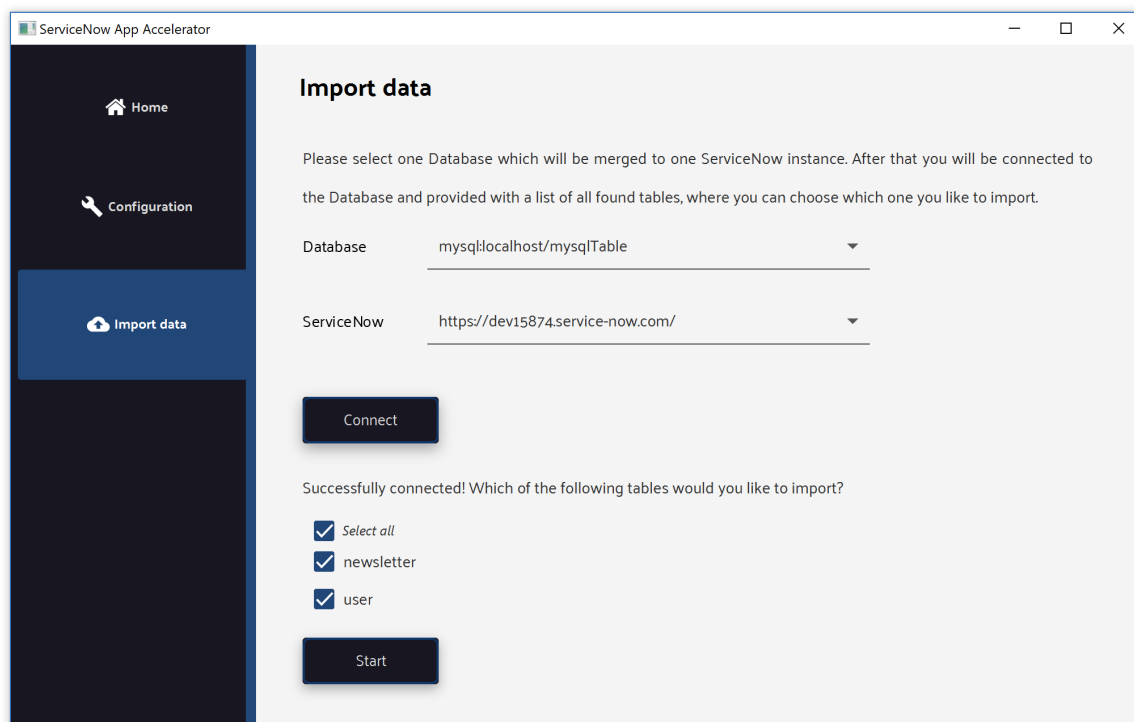


Figure 4.11.: Table selection

interface and supplied to the *execute* method of the *ExecutorService* instance to start a new thread by invoking the *call* method.

```

484 /**
485  * Saves the database table choice in the .ImportTask class and starts the
    import job as a Task
486  *
487  * @param slotDir - BSF4ooRexx supplies the SlotDir argument at the Java side
488  */
489 ::routine startImport public
490 use arg slotDir
491 scriptContext = slotDir~scriptContext
492 /* @get(importTabPane databaseCheckboxPane) */
493 checkboxes = databaseCheckboxPane~getChildren
494 numOfTables = checkboxes~size - 1
495 if numOfTables = 0 then do
496     call showPopup "No tables selected"
497     return
498 end
499 loop i=1 to numOfTables -- prevent IndexOutOfBoundsException
500     checkbox = checkboxes~get(i)
501     if checkbox~isSelected then do
502         tableName = checkbox~getText
503         .my.app~importTask~addTable(tableName)
504     end
505 end
506 /* switch the view to the Log Area and the Progress Bar to accompany the user
    with more information */
507 importTabPane~getSelectionModel~selectNext
508 call SysSleep .5 -- wait for 500 ms for Java to update the GUI
509 /* execute the import task process in its own Thread to not block the GUI */

```

```

510 .environment~jTask =
    bsfCreateRexxProxy(.my.app~importTask,, "javafx.concurrent.Task")
511 .Executor~execute(.jTask)

```

Listing 4.41.: startImport Routine

The overridden *call* method from the *Task* interface gets invoked by the *Executor* to do some background work in its own thread. The method first invokes *calculateTotalApiCalls* to get an estimate on how many items all tables combined have. This is needed for the progress bar to calculate the current progress.

Lines 583 to 586 print out the number of tables and data to the console as an information for the user. Line 589 completes the step "Reading database table schema" and unlocks step 2 "Creating ServiceNow tables". All table names are sent to the custom REST API in the *createServiceNowTable* method individually, whose return value is the ServiceNow adjusted table name. This value is saved in an two-dimensional array *serviceNowTables* alongside the original table name of the database table. Now this array can be used to map database table names to ServiceNow table names.

After creating all tables, the application proceeds to step 3 "Uploading data". This step solely invokes the *uploadData* method with the Database and ServiceNow table names. The switch to the last step 4 "All done" will be done in another method when the last callback of the *uploadData* returns.

```

575 /**
576  * Iterates over all selected tables, extracts meta information and calls the
    REST API to create new tables
577  * Runs in a separate thread
578  */
579 :method call
580   expose databaseTables
581   signal on syntax
582   totalApiCalls = self~calculateTotalApiCalls
583   tableNum = databaseTables~items
584   dataNum = totalApiCalls - tableNum      -- one api call per table creation +
    data insertion
585   s = (tableNum <> 1)~?("s", "")
586   self~log(dataNum "records found in" tableNum "table" || s)
587   serviceNowTables = .array~new
588   /* show loading symbol at "creating tables" pane */
589   self~unlockNextStep
590   loop tableName over databaseTables
591     /* gather information for the REST API call */
592     serviceNowTable = self~createServiceNowTable(tableName)
593     if serviceNowTable <> .nil then do
594       /* both the database table name and the servicenow table name are needed,
        so save them in an array */
595       tableNames = .array~of(tableName, serviceNowTable)
596       serviceNowTables~append(tableNames)
597     end
598   end
599   /* show loading symbol at "uploading data" pane */
600   self~unlockNextStep
601   /* all tables are created, now insert data */
602   loop tableNames over serviceNowTables
603     dbTable = tableNames[1]
604     serviceNowTable = tableNames[2]
605     self~uploadData(dbTable, serviceNowTable)
606   end

```

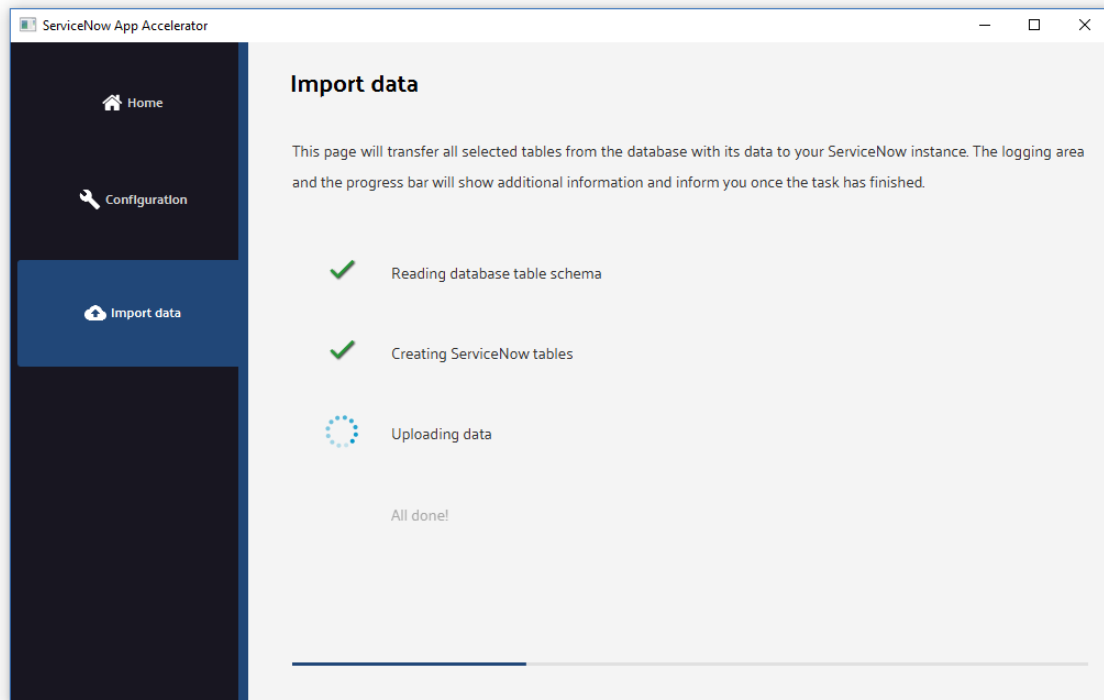



Figure 4.12.: Uploading started

```

607 return
608 syntax:
609     say ppCondition2(condition("object"))
610     raise propagate

```

Listing 4.42.: ImportTask's "call" Method

The method *calculateTotalApiCalls* initiates the *totalApiCalls* value by counting all selected tables in line 649. After that, it loops over all table names and creates an SQL query that returns the total sum of items. This query is executed with the *~executeQuery()* method of *databaseStatement*.

Using the *getInt* method of the *ResultSet* object with the ID 1, the application retrieves the row count of this specific table. This procedure is repeated for every table, each increasing the return value *totalApiCalls*. Of course, this approach does not take into account the complexity of the data or the amount of table columns, but it calculates a basic estimate for the progress bar. This will definitely come in handy if more than 10,000 or even 100,000 data sets are imported to ServiceNow using this tool.

```

641 /**
642  * Calculates the number of all API calls that will be made
643  *
644  * @return totalApiCalls - The number of all tables and its rows
645  */
646 ::method calculateTotalApiCalls
647     expose databaseTables databaseStatement totalApiCalls currentProgress
648     currentProgress = 0
649     totalApiCalls = databaseTables~items      -- one api call for each table
650     jint1 = .java.lang.Integer~new(1)      -- convert the Rexx String "1" to a
        java.lang.Integer 1

```

```

651 loop tableName over databaseTables
652   sql = "SELECT COUNT(1) FROM" tableName
653   resultSet = databaseStatement~executeQuery(sql)
654   resultSet~next
655   rowCount = resultSet~getInt(jInt1)
656   totalApiCalls += rowCount           -- one api call for each row
657 end
658 return totalApiCalls

```

Listing 4.43.: ImportTask's "calculateTotalApiCalls" Method

The *log* method simply writes the provided message to the output string. The *logAndProgress* method basically does the same, but also invokes the *progress* method, which increments the *currentProgress* and calculates a value between 0 and 1 of the *currentProgress* divided by *totalApiCalls*. The variable *finished* is set to *.true*, if the current progress is as high as *totalApiCalls*, in which case the last step 4 "All done" is unlocked.

Since this method operates in its own background thread, no GUI related operations can be performed. However the *.FxGUIThread* class is supplied by BSF4ooRexx that allows to update GUI controls by invoking its *~runLater()* method with the following arguments:

1. The reference to a GUI control
2. The name of the method that one wants to invoke
3. "A" if the forth argument is an array, otherwise "I"
4. A list of arguments

Using this class, line 714 sets the progress value of the progress bar to the previously calculated variable *progressTo*. If this value reaches 1, line 716 unlocks the last step and therefore concludes the import task.

```

686 /**
687  * Writes the provided message to the output stream
688  *
689  * @param message - the message without a line break in the beginning/at the end
690  */
691 ::method log
692   use arg message
693   say message
694
695 /**
696  * Forwards the message to the `log` method and invokes `progress`
697  *
698  * @param message - The message that is to be logged
699  */
700 ::method logAndProgress
701   use arg message
702   self~~log(message) -
703     ~~progress
704
705 /**
706  * Progresses by one API call and calculates the new progress
707  * Unlocks the last step if task has finished
708  */
709 ::method progress
710   expose totalApiCalls currentProgress importProgressbar
711   currentProgress += 1
712   progressTo = currentProgress / totalApiCalls
713   finished = progressTo = 1
714   .FxGUIThread~runLater(importProgressbar, "setProgress", "I", progressTo)

```

```

715   if finished then
716     self~unlockNextStep    -- unlock last step, show "All done!" message

```

Listing 4.44.: ImportTask's "log", "logAndProgress" and "progress" Methods

The *unlockNextStep* method creates references to the *currentPane* and *nextPane* by fetching the *fx:id* properties of the *steps* array and incrementing the *step* variable in lines 619, 625 and 626. Each pane of a step consists of a check mark if the step was already completed, a loading symbol for the current step or no graphic at all by default.

The instructions in line 623 and 624 use the *.FxGUIThread* class to first remove the loading symbol of the current pane, and then add a check symbol showing the user that this step is finished. In lines 630 to 632 the application shows the loading symbol for the next step and sets its opacity to 100 percent. The FXML source code in A.2 shows, that each step has in fact a hidden loading symbol and the labels an opacity of only 40 percent, except for the first step. If the last step was reached, the *if* condition in line 627 will simply skip the next pane treatment expectantly.

```

612  /**
613   * Replaces the loading symbol in the current pane with a tick mark
614   * Displays the loading symbol in the next pane
615   * Increases the font pacity to 100 percent in the label of the next pane
616   */
617  :method unlockNextStep
618    expose steps step
619    idOfPane = steps~at(step)~upper
620    currentPane = .my.app~SAA.fxml[idOfPane]
621    checkSymbol = .bsf~new("de.jensd.fx.glyphs.materialicons.MaterialIconView",
        .MaterialIcon~CHECK) ~~setY(30)
622    paneItems = currentPane~getChildren
623    .FxGUIThread~runLater(paneItems, "remove", "I", 0)
624    .FxGUIThread~runLater(paneItems, "add", "I", checkSymbol)
625    step += 1
626    idOfPane = steps~at(step)~upper
627    if idOfPane <> .nil then do
628      nextPane = .my.app~SAA.fxml[idOfPane]
629      loadingSymbol = nextPane~getChildren~get(0)
630      .FxGUIThread~runLater(loadingSymbol, "setVisible", "I", .true)
631      stepDescriptionLabel = nextPane~getParent~getChildren~get(1)
632      .FxGUIThread~runLater(stepDescriptionLabel, "setOpacity", "I", 1.0)
633  end

```

Listing 4.45.: ImportTask's "unlockNextStep" Method

After concluding the first step, the application calls the *~createServiceNowTable()* method for each database table name. This second step is labelled "Creating ServiceNow tables" and starts by invoking the *~getTableSchema()* method in line 782 with the table name as argument to fetch information about the database table, whose implementation is discussed in listing 4.48 on page 71. After creating a JSON encoded string with the table information, the application assembles an http request to the custom REST API in lines 787 to 790. It consists of both a query parameter and a body. The request is intentionally handled synchronously to guarantee that all tables are created before uploading data to prevent information loss.

The following lines starting from line 792 fetch the response body and its status code to determine if the table creation was successful. In case a status code different than 200 returns, the entire response is printed out using the *logAndProgress()* method. Otherwise,

the REST API call was successful and the application extracts the name of the newly created ServiceNow table alongside the number of columns.

```

773 /**
774  * Converts the database table and uploads its structure to ServiceNow
775  *
776  * @param tableName - the name of the database table
777  * @return serviceNowTable - the adjusted name of the created ServiceNow table
778  */
779 :method createServiceNowTable
780   expose serviceNowURL jsonEngine
781   use arg tableName
782   tableSchema = self~getTableSchema(tableName)
783   /* convert to json without unnecessary line breaks and white spaces */
784   jsonString = jsonEngine~toMinifiedJson(tableSchema)
785   body = .bsf~new("com.mashape.unirest.http.JsonNode", jsonString)
786   API.URL = serviceNowURL || "/api/x_146620_serviceno/create_table/" ||
       tableName
787   httpResponse = .Unirest~put(API.URL) -
788       ~header("Content-type", "application/json") -
789       ~body(body) -
790       ~asJson -- synchronous call for table creation
791   /* convert HttpResponse to a json encoded string, create a Rexx object,
       extract information */
792   jsonResponse = httpResponse~getBody~toString
793   statusCode = httpResponse~getStatus
794   resultObj = jsonEngine~fromJson(jsonResponse)
795   /* no appropriate content returned so output the json response as error
       message */
796   if statusCode <> 200 then do -- custom status code
797     self~logAndProgress("[Error]" jsonResponse)
798     return .nil
799   end
800   else do
801     serviceNowTable = resultObj["result"]["Name"]
802     columnCount = resultObj["result"]["Columns"]
803     message = "Table" pp(tableName) "created as" pp(serviceNowTable) "with"
       columnCount "columns"
804   end
805   self~logAndProgress(message)
806   return serviceNowTable

```

Listing 4.46.: ImportTask's "createServiceNowTable" Method

At this point, all table structures were transferred to the ServiceNow cloud. The next step is to also transfer the data from the database to the corresponding ServiceNow table using existing APIs. This step is initiated during the for-loop in lines 602 to 606 in listing 4.42 on page 65, which calls the *uploadData()* method for every selected table. It requires the names of both the database and ServiceNow table as parameter and makes use of the class variables for the ServiceNow instance URL, the json engine and the database statement.

Line 824 creates a Rexx proxy class of a new instance of the *.DataInsertionCallback* class whose definition starts in line 851, therefore extending the Java *Callback* interface. This will be used for fetching callback responses for every asynchronous Unirest request. The variable *API.URL* consists of the ServiceNow instance url, a fixed string "api/now/table" and the name of the ServiceNow table. Every instance has a set of predefined REST APIs for manipulating an existing table - this one inserts a new data set. [30]

In lines 829 and 830, the application executes an SQL statement to fetch all data from the table. The following for-loop iterates over all rows and creates a new Rexx object of the

`.stringTable` class holding the value of each column. This object is then formatted to a minimal JSON string using the json utility, forming the body of the request.

If the task was not marked as cancelled, the Unirest object creates a POST request to the priorly defined API URL with *application/json* as content type, a body and the Rexx proxy class as callback. This call is done asynchronously, so that many calls can happen at the same time. When executed, all the requests are likely to be sent using this technique even before the first response from the server arrives.

The *DataInsertionCallback* class implements the methods *completed* and *failed*. In both cases it progresses the progress bar using the *progress* or *logAndProgress* method of the *importTask* class. In case the request failed, a detailed exception report is additionally printed out.

```

814 /**
815  * Iterates over all rows of the database table, extracting all the data and
      sending it to the ServiceNow instance
816  *
817  * @param tableName - the name of the database table
818  * @param serviceNowTable - the name of the corresponding ServiceNow table (came
      with REST API table creation response)
819  */
820 ::method uploadData
821   expose serviceNowURL jsonEngine databaseStatement
822   use strict arg tableName, serviceNowTable
823   signal on syntax
824   rexxProxy = bsfCreateRexxProxy(.DataInsertionCallback~new,, .Callback)
825   API.URL = serviceNowURL || "/api/now/table/" || serviceNowTable
826   /* get table schema for column name */
827   columns = self~getTableSchema(tableName)
828   /* extract data from this table */
829   sql = "SELECT * FROM" tableName
830   resultSet = databaseStatement~executeQuery(sql)
831   do while resultSet~next
832     rxObj = .stringTable~new
833     loop column over columns
834       rxObj[column~label~lower] = resultSet~getString(column~label)  -- save
      the value of the database cell in rxObj
835   end
836   jsonString = jsonEngine~toMinifiedJson(rxObj)
837   body = .bsf~new("com.mashape.unirest.http.JsonNode", jsonString)
838   if \.jTask~isCancelled then
839     .Unirest~post(API.URL) -
840       ~header("Content-type", "application/json") -
841       ~body(body) -
842       ~asJsonAsync(rexxProxy)
843   end
844   return
845   syntax:
846     say ppCondition2(condition("object"))
847     raise propagate
848
849
850
851 ::class DataInsertionCallback
852 /**
853  * The REST API call was successful and returns a response
854  * Examines the status code and displays the result to the user
855  *
856  * @param response - <code>com.mashape.unirest.http.HttpResponse</code>

```

```

857 */
858 ::method completed
859   use arg response
860   .my.app~importTask~progress
861
862
863 ::method failed
864   use arg exception
865   message = exception~getMessage
866   suppressed = exception~getSuppressed
867   cause = exception~getCause
868   .my.app~importTask~log("Upload failed")
869   .my.app~importTask~logAndProgress("Message:" message suppressed cause)

```

Listing 4.47.: ImportTask's "uploadData" Method

Both the *createServiceNowTable* and *uploadData* methods make use of the *getTableSchema* method in lines 782 and 827. This helper method uses the database's meta data to get a result set of column information about a specific table. It iterates over all of them and passes the result set, pointing to a specific column, to the *getColumnSchema* method, whose returned result is stored in an array called *fieldList* and finally returned.

The *getColumnSchema* method extracts various information about the column:

- **COLUMN_NAME**: the label
- **IS_NULLABLE**: if set to "NO", the data in this column cannot be NULL, which makes it mandatory
- **REMARKS**: custom meta data
- **TYPE_NAME**: the column type

The last bit of information can be one of the ones defines in lines 751 to 765. If it is not in that list, it will most likely be a string and handled as such. The saved value of the variable *type* has to be one of the predefined field types set by the ServiceNow Platform. It is worth noting that ServiceNow offers many more types that SQL databases do, like CHOICE or HTML. After determining the corresponding ServiceNow field type, it is saved in a *stringTable* object holding all information about the provided column. The primary property is always set to *.false*, because ServiceNow has its own unique indexing mechanism.

```

719 /**
720  * Extracts all relevant information about the given table using JDBC meta data
721  *
722  * @param tableName - the name of a table
723  * @return fieldList - a .stringTable with the naming convention based on a
724  *                   ServiceNow `fieldList`
725  */
726 ::method getTableSchema
727   expose databaseConnection
728   use arg tableName
729   fieldList = .array~new
730   metaData = databaseConnection~getMetaData
731   resultSet = metaData~getColumns(.nil, .nil, tableName, .nil)
732   loop while resultSet~next
733     columnInfo = self~getColumnSchema(resultSet)
734     fieldList~append(columnInfo)
735   end
736   return fieldList
737
738 /**
739  * Converts column information about a table to a .stringTable object

```

```

739 *
740 * @param resultSet - pointed to a specific table
741 * @return columnInfo - a stringTable object containing relevant information
742   about the table
743 */
744 ::method getColumnSchema
745   use arg resultSet
746   columnInfo = .stringTable~new
747   columnInfo~label = resultSet~getString("COLUMN_NAME")
748   nullable = resultSet~getString("IS_NULLABLE")
749   columnInfo~mandatory = (nullable = "NO")~?(.true, .false) -- if the field is
750     not nullable, it is mandatory
751   columnInfo~comments = resultSet~getString("REMARKS")
752   columnInfo~max_length = resultSet~getString("COLUMN_SIZE")
753   select case resultSet~getString("TYPE_NAME")
754     when "TINYINT", "SMALLINT", "MEDIUMINT", "INT" then
755       type = "Integer"
756     when "BIGINT" then
757       type = "Long"
758     when "DECIMAL", "FLOAT", "REAL", "DOUBLE" then
759       type = "Decimal"
760     when "BIT", "BOOLEAN" then
761       type = "Boolean"
762     when "DATE" then
763       type = "Date"
764     when "TIME" then
765       type = "Time"
766     when "DATETIME", "TIMESTAMP" then
767       type = "DateTime"
768     otherwise
769       type = "String"
770   end
771   columnInfo~type = type
772   columnInfo~primary = .false
773   return columnInfo

```

Listing 4.48.: ImportTask's "getTableSchema" and "getColumnSchema" Methods

When the last response arrives, the *progress* method realizes that the task is finished and unlocks the last step. This action displays the user a final message labelled "All done!". Now he can switch to his ServiceNow instance in the Browser to use the newly created table and examine, if the import task went well.

4.7.1. Create Table API Implementation

The REST API was created using ServiceNow Studio and its tools. It utilizes the following globally scoped Script Includes:

- **TableUtils** to fetch a valid ServiceNow table name
- **TableDescriptor** is a script that allows creating new tables
- **FieldDescriptor** is used to create fields, the equivalent of an SQL database's column

The script itself uses ECMAScript 5 but can also run in compatibility mode. A custom REST API script has to overwrite a *process()* function with the parameters *RESTAPIRequest* and *RESTAPIResponse*. Both are objects that hold various information about the request and response of the API call.

In line 5, the script uses the request object to extract the *tableLabel* parameter from the request path. If the value is empty or set to *{testApiExistence}*, it returns from this function

call without further processing. This will let the ServiceNow App Accelerator know, that this REST API script is installed properly and the basic authentication was successful.

The indefinite loop starting from line 11 tries to repeatedly adapt the table name to the Platform's needs. After sanitizing the table name and retrieving a valid identification in the global scope, line 18 checks if there is a table with that name. In case the sanitization returned with an error, the table name would result in *u_undefined*, where the prefix *u_* is generally used for global tables.

After agreeing on a table name, the script sets up a table creator using the *TableDescriptor* class. In lines 27 to 37, all data of the request body is extracted and iterated over. It consists of an array of objects, which hold information about all fields in the table. Every one of those information objects is then used as parameter to the *createField()* function defined in lines 51 to 67 and finally added to the table using the *addField()* method.

The *createField()* function fetches some predefined values from the parameter object, that were set using the ooRexx *getColumnSchema()* method from the Import class in Listing 4.48, such as LABEL, TYPE and MAX_LENGTH. Those are attached to a field object using the *setField()* method in line 62.

The described field object is initially created with the *__generateFieldDescriptor()* function defined in lines 68 to 83. The *FieldDescriptor* object works similar to the *TableDescriptor*. It holds a list of properties that describe a field. The *__generateFieldDescriptor()* function iterates over all properties in lines 78 to 80 and sets them to *null*, because the implementation of the constructor of the field descriptor falsely sets them to the string value "NULL", instead of the JavaScript *null* object. This incorrect implementation makes it impossible to create tables that use the *FieldDescriptor* class, because some values cannot be expressed as string, such as number identifiers.

The final lines of code set some default roles to the table, create it and return the final name of the table, its label and the number of created fields with the status code 200.

```

1  /*jshint esnext: true*/
2  (function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
3      /* fetch query parameters */
4      var tableLabel = request.pathParams.tableLabel;
5
6      /* return nothing if the label is empty or has a specific predefined value
7       for testing api existence purposes */
8      if(tableLabel.trim().length === 0 || tableLabel == "{testApiExistence}")
9          return;
10
11     /* sanitize the table name to serve all ServiceNow specifications */
12     while(true) {
13         var tableUtils = new global.TableUtils(tableLabel);
14         tableName = tableUtils.sanitizeTableName(tableLabel);
15         tableName = tableUtils.getValidTableName("global", tableName);
16         tableUtils.tableName = tableName;
17
18         /* check if the table already exists */
19         if(tableUtils.tableExists() ||
20             tableUtils.tableName.includes("u_undefined"))
21             tableLabel = tableName + "_new";
22         else
23             break;
24     }
25
26     /* set up the table creator */
27     var table = new global.TableDescriptor(tableName, tableLabel);

```



```

26
27  /* iterate over the fields array */
28  var requestBody = request.body;
29  var entry, field;
30  if(requestBody.data instanceof Array) {
31      while(requestBody.hasNext()){
32          /* create a field and add it to the table */
33          entry = requestBody.nextEntry();
34          field = createField(entry);
35          table.addField(field);
36      }
37  }
38
39  /* set default roles and finally create the table */
40  table.setRoles(tableName);
41  table.create();
42
43  return {
44      "Name": tableName,
45      "Label": tableLabel,
46      "Columns": table.fields.length
47  };
48
49
50
51  /**
52   * Extracts some predefined properties from the information object and saves
53   * them in the FieldDescriptor
54   * @param informationObj - An Object with information about the NAME, LABEL
55   * and TYPE of the Field
56   * @return field - A FieldDescriptor
57   */
58  function createField(informationObj) {
59      var name = informationObj.LABEL;
60      var field = _generateFieldDescriptor(name);
61      var fieldProperties = ["label", "type", "primary", "mandatory",
62          "comments", "max_length"];
63      fieldProperties.forEach(function(prop) {
64          field.setField(prop, informationObj[prop.toUpperCase()]); // rexx
65          translates all properties to upper case
66      }
67  );
68  };
69  return field;
70
71
72
73  /**
74   * As of 2017-09-02, there is a bug in the global Script Include
75   "FieldDescriptor", where the constructor calls "_setFieldNames" and
76   automatically defaults all values to the string <code>"NULL"</code> instead
77   of the nil object <code>null</code>, which ultimately breaks the
78   table.create() method.
79   * This function iterates over the object properties of the FieldDescriptor
80   and sets them all to nil
81   *
82   * @param fieldName - the name of the Field, accessible through
83   FieldDescriptor.fieldName, *warning*: the field name is unchangeable after
84   table.create()!
85   * @return field - the FieldDescriptor
86   */
87  function _generateFieldDescriptor(fieldName) {

```

```
76     var field = new global.FieldDescriptor(fieldName);
77     var fieldList = field.fieldList;
78     Object.keys(fieldList).forEach(function(key) {
79         fieldList[key] = null;
80     })
81     );
82     return field;
83 }
84
85 })(request, response);
```

Listing 4.49.: REST API JavaScript

4.8. Functional Testing

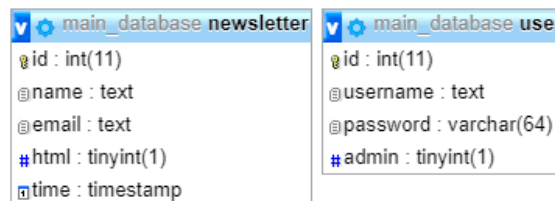
The ServiceNow App Accelerator was tested on Windows 10 with Java 8 Update 162 and Java 9.0.4, ooRexx Version 5.0.0 and BSF4ooRexx Version 600.20180317. The used MySQL database is of type MariaDB Version 10.1.16 with a phpMyAdmin front end user interface.

The two tables "Newsletter" and "User" were created for testing purposes to simulate a real world scenario. The table structure is shown in figure 4.13. Newsletter has the following five fields:

- **id**: the unique identifier mainly used for MySQL indexing. It is an integer with a default maximum length of 11 characters.
- **name**: a text that holds the name of the newsletter subscriber
- **email**: the E-Mail address of the subscriber
- **html**: a random boolean value to indicate if the subscriber wants to receive html formatted eMails. MySQL expresses booleans as tiny integers with length 1.
- **time**: the current timestamp

The user table uses similar field types:

- **id** of type integer
- **username** of type text
- **password** is a text whose maximum number of characters is limited to 64. This is due to the hashing algorithm SHA256 used. It hashes the user's password to a fixed size string of always 256 bits, which can be saved in hexadecimal characters of length 64 where each byte is represented as two hex characters.
- **admin** of type boolean



main_database newsletter	main_database user
id : int(11)	id : int(11)
name : text	username : text
email : text	password : varchar(64)
html : tinyint(1)	admin : tinyint(1)
time : timestamp	

Figure 4.13.: The table structure of "Newsletter" and "User"

Both tables were filled with dummy data generated through an ooRexx script, which will not be discussed in this master's thesis. This way, 100 random users and 1,500 newsletter subscribers were dumped to the database.

After saving the database credentials in the ServiceNow App Accelerator, one can select both tables to be extracted and uploaded to the respective ServiceNow instance. The upload took 12 seconds to complete and logged the following information:

```
REXXout>1600 records found in 2 tables
REXXout>Table [newsletter] created as [u_newsletter] with 5 columns
REXXout>Table [user] created as [u_user] with 4 columns
```

The upload speed may vary depending on the general network speed of the host and ServiceNow server availability. Every output that comes from Rexx side is automatically appended to the fixed string *REXXout>* to distinguish between Rexx and Java outputs.

The credentials file holding the login data is encrypted as stated in listing 4.22. It is not possible to reconstruct the plain text without the right master password, therefore its content is tenuous for unauthorized users without access to the decryption key:

```
gqHXz86BjNOE8fJO7npj7H4R3DNJHGTSag1VOzM5M5vVfc7F6wkcWA5U1XqhOmLrBT
Tuvf7CUJIWG7lk2n17BwoZWY+IcMqJC6qctvtXZTjM/rhTG/jqC9L7EL7CwSJ9/
oWGPwTile50fGtYYCHsCtQURJYmgbp5UJFx4RB5MntledtN+
nMamsjRTELKSSli66cm45BBjfMGLn/B50Xox1jrj6ptJcojJ+1
Ypk6USDStW09xO0IJdC1oozUK/2NfGffjZolIbFWned4/
GG3oHIcxHEboKeDk8r61cNZiQGSUFez5BjbXQnKYS3h8V+
gGU04YkPsKYyRfAkArktFPTc55kVMUrL5HLgnrOoqjIN/h95Try9glXz+
aBKMujfnP
```

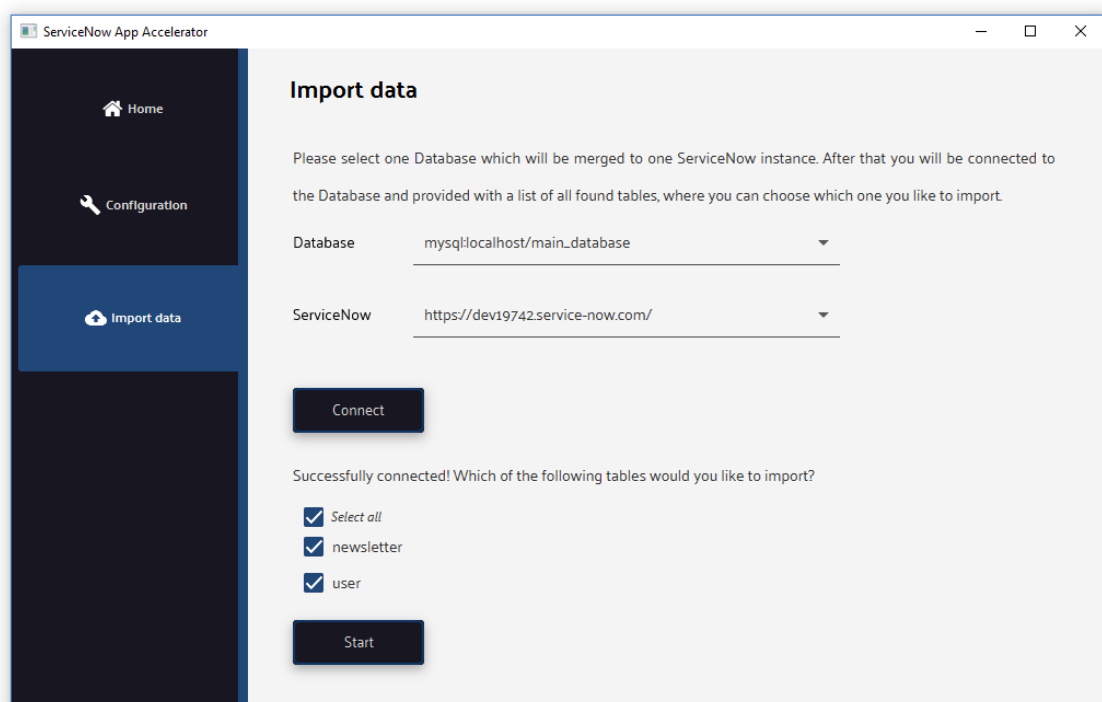


Figure 4.14.: Both tables can be selected in the last step of the Import data tab

A login in the ServiceNow instance reveals the newly created tables under *System definitions > Tables*. They have a few more additional meta fields than their database counterparts, because those were automatically filled in by ServiceNow. Figure 4.16 shows the exact structure of the users table. The fields "Created by", "Created", "Sys ID", "Updates", "Updated by" and "Updated" are necessary for the ServiceNow table to work properly.

☰	Tables	New	Go to	Name ▼	Search
🔍	All > Update name is not empty > Name >= u_user > or Name contains u_newsletter				
⚙️	🔍	☰ Label		☰ Name ▲	
☐	📘	<u>newsletter</u>			u_newsletter
☐	📘	<u>user</u>			u_user

Figure 4.15.: Filtered table list of tables that contain "newsletter" or "user" in their name

ColumnsControlsApplication Access

☰

Table Columns

New

Search

for text

▼

Search

◀◀◀

1

to 10 of 10

▶▶▶

⌵

🔍

Dictionary Entries

⚙️

🔍

☰ Column label

☰ Type

☰ Reference

☰ Max length

☰ Default value

☰ Display

📘	Admin	True/False		40	false
📘	ID	Integer		10	false
📘	Password	String		64	false
📘	Created by	String		40	false
📘	Created	Date/Time		40	false
📘	Sys ID	Sys ID (GUID)		32	false
📘	Updates	Integer		40	false
📘	Updated by	String		40	false
📘	Updated	Date/Time		40	false
📘	Username	String		65,535	false
+	Insert a new row...				

Figure 4.16.: The table structure of our "user" table

This table now offers many possibilities to use it. One of which is a list view as shown in figure 4.17, which ultimately displays all 100 imported users and their data.

≡ Admin	≡ ID	≡ Password	≡ Username
false	22	00b5a4d9aa03c430f055c8ad85497d78295eedfe...	burydiligent
false	68	cbc1495d75952a253ecfe157171e611a8ef5fd55...	bustbudding
false	70	00b5a4d9aa03c430f055c8ad85497d78295eedfe...	crunchbracket
false	24	856129cdc1aa0fa752d0e13a9542d2bcacfa7bb1...	moannational
false	72	51d504d3336e049271edc2bff3b7700c6a03dd1c...	deviantarteight
false	79	cbc1495d75952a253ecfe157171e611a8ef5fd55...	lumberingunripe
false	30	393ca85686975f624b00b95f088bd1a9669daad6...	relationcelest

Figure 4.17.: The list view of our "user" table

Also, one can create an empty form to insert new users to that list as shown in Figure 4.18. This view can be easily displayed in the service portal, other applications or in the front end using the AngularJS framework.

* Admin ☐

* Password

* ID

* Username

Submit

Figure 4.18.: An empty form of our "user" table

The exact same applies to the newsletter table. In the end, both tables were created with the right amount of columns, the right information about those columns and 1,600 data sets in total. The upload speed was very quick, although real world databases would hold a lot more data than used for this testing purpose. Nevertheless, since the data upload to the cloud is usually a one time thing when migrating the business infrastructure to the cloud, its importance really is negligible.

≡ Name	≡ Email	≡ HTML	≡ ID ▼	≡ Time
altenhein	mail@kurzpost.de	true	1,500	2017-09-12 19:32:16
truax	mail@ero-tube.org	true	1,499	2017-09-12 19:32:16
ethel	mail@SmellFear.com	true	1,498	2017-09-12 19:32:16
fitzhugh	mail@trash-mail.com	true	1,497	2017-09-12 19:32:16
rumble	mail@FudgeRub.com	true	1,496	2017-09-12 19:32:16
cutts	mail@mail4trash.com	true	1,495	2017-09-12 19:32:16
mcclure	mail@tempemail.net	true	1,494	2017-09-12 19:32:16
of orange	mail@watchfull.net	true	1,493	2017-09-12 19:32:16
vansickle	mail@willhackforfood.biz	true	1,492	2017-09-12 19:32:16

Figure 4.19.: The list view of our "newsletter" table

5. Related Work

Albeit the scientific resources on ServiceNow are scarce at the time of writing, the company itself provides a wide range of white papers, eBooks, case studies, analyst reports and webinars in the ServiceNow Resource Center at <https://www.servicenow.com/resources.html>. Those assets are available in multiple languages such as English, French, German and Spanish. Most of those resources are protected by a registration form that needs to be filled out in order to download all provided files. Apart from this, they are publicly available and regularly updated, which makes them a great source to understand what ServiceNow offers and how it works.

Open Object Rexx on the other hand is a very well researched topic. There are numerous resources available online, mainly because the original REXX language has been around for a long time, since 1979. The research paper "Resurrecting REXX, Introducing Object Rexx" by Rony G. Flatscher [26] depicts the main concepts of IBM's former programming language REXX and its rewritten object-oriented counterpart "ooRexx" using simple, yet powerful nutshell examples. Further, the research paper "The 2009 Edition of BSF4Rexx" [47] describes the progress of the Bean Scripting Framework for ooRexx which it made since its creation in 2001. To this date, the core of the framework stayed the same and is well described in the mentioned paper. On top of that, the recent article "JavaFX for ooRexx - Creating Powerful Portable GUIs for ooRexx" from 2017 [48] describes the creation of GUI applications with help of the JavaFX framework provided by Java and how BSF4ooRexx can be used to take advantage those classes to build advanced graphical user interfaces with less code and complexity.

An excellent reference for cryptography related tasks, such as choosing an appropriate encryption algorithm for the Jasypt library used in the ServiceNow App Accelerator is "Everyday Cryptography: Fundamental Principles and Applications" by Keith Martin [49]. This book provides a detailed overview of the history of cryptography, the current standards for encryption, signing, key derivation and many more, as well as real life cryptography applications.

The book "Essentials of Cloud Computing" by K. Chandrasekaran [3] is a good reference for exploring the cloud computing processes. It describes the importance of cloud computing, cloud deployment and service models, migration plans to the cloud and a new perspective of software development in the cloud. The provided detailed analysis of cloud computing allows for a deeper understanding of the ServiceNow cloud, and is therefore essential for this master's thesis.

Migration concepts are best described in "Legacy Information Systems: Issues and Directions" by J. Bisbal et al. [17] The discussed topics in this research paper show just how many options there are when moving away from a legacy information system that is currently in place to a modern solution using the latest architecture, tools and hardware. The research paper "Strategies and Methods for Cloud Migration" by J.-F. Zhao and J.-T. Zhou [50] additionally compares many different migration strategies for all cloud computing service models.

6. Improvements and Outlook

The ServiceNow App Accelerator in this version serves as a prototype for transferring SQL databases to the ServiceNow Cloud. Although the graphical user interface seems finished with the help of JFoenix's Material Design Implementation for JavaFX, there are improvements to make that might enhance user experience even more or add new features.

Most of the future user interface development improvements can be made in the "Manage credentials" tab under "Configuration", which features a JFXTreeTableView to only display the saved data. It is indeed possible to implement a double click handler in the respective table rows which could lead to changing the value of existing data cells. [51] Alternatively, one could insert an extra table column with buttons to change or delete a row. This is especially useful in case some login data change over time or the user makes a mistake when inserting new credentials. Some organizations enforce a cyclic password change policy, which could also affect the ServiceNow login.

The Import step currently imports both a table structure and its content. Certainly there are some use-cases where the user would only need to import one of which, for example to update the ServiceNow table with new data of the still active SQL database. However this version of the ServiceNow App Accelerator forces him to create a new table and re-import all the data again, instead of calculate a delta and update the data sets. This way an interactive synchronisation functionality can be offered.

This application was developed with the intention of submitting it to the ServiceNow app market as an open source project, so that other talented programmers can build on top of it. Obviously the easiest solution to this research topic is for ServiceNow to natively implement such SQL importing functionality to the platform themselves. Since even the Kingston release from 11th January 2018 did not include that and they do not seem to be working on it, the ServiceNow App Accelerator is a great tool for enterprises to migrate their existing applications to the cloud quickly.

Bibliography

- [1] (2018, feb) Wrangu gdpr data scanner press release. [Online]. Available: <http://wrangu.com/2017/07/24/wrangu-gdpr-data-scanner-press-release/> i
- [2] (2018, feb) Studie zu prognostiziertem umsatz von cloud computing. [Online]. Available: <https://de.statista.com/statistik/daten/studie/180537/umfrage/prognostizierter-umsatz-im-b2bund-b2c-segment-von-cloud-computing/> 1
- [3] K. Chandrasekaran, *Essentials of Cloud Computing*, 1st ed. Chapman & Hall/CRC, 2014. 2, 5, 18, 80
- [4] (2017, oct) Gartner prediction. [Online]. Available: <https://www.gartner.com/newsroom/id/3616417> 2
- [5] P. M. T. Grance, “The NIST Definition of Cloud Computing,” *Recommendations of the National Institute of Standards and Technology*, pp. 2–3, nov 2011. 2
- [6] (2017, oct) Cloud computing advantages. [Online]. Available: <https://www.lifewire.com/cloud-computing-explained-2373125> 2
- [7] (2017, oct) Servicenow financial reports. [Online]. Available: <http://www.businesswire.com/news/home/20171025006277/en/ServiceNow-Reports-Financial-Results-Quarter-2017> 3, 85
- [8] M. Fouquet, H. Niedermayer, and G. Carle, “Cloud computing for the masses,” in *Proceedings of the 1st ACM workshop on User-provided networking: challenges and opportunities*. ACM, 2009, pp. 31–36. 3
- [9] (2017, oct) Fundamental Course I. [Online]. Available: <https://www.youtube.com/watch?v=yDmGTEDp5g4> 4
- [10] (2018, jan) ServiceNow’s components. [Online]. Available: <https://www.t-systems.com/de/de/loesungen/cloud/saas-loesungen/soa/service-oriented-architecture-361370> 4, 85
- [11] (2017, oct) Platform architecture. [Online]. Available: <https://www.youtube.com/watch?v=CHIGvbqirQs> 6
- [12] “Building Business Apps at Lightspeed - Profiles in Digital Transformation,” ServiceNow, Tech. Rep., 2017. 7
- [13] K. Zurkus. (2017, oct) Defense in depth: Stop spending, start consolidating. [Online]. Available: <https://www.csoonline.com/article/3042601/security/defense-in-depth-stop-spending-start-consolidating.html> 7
- [14] Ponemon Institute, “2017 cost of data breach study,” IBM Security, Research Report, 2017. 7
- [15] “Streamlining Security Incident and Vulnerability Response,” ServiceNow Security Operations, Tech. Rep., 2017. 8
- [16] “ServiceNow Trusted Security Circles: Sharing Threat Intelligence to Curb Targeted Attacks,” ServiceNow Security Operations, Tech. Rep., 2017. 8, 85
- [17] J. Bisbal, D. Lawless, B. Wu, and J. Grimson, “Legacy information systems: issues and directions,” *IEEE Software*, vol. 16, no. 5, pp. 103–111, Sep 1999. 8, 9, 80, 85

-
- [18] K. Bennett, "Legacy systems: Coping with success," *IEEE Softw.*, vol. 12, no. 1, pp. 19–23, Jan. 1995. [Online]. Available: <https://doi.org/10.1109/52.363157> 8
 - [19] (2018, jan) Servicenow review. [Online]. Available: <https://reviews.financesonline.com/p/servicenow-express/> 10
 - [20] (2018, jan) Servicenow alternatives. [Online]. Available: <https://www.g2crowd.com/products/servicenow/competitors/alternatives> 10
 - [21] (2018, jan) Mavenlink Website. [Online]. Available: <https://www.mavenlink.com/pricing/project-management-software-comparison> 10
 - [22] (2018, jan) Redmine Website. [Online]. Available: <https://www.redmine.org/projects/redmine/wiki> 10
 - [23] (2018, jan) Servicenow's customers are 80 percent enterprises. [Online]. Available: <https://www.g2crowd.com/compare/jira-service-desk-vs-servicenow> 11
 - [24] J. K. Ousterhout, "Scripting: higher level programming for the 21st century," *Computer*, vol. 31, no. 3, pp. 23–30, Mar 1998. 12
 - [25] (2017, nov) The ieee 2017 top programming languages. [Online]. Available: <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages> 12
 - [26] R. G. Flatscher, "Resurrecting rexx, introducing object rexx," *Wirtschaftsuniversitaet Wien, Tech. Rep.*, may 2006. 12, 13, 80
 - [27] (2017, nov) Sourceforge oorexx download links. [Online]. Available: <https://sourceforge.net/projects/oorexx/files/oorexx/> 13
 - [28] R. G. Flatscher, "The 2009 edition of bsf4rexx," *WU Wien, Tech. Rep.*, 2009. 15
 - [29] Oracle. (2017, dec) Defining and starting a thread. [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html> 17, 86
 - [30] ServiceNow. (2017, dec) Table api. [Online]. Available: https://docs.servicenow.com/bundle/geneva-servicenow-platform/page/integrate/inbound_rest/concept/c_TableAPI.html 18, 68
 - [31] J. Bisbal, D. Lawless, B. Wu, J. Grimson, V. Wade, R. Richardson, and D. O'Sullivan, "A survey of research into legacy system migration," *Tech. Rep.*, 1997. 18, 19
 - [32] (2017, dec) Fontawesomefx at bitbucket. [Online]. Available: <https://bitbucket.org/Jerady/fontawesomefx> 23
 - [33] (2017, dec) Jasypt official website. [Online]. Available: <http://www.jasypt.org/> 23
 - [34] (2017, dec) Jfoenix official website. [Online]. Available: <http://www.jfoenix.com/> 23
 - [35] (2017, dec) Unirest official website. [Online]. Available: <http://unirest.io/> 23
 - [36] (2017, dec) Open object rexx: Reference (parse). [Online]. Available: <http://www.oorexx.org/docs/rexxref/x3647.htm> 25
 - [37] (2017, dec) Bouncy castle provider. [Online]. Available: <http://bouncycastle.org/wiki/display/JA1/Provider+Installation> 28
 - [38] (2017, dec) Unirest java shutdown. [Online]. Available: <http://unirest.io/java.html#user-content-exiting-an-application> 29
 - [39] (2017, dec) Jfoenix validators combobox issue. [Online]. Available: <https://github.com/jfoenixadmin/JFoenix/issues/130> 31
 - [40] (2017, dec) Javafx for oorexx(creating powerful portable guis). [Online]. Available: <http://www.rexxla.org/events/2017/presentations/AutoJava-BSF4ooRexx-07-JavaFx-201711.pdf> 36

-
- [41] (2017, dec) The java tutorials - establishing a connection. [Online]. Available: <https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html> 37
- [42] W. D. A. et al, *ooRexx Documentation 5.0.0.r11316 Open Object Rexx Reference*. Rexx Language Association, 2017. 46
- [43] (2018, jan) Jfxtreetableview at Github. [Online]. Available: <https://github.com/jfoenixadmin/JFoenix/blob/master/jfoenix/src/main/java/com/jfoenix/controls/JFXTreeTableView.java> 51
- [44] (2018, jan) Recursivetreeitem source at Github. [Online]. Available: <https://github.com/jfoenixadmin/JFoenix/blob/master/jfoenix/src/main/java/com/jfoenix/controls/RecursiveTreeItem.java> 54
- [45] (2018, jan) Celldatafeatures source. [Online]. Available: <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/TableColumn.CellDataFeatures.html> 54
- [46] (2018, jan) Connection source. [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html> 62
- [47] R. G. Flatscher, “The 2009 edition of bsf4rex,” *Business Informatics, Vienna University for Economics and Business Administration, Vienna*, 2009. 80
- [48] —, “Javafx for oorexx - creating powerful portable guis for oorexx,” *Business Informatics, Vienna University for Economics and Business Administration, Vienna*, 2017. 80
- [49] K. Martin, *Everyday Cryptography: Fundamental Principles and Applications*, ser. Everyday Cryptography: Fundamental Principles and Applications. OUP Oxford, 2012. [Online]. Available: https://books.google.at/books?id=5DZ_vv-gl4oC 80
- [50] J.-F. Zhao and J.-T. Zhou, “Strategies and methods for cloud migration,” *International Journal of Automation and Computing*, vol. 11, no. 2, pp. 143–152, Apr 2014. [Online]. Available: <https://doi.org/10.1007/s11633-014-0776-7> 80
- [51] (2018, feb) Demo source code of JFXTreeTableView. [Online]. Available: <https://github.com/jfoenixadmin/JFoenix/blob/master/demo/src/main/java/demos/components/TreeViewDemo.java> 81

List of Figures

2.1. ServiceNow market opportunities [7]	3
2.2. Components of the ServiceNow Platform [10]	4
2.3. Multi-instance architecture model	5
2.4. "Create Application File" Mask (Source: Screenshot)	7
2.5. Trusted Security Circles [16]	8
2.6. Solutions to legacy information systems [17, p.104]	9
4.1. Application Start Screen	34
4.2. Installation Manual	35
4.3. Configuration - Database	37
4.4. Master Password Prompt	42
4.5. Database Connection saved	45
4.6. Configuration - ServiceNow	46
4.7. Test ServiceNow returns an error	49
4.8. ServiceNow Connection tested and saved	50
4.9. Manage credentials	52
4.10. Application Start Screen	58
4.11. Table selection	63
4.12. Uploading started	65
4.13. The table structure of "Newsletter" and "User"	75
4.14. Both tables can be selected in the last step of the Import data tab	76
4.15. Filtered table list of tables that contain "newsletter" or "user" in their name	77
4.16. The table structure of our "user" table	77
4.17. The list view of our "user" table	78
4.18. An empty form of our "user" table	78
4.19. The list view of our "newsletter" table	79

Listings

3.1. Loops and Arrays in ooRexx	13
3.2. An advanced example demonstrating object-oriented functionality	14
3.3. Calling routines	15
3.4. BSF4ooRexx nutshell example	16
3.5. Starting a thread in Java [29]	17
4.1. License and Bootstrap	25
4.2. Internal Routine "Syntax"	25
4.3. Internal Routine "addJarsFromDirToClasspath"	26
4.4. Adjusting the Classpath environment variable	26
4.5. Static class import	27
4.6. JavaFX application start	29
4.7. The .RxApplication Class	29
4.8. CloseRequest Event Handler	30
4.9. Completing the Stage initialisation	31
4.10. The formValidator class	31
4.11. The manageComboboxChangeListener class	32
4.12. Tab pane change listener implementations	33
4.13. Completing the Stage Initialisation	33
4.14. Static Imports and "Home" Routines from SAA-controller.rxj	37
4.15. testDatabaseConnection Routine	38
4.16. validateForm Routine	39
4.17. setLoadingSymbolToButton Routine	39
4.18. dbConnect Routine	40
4.19. showPopup Routine	40
4.20. setCheckGraphicToButton Routine	41
4.21. saveDatabase Routine	41
4.22. saveRexxObjToJson and getRexxObjFromJson Routines	44
4.23. promptMasterPassword Routine	44
4.24. testServiceNow Routine	47
4.25. TestServiceNowCallback Class	48
4.26. saveServiceNow Routine	49
4.27. JFXTreeTableView	51
4.28. ServiceNow data model in ServiceNowAppAccelerator.rxj	52
4.29. Database data model	53
4.30. JFXTableManager Class	54
4.31. GetChildrenCallback Class	54
4.32. PropertyValueFactory Class	55
4.33. The JFXTableManager methods showAndFill, showTable and clearData . . .	56
4.34. JFXTableManager's method fillTable	57
4.35. prepareImport Routine	59
4.36. selectAllCheckboxes Routine	60

4.37. ImportTask Constructor	61
4.38. ImportTask's getBasicAuth Method	61
4.39. ImportTask's connectToDatabase Method	62
4.40. ImportTask's extractTables Method	62
4.41. startImport Routine	64
4.42. ImportTask's "call" Method	65
4.43. ImportTask's "calculateTotalApiCalls" Method	66
4.44. ImportTask's "log", "logAndProgress" and "progress" Methods	67
4.45. ImportTask's "unlockNextStep" Method	67
4.46. ImportTask's "createServiceNowTable" Method	68
4.47. ImportTask's "uploadData" Method	70
4.48. ImportTask's "getTableSchema" and "getColumnSchema" Methods	71
4.49. REST API JavaScript	74
A.1. SAA.css	91
A.2. SAA.fxml	104
A.3. JFXAlert.fxml	105

A. Appendix

```
1 .h1 {
2   -fx-font-family: "Palanquin Dark";
3   -fx-font-size: 22px;
4 }
5
6 .root {
7   -fx-font-size: 14px;
8 }
9
10 .root > .tab-pane > .tab-header-area > .tab-header-background {
11   -fx-opacity: 0;
12 }
13
14 .tabGraphic {
15   -glyph-size: 20px;
16 }
17
18 .root > .tab-pane > .tab-header-area {
19   -fx-font-family: "Palanquin Dark";
20   -fx-cursor: hand;
21   -fx-font-size: 16px;
22   -fx-background-color: #181621;
23   -fx-border-width: 0 0 9 0;
24   -fx-border-color: #214778;
25   -fx-padding: 22 0 0 6;
26 }
27
28 .tab-content-area {
29   -fx-font-family: Palanquin;
30 }
31
32 .root > .tab-pane
33 {
34   -fx-tab-min-width: 20px;
35   -fx-tab-min-height: 70px;
36 }
37 .tab {
38   -fx-focus-color: transparent;
39   -fx-focus-faint-color: transparent;
40   -fx-background-color: transparent;
41 }
42
43 .tab:selected {
44   -fx-background-color: #214778;
45   -fx-focus-color: transparent;
46   -fx-focus-faint-color: transparent;
47 }
48
49 .root > .tab-pane > .tab:hover > .tab-label {
50   -fx-text-fill: white;
51 }
52
```



```

53 .tab .tab-label {
54     -fx-alignment: CENTER;
55     -fx-text-fill: #d8d8d8;
56     -fx-font-size: 12px;
57     -fx-font-weight: bold;
58 }
59
60 .tab:selected .tab-label {
61     -fx-alignment: CENTER;
62     -fx-text-fill: white;
63 }
64
65
66 .tab-pane:focused > .tab-header-area > .headers-region > .tab:selected
    .focus-indicator {
67     -fx-focus-color: transparent;
68     -fx-faint-focus-color: transparent;
69 }
70
71
72 .tab *.tab-label {
73     -fx-rotate: 90;
74 }
75
76 .tab {
77     -fx-padding: 70px 5px;
78 }
79
80
81 .scroll-pane {
82     -fx-background-color: transparent;
83 }
84
85 .button {
86     -fx-padding: 5 22;
87     -fx-border-color: #113666;
88     -fx-border-width: 2;
89     -fx-border-radius: 5;
90     -fx-background-color: #181621;
91     -fx-font-size: 10pt;
92     -fx-text-fill: #d8d8d8;
93     -fx-background-insets: 0 0 0 0, 0, 1, 2;
94     -fx-cursor: hand;
95     -fx-min-width: 120px;
96     -fx-max-height: 20px;
97 }
98
99 .glyph-icon {
100     -fx-fill: white;
101 }
102
103 .info-button > .glyph-icon {
104     -fx-fill: inherit;
105 }
106
107 .hasGraphic {
108     -fx-padding: 5 22 5 0;
109 }
110
111 .button:hover {
112     -fx-text-fill: white;

```

```

113     -fx-background-color:#214778;
114 }
115
116 .info-button {
117     -fx-background-color:white;
118     -fx-text-fill:#333;
119     -fx-border-color:transparent;
120 }
121
122 .info-button:hover {
123     -fx-text-fill: inherit;
124     -fx-background-color: #f4f5f7;
125 }
126
127
128 .checkbox-box {
129     -fx-cursor:hand;
130     -fx-focus-color: transparent;
131     -fx-focus-faint-color: transparent;
132     -fx-background-insets: 0, 0, 1, 2;
133      jfx-checked-color: #214778;
134     -fx-padding:0 0 5px 0;
135 }
136
137
138 .jfx-progress-bar > .track, .jfx-progress-bar > .bar {
139     -fx-background-radius: 0;
140     -fx-background-insets: 0;
141 }
142
143 .jfx-progress-bar > .track {
144     -fx-background-color: #E0E0E0;
145 }
146
147 .jfx-progress-bar > .bar {
148     -fx-background-color: #214778;
149 }
150
151 .jfx-tab-pane .headers-region,
152 .jfx-tab-pane .tab-header-background {
153     -fx-background-color: transparent;
154     -fx-effect:null;
155 }
156
157 .jfx-tab-pane .tab-selected-line {
158     -fx-background-color:#214778;
159 }
160
161 .jfx-tab-pane .tab-header-area .jfx-ripler{
162      jfx-ripler-fill: #E0E0E0;
163 }
164
165 .jfx-tab-pane .tab *.tab-label {
166     -fx-rotate: 0;
167 }
168
169 .jfx-tab-pane .tab:selected {
170     -fx-text-fill: white;
171     -fx-background-color:transparent;
172 }
173 .jfx-tab-pane .tab:selected .tab-label {

```

```

174     -fx-text-fill: #214778;
175 }
176
177 .jfx-tab-pane {
178     -fx-padding: 0 0 0 0;
179     -fx-tab-min-width: 200px;
180     -fx-tab-min-height: 50px;
181     -fx-background-color: transparent;
182 }
183
184 .jfx-tab-pane .tab-header-area {
185     -fx-cursor: hand;
186 }
187
188 .jfx-tab-pane .tab-label {
189     -fx-text-fill: #181621;
190 }
191
192 .jfx-tab-pane .tab {
193     -fx-padding: 0;
194 }
195
196 .jfx-tab-pane .tab-header-area {
197     -fx-padding: 15px 0 0 40px;
198 }
199
200 .no-tab-header {
201     -fx-tab-max-height: 0 ;
202 }
203
204 .no-tab-header .tab-header-area {
205     visibility: hidden ;
206 }
207
208 .importProgressStep .glyph-icon {
209     -fx-fill: #289637;
210     -fx-effect: dropshadow(one-pass-box, rgba(0,0,0,0.8), 4, 0.0, 1, 1);
211     -glyph-size: 30;
212 }

```

Listing A.1.: SAA.css

```

1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <?import com.jfoenix.controls.JFXButton?>
4  <?import com.jfoenix.controls.JFXCheckBox?>
5  <?import com.jfoenix.controls.JFXComboBox?>
6  <?import com.jfoenix.controls.JFXPasswordField?>
7  <?import com.jfoenix.controls.JFXProgressBar?>
8  <?import com.jfoenix.controls.JFXTabPane?>
9  <?import com.jfoenix.controls.JFXTextField?>
10 <?import com.jfoenix.controls.JFXTreeTableColumn?>
11 <?import com.jfoenix.controls.JFXTreeTableView?>
12 <?import com.jfoenix.validation.RequiredFieldValidator?>
13 <?import com.jfoenix.validation.ValidationFacade?>
14 <?import de.jensd.fx.glyphs.fontawesome.FontAwesomeIconView?>
15 <?import de.jensd.fx.glyphs.materialicons.MaterialIconView?>
16 <?import java.lang.String?>
17 <?import javafx.collections.FXCollections?>
18 <?import javafx.geometry.Insets?>

```

```

19 <?import javafx.scene.control.Hyperlink?>
20 <?import javafx.scene.control.Label?>
21 <?import javafx.scene.control.ProgressIndicator?>
22 <?import javafx.scene.control.ScrollPane?>
23 <?import javafx.scene.control.Tab?>
24 <?import javafx.scene.control.TabPane?>
25 <?import javafx.scene.control.TreeTableView?>
26 <?import javafx.scene.layout.AnchorPane?>
27 <?import javafx.scene.layout.HBox?>
28 <?import javafx.scene.layout.Pane?>
29 <?import javafx.scene.layout.VBox?>
30 <?import javafx.scene.text.Font?>
31 <?import javafx.scene.text.Text?>
32 <?import javafx.scene.text.TextFlow?>
33 <?language rexx?>
34
35 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="50.0"
    minWidth="-Infinity" prefHeight="600.0" prefWidth="1000.0"
    stylesheets="@SAA.css" xmlns="http://javafx.com/javafx/8.0.102"
    xmlns:fx="http://javafx.com/fxml/1">
36 <!-- the controller file holds all onAction target routines of this fxml file
    -->
37 <fx:script source="../ooRexx/SAA-controller.rxj" />
38 <children>
39 <TabPane fx:id="mainTabPane" maxWidth="200.0" minHeight="50.0"
    minWidth="50.0" prefHeight="400.0" prefWidth="600.0" rotateGraphic="true"
    side="LEFT" tabClosingPolicy="UNAVAILABLE" AnchorPane.bottomAnchor="0.0"
    AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
    AnchorPane.topAnchor="0.0">
40 <tabs>
41 <Tab text="Home">
42 <graphic>
43 <FontAwesomeIconView glyphName="HOME" styleClass="tabGraphic" />
44 </graphic>
45 <content>
46 <ScrollPane>
47 <content>
48 <VBox prefHeight="371.0" prefWidth="690.0">
49 <children>
50 <Text fontSmoothingType="LCD" strokeType="OUTSIDE"
    strokeWidth="0.0" styleClass="h1" text="Welcome to the
    ServiceNow App Accelerator!">
51 <VBox.margin>
52 <Insets bottom="20.0" />
53 </VBox.margin>
54 <font>
55 <Font size="22.0" />
56 </font>
57 </Text>
58 <Label lineSpacing="0.0" text="This ooRexx based application
    will help you to extract local databases into your
    ServiceNow instance while avoiding unnecessary
    complexity. Simply follow the instructions by connecting
    to the database and your ServiceNow instance and start
    uploading. Be assured that all data will be treated in
    confidence and not saved or shared in any way
    persistantly." textAlignment="JUSTIFY" wrapText="true" />
59 <TextFlow lineSpacing="0.0" textAlignment="JUSTIFY">
60 <children>
61 <Label text="If you are missing the necessary APIs,
    follow the instructions">

```

```

62         <padding>
63             <Insets top="2.0" />
64         </padding>
65     </Label>
66     <Hyperlink onAction="call openManual" text="in the
67         installation manual">
68         <opaqueInsets>
69             <Insets />
70         </opaqueInsets>
71     </Hyperlink>
72     <Text strokeType="OUTSIDE" strokeWidth="0.0" text="." />
73 </children>
74 <VBox.margin>
75     <Insets top="2.0" />
76 </VBox.margin>
77 </TextFlow>
78 <TextFlow lineSpacing="0.0" textAlignment="JUSTIFY">
79     <children>
80         <Label text="Should you encounter any unforeseen errors,
81             do not hesitate to contact">
82             <padding>
83                 <Insets top="2.0" />
84             </padding>
85         </Label>
86         <Hyperlink onAction="call openMail arg(arg())"
87             text="abaginski@live.de">
88             <opaqueInsets>
89                 <Insets />
90             </opaqueInsets>
91         </Hyperlink>
92         <Text strokeType="OUTSIDE" strokeWidth="0.0" text="." />
93     </children>
94 <VBox.margin>
95     <Insets top="2.0" />
96 </VBox.margin>
97 </TextFlow>
98 <JFXButton ButtonType="RAISED" defaultButton="true"
99     onAction="call goToConfiguration" text="Start">
100     <VBox.margin>
101         <Insets top="20.0" />
102     </VBox.margin>
103 </JFXButton>
104 </children>
105 <padding>
106     <Insets left="40.0" top="25.0" />
107 </padding>
108 </VBox>
109 </content>
110 </ScrollPane>
111 </content>
112 </Tab>
113 <Tab text="Configuration">
114     <graphic>
115         <MaterialIconView glyphName="BUILD" styleClass="tabGraphic" />
116     </graphic>
117     <content>
118         <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0"
119             prefWidth="200.0">
120             <children>

```

```

116 <JFXTabPane fx:id="configurationTabPane" prefHeight="500.0"
    prefWidth="783.0" tabClosingPolicy="UNAVAILABLE"
    AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
    AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
117 <tabs>
118 <Tab text="Database">
119 <content>
120 <ScrollPane>
121 <content>
122 <VBox prefWidth="700.0">
123 <children>
124 <Text fontSmoothingType="LCD"
    strokeType="OUTSIDE" strokeWidth="0.0"
    styleClass="h1" text="Database connection">
125 <VBox.margin>
126 <Insets bottom="20.0" />
127 </VBox.margin>
128 <font>
129 <Font size="22.0" />
130 </font>
131 </Text>
132 <Label text="Please provide the required
    database related information below and ensure
    that the ServiceNow App Accelerator is
    allowed to access the given database."
    wrapText="true" />
133 <HBox layoutX="10.0" layoutY="173.0"
    maxWidth="360.0">
134 <children>
135 <Text strokeType="OUTSIDE" strokeWidth="0.0"
    text="Database type">
136 <HBox.margin>
137 <Insets top="5.0" />
138 </HBox.margin>
139 </Text>
140 <Pane HBox.hgrow="ALWAYS" />
141 <JFXComboBox fx:id="databaseType"
    editable="true" prefWidth="250.0"
    promptText="please choose">
142 <items>
143 <FXCollections
    fx:factory="observableArrayList">
144 <String fx:value="mysql" />
145 <String fx:value="postgresql" />
146 <String fx:value="sqlserver" />
147 </FXCollections>
148 </items>
149 </JFXComboBox>
150 </children>
151 <VBox.margin>
152 <Insets top="10.0" />
153 </VBox.margin>
154 </HBox>
155 <HBox layoutX="50.0" layoutY="125.0"
    maxWidth="360.0">
156 <children>
157 <Text strokeType="OUTSIDE" strokeWidth="0.0"
    text="Host">
158 <HBox.margin>
159 <Insets top="5.0" />
160 </HBox.margin>

```

```

161         </Text>
162         <Pane HBox.hgrow="ALWAYS" />
163         <JFXTextField fx:id="databaseHost"
164             prefWidth="250.0" promptText="localhost
165             or IP address">
166             <validators>
167                 <RequiredFieldValidator />
168             </validators>
169         </JFXTextField>
170     </children>
171     <VBox.margin>
172         <Insets top="10.0" />
173     </VBox.margin>
174 </HBox>
175 <HBox fx:id="databasePortpane" maxWidth="360.0">
176     <children>
177         <Text strokeType="OUTSIDE" strokeWidth="0.0"
178             text="Port">
179             <HBox.margin>
180                 <Insets top="5.0" />
181             </HBox.margin></Text>
182         <Pane HBox.hgrow="ALWAYS" />
183         <JFXTextField fx:id="databasePort"
184             prefWidth="250.0" promptText="(optional)
185             Port number" />
186     </children>
187     <VBox.margin>
188         <Insets right="10.0" top="10.0" />
189     </VBox.margin>
190 </HBox>
191 <HBox maxWidth="360.0">
192     <children>
193         <Text strokeType="OUTSIDE" strokeWidth="0.0"
194             text="Database">
195             <HBox.margin>
196                 <Insets top="5.0" />
197             </HBox.margin>
198         </Text>
199         <Pane HBox.hgrow="ALWAYS" />
200         <JFXTextField fx:id="databaseDatabase"
201             prefWidth="250.0" promptText="enter
202             Database name">
203             <validators>
204                 <RequiredFieldValidator />
205             </validators>
206         </JFXTextField>
207     </children>
208     <VBox.margin>
209         <Insets right="10.0" top="10.0" />
210     </VBox.margin>
211 </HBox>
</HBox maxWidth="360.0">
    <children>
        <Text strokeType="OUTSIDE" strokeWidth="0.0"
            text="Username">
            <HBox.margin>
                <Insets top="5.0" />
            </HBox.margin>
        </Text>
        <Pane HBox.hgrow="ALWAYS" />

```

```

212         <JFXTextField fx:id="databaseUsername"
213             prefWidth="250.0" promptText="enter
214             Username">
215             <validators>
216                 <RequiredFieldValidator />
217             </validators>
218         </JFXTextField>
219     </children>
220     <VBox.margin>
221         <Insets right="10.0" top="10.0" />
222     </VBox.margin>
223 </HBox>
224 <HBox maxWidth="360.0">
225     <children>
226         <Text strokeType="OUTSIDE" strokeWidth="0.0"
227             text="Password">
228             <HBox.margin>
229                 <Insets top="5.0" />
230             </HBox.margin>
231         </Text>
232         <Pane HBox.hgrow="ALWAYS" />
233         <JFXPasswordField fx:id="databasePassword"
234             prefWidth="250.0" promptText="enter
235             Password" />
236     </children>
237     <VBox.margin>
238         <Insets right="10.0" top="10.0" />
239     </VBox.margin>
240 </HBox>
241 <HBox>
242     <VBox.margin>
243         <Insets top="30.0" />
244     </VBox.margin>
245     <children>
246         <JFXButton fx:id="saveDatabaseButton"
247             ButtonType="RAISED" defaultButton="true"
248             mnemonicParsing="false" onAction="call
249             saveDatabase arg(arg())" text="Save">
250             <graphic>
251                 <FontAwesomeIconView glyphName="SAVE" />
252             </graphic>
253         </JFXButton>
254         <JFXButton
255             fx:id="testDatabaseConnectionButton"
256             ButtonType="RAISED" onAction="call
257             testDatabaseConnection arg(arg())"
258             styleClass="info-button" text="Test
259             connection">
260             <HBox.margin>
261                 <Insets left="25.0" />
262             </HBox.margin>
263         </JFXButton>
264     </children>
265 </HBox>
266 </children>
267 <padding>
268     <Insets bottom="20.0" left="40.0" right="20.0"
269     top="15.0" />
270 </padding>
271 </VBox>
272 </content>

```



```

259         </ScrollPane>
260     </content>
261 </Tab>
262 <Tab text="ServiceNow">
263     <content>
264         <ScrollPane prefWidth="600.0">
265             <content>
266                 <VBox prefWidth="700.0">
267                     <children>
268                         <Text fontSmoothingType="LCD"
269                             strokeType="OUTSIDE" strokeWidth="0.0"
270                             styleClass="h1" text="ServiceNow instance">
271                             <font>
272                                 <Font size="22.0" />
273                             </font>
274                             <VBox.margin>
275                                 <Insets bottom="20.0" />
276                             </VBox.margin>
277                         </Text>
278                         <Label text="Please enter the URL of your
279                             ServiceNow instance as well as Username and
280                             Password for authentication purposes."
281                             wrapText="true" />
282                         <HBox maxWidth="360.0">
283                             <children>
284                                 <Text fontSmoothingType="LCD"
285                                     strokeType="OUTSIDE" strokeWidth="0.0"
286                                     text="Instance URL">
287                                 <HBox.margin>
288                                     <Insets top="5.0" />
289                                 </HBox.margin>
290                                 </Text>
291                                 <Pane HBox.hgrow="ALWAYS" />
292                                 <JFXTextField fx:id="servicenowURL"
293                                     prefWidth="250.0" promptText="e.g.
294                                     https://dev36664.service-now.com/">
295                                 <validators>
296                                     <RequiredFieldValidator />
297                                 </validators>
298                                 </JFXTextField>
299                             </children>
300                             <VBox.margin>
301                                 <Insets top="25.0" />
302                             </VBox.margin>
303                         </HBox>
304                         <HBox layoutX="30.0" layoutY="120.0"
305                             maxWidth="360.0">
306                             <children>
307                                 <Text fontSmoothingType="LCD"
308                                     strokeType="OUTSIDE" strokeWidth="0.0"
309                                     text="Username">
310                                 <HBox.margin>
311                                     <Insets top="5.0" />
312                                 </HBox.margin>
313                                 </Text>
314                                 <Pane HBox.hgrow="ALWAYS" />
315                                 <JFXTextField fx:id="servicenowUsername"
316                                     prefWidth="250.0" promptText="Username
317                                     for this instance">
318                                 <validators>
319                                     <RequiredFieldValidator />

```

```

306         </validators>
307     </JFXTextField>
308 </children>
309 <VBox.margin>
310     <Insets top="10.0" />
311 </VBox.margin>
312 </HBox>
313 <HBox layoutX="30.0" layoutY="145.0"
314     maxWidth="360.0">
315     <children>
316         <Text fontSmoothingType="LCD"
317             strokeType="OUTSIDE" strokeWidth="0.0"
318             text="Password">
319             <HBox.margin>
320                 <Insets top="5.0" />
321             </HBox.margin>
322         </Text>
323         <Pane HBox.hgrow="ALWAYS" />
324         <JFXPasswordField fx:id="servicenowPassword"
325             prefWidth="250.0" promptText="Password
326             for this instance">
327             <validators>
328                 <RequiredFieldValidator />
329             </validators>
330         </JFXPasswordField>
331 </children>
332 <VBox.margin>
333     <Insets top="10.0" />
334 </VBox.margin>
335 </HBox>
336 <HBox>
337     <children>
338         <JFXButton fx:id="saveServiceNowButton"
339             ButtonType="RAISED" defaultButton="true"
340             mnemonicParsing="false" onAction="call
341             saveServiceNow arg(arg())" text="Save">
342             <graphic>
343                 <FontAwesomeIconView glyphName="SAVE" />
344             </graphic>
345         </JFXButton>
346         <JFXButton fx:id="testServiceNowButton"
347             ButtonType="RAISED" onAction="call
348             testServiceNow arg(arg())"
349             styleClass="info-button" text="Test
350             connection">
351             <HBox.margin>
352                 <Insets left="25.0" />
353             </HBox.margin>
354         </JFXButton>
355     </children>
356     <VBox.margin>
357         <Insets top="30.0" />
358     </VBox.margin>
359 </HBox>
360 </children>
361 <padding>
362     <Insets bottom="20.0" left="40.0" right="20.0"
363     top="15.0" />
364 </padding>
365 </VBox>
366 </content>

```

```

354         </ScrollPane>
355     </content>
356 </Tab>
357 <Tab text="Manage">
358     <content>
359         <ScrollPane prefWidth="600.0">
360             <content>
361                 <VBox prefWidth="770.0">
362                     <children>
363                         <Text fontSmoothingType="LCD"
364                             strokeType="OUTSIDE" strokeWidth="0.0"
365                             styleClass="h1" text="Manage credentials">
366                             <font>
367                                 <Font size="22.0" />
368                             </font>
369                             <VBox.margin>
370                                 <Insets bottom="20.0" />
371                             </VBox.margin>
372                         </Text>
373                         <Label lineSpacing="6.0" text="Here you can
374                             manage your saved Database connections and
375                             ServiceNow instances." wrapText="true" />
376                         <JFXComboBox fx:id="manageCombobox" promptText="
377                             List Databases">
378                             <items>
379                                 <FXCollections
380                                     fx:factory="observableArrayList">
381                                     <String fx:value="List Databases" />
382                                     <String fx:value="List ServiceNow
383                                         instances" />
384                                 </FXCollections>
385                             </items>
386                             <VBox.margin>
387                                 <Insets top="10.0" />
388                             </VBox.margin>
389                         </JFXComboBox>
390                         <JFXTreeTableView
391                             fx:id="configurationManageDatabases"
392                             managed="true" visible="true"
393                             prefHeight="260.0">
394                             <VBox.margin>
395                                 <Insets top="15.0" />
396                             </VBox.margin>
397                         <columns>
398                             <JFXTreeTableColumn text="Type" prefWidth="100.0" />
399                             <JFXTreeTableColumn text="Host" prefWidth="100.0" />
400                             <JFXTreeTableColumn text="Port" prefWidth="60.0" />
401                             <JFXTreeTableColumn text="Name" />
402                             <JFXTreeTableColumn text="Username" />
403                             <JFXTreeTableColumn text="Password" />
404                         </columns>
405                         <columnResizePolicy>
406                             <TreeTableView fx:constant="CONSTRAINED_RESIZE_POLICY" />
407                         </columnResizePolicy>
408                     </JFXTreeTableView>
409                     <JFXTreeTableView
410                         fx:id="configurationManageServiceNowInstances"
411                         managed="false" prefHeight="260.0"
412                         visible="false">
413                         <columns>
414                             <JFXTreeTableColumn text="URL" />

```

```

402         <JFXTreeTableColumn text="Username" />
403         <JFXTreeTableColumn text="Password" />
404     </columns>
405     <columnResizePolicy>
406         <TreeTableView
407             fx:constant="CONSTRAINED_RESIZE_POLICY"
408             />
409     </columnResizePolicy>
410     </JFXTreeTableView>
411 </children>
412 <padding>
413     <Insets bottom="20.0" left="40.0" right="20.0"
414         top="15.0" />
415 </padding>
416 </VBox>
417 </content>
418 </ScrollPane>
419 </content>
420 </Tab>
421 </tabs>
422 </JFXTabPane>
423 </children>
424 </AnchorPane>
425 </content>
426 </Tab>
427 <Tab text="Import data">
428     <graphic>
429         <MaterialIconView glyphName="CLOUD_UPLOAD" styleClass="tabGraphic"
430             />
431     </graphic>
432     <content>
433         <AnchorPane>
434             <children>
435                 <Text fontSmoothingType="LCD" layoutX="38.0" layoutY="45.0"
436                     strokeType="OUTSIDE" strokeWidth="0.0" styleClass="h1"
437                     text="Import data">
438                     <font>
439                         <Font size="22.0" />
440                     </font>
441                 </Text>
442                 <JFXTabPane fx:id="importTabPane" styleClass="no-tab-header"
443                     AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
444                     AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="40.0">
445                     <tabs>
446                         <Tab>
447                             <content>
448                                 <ScrollPane>
449                                     <content>
450                                         <VBox fx:id="importStep1VBox" prefWidth="760.0">
451                                             <children>
452                                                 <Label lineSpacing="6.0" text="Please select one
453                                                     Database which will be merged to one
454                                                     ServiceNow instance. After that you will be
455                                                     connected to the Database and provided with a
456                                                     list of all found tables, where you can
457                                                     choose which one you like to import."
458                                                     textAlignment="JUSTIFY" wrapText="true" />
459                                         <HBox maxWidth="500.0">
460                                             <children>

```

```

448     <Text strokeType="OUTSIDE" strokeWidth="0.0"
449         text="Database">
450         <HBox.margin>
451             <Insets top="6.0" />
452         </HBox.margin>
453     </Text>
454     <Pane HBox.hgrow="ALWAYS" />
455     <ValidationFacade maxHeight="10.0">
456         <control>
457             <JFXComboBox fx:id="importDatabase"
458                 editable="false" prefWidth="390.0"
459                 promptText="please choose" />
460         </control>
461         <validators>
462             <RequiredFieldValidator />
463         </validators>
464     </ValidationFacade>
465 </children>
466 <VBox.margin>
467     <Insets top="10.0" />
468 </VBox.margin>
469 </HBox>
470 <HBox maxWidth="500.0">
471     <children>
472         <Text strokeType="OUTSIDE" strokeWidth="0.0"
473             text="ServiceNow">
474             <HBox.margin>
475                 <Insets top="6.0" />
476             </HBox.margin>
477         </Text>
478         <Pane HBox.hgrow="ALWAYS" />
479         <ValidationFacade maxHeight="10.0">
480             <control>
481                 <JFXComboBox fx:id="importServiceNow"
482                     editable="false" prefWidth="390.0"
483                     promptText="please choose" />
484             </control>
485             <validators>
486                 <RequiredFieldValidator />
487             </validators>
488         </ValidationFacade>
489     </children>
490 </HBox>
491 <JFXButton ButtonType="RAISED"
492     defaultButton="true" mnemonicParsing="false"
493     onAction="call prepareImport arg(arg())"
494     text="Connect">
495     <VBox.margin>
496         <Insets top="20.0" />
497     </VBox.margin>
498 </JFXButton>
499 <VBox fx:id="databaseAfterConnect"
500     visible="false">
501     <children>
502         <Label text="Successfully connected! Which
503             of the following tables would you like to
504             import?">
505             <VBox.margin>
506                 <Insets bottom="10.0" top="5.0" />
507             </VBox.margin>
508         </Label>

```

```

497         <VBox fx:id="databaseCheckboxPane">
498             <children>
499                 <JFXCheckBox fx:id="databaseSelectAll"
500                     mnemonicParsing="false"
501                     onAction="call selectAllCheckboxes
502                         arg(arg())" text="Select all">
503                     <font>
504                         <Font name="System Italic"
505                             size="12.0" />
506                     </font>
507                 </JFXCheckBox>
508             </children>
509             <VBox.margin>
510                 <Insets top="5.0" />
511             </VBox.margin>
512         </VBox>
513         <JFXButton ButtonType="RAISED"
514             mnemonicParsing="false" onAction="call
515                 startImport arg(arg())" text="Start">
516             <VBox.margin>
517                 <Insets top="15.0" />
518             </VBox.margin>
519         </JFXButton>
520     </children>
521     <opaqueInsets>
522         <Insets />
523     </opaqueInsets>
524     <VBox.margin>
525         <Insets top="20.0" />
526     </VBox.margin>
527 </VBox>
528 </content>
529 </ScrollPane>
530 </content>
531 </Tab>
532 <Tab>
533     <content>
534         <VBox prefHeight="453.0" prefWidth="768.0">
535             <children>
536                 <Label minHeight="-Infinity" text="This page will
537                     transfer all selected tables from the database
538                     with its data to your ServiceNow instance. The
539                     logging area and the progress bar will show
540                     additional information and inform you once the
541                     task has finished." textAlignment="JUSTIFY"
542                     wrapText="true">
543                     <VBox.margin>
544                         <Insets bottom="20.0" />
545                     </VBox.margin>
546                 </Label>
547                 <HBox styleClass="importProgressStep">
548                     <children>
549                         <Pane fx:id="readingDatabasePane"
550                             prefWidth="60.0">
551                             <children>

```

```

544         <ProgressIndicator prefHeight="34.0"
545             prefWidth="30.0" />
546     </children>
547     <HBox.margin>
548         <Insets left="30.0" />
549     </HBox.margin>
550 </Pane>
551 <Label text="Reading database table schema">
552     <HBox.margin>
553         <Insets top="5.0" />
554     </HBox.margin>
555 </Label>
556 </children>
557 <VBox.margin>
558     <Insets top="30.0" />
559 </VBox.margin>
560 </HBox>
561 <HBox layoutX="50.0" layoutY="100.0"
562     styleClass="importProgressStep">
563     <children>
564         <Pane fx:id="creatingTablesPane"
565             prefWidth="60.0">
566             <children>
567                 <ProgressIndicator prefHeight="34.0"
568                     prefWidth="30.0" visible="false" />
569             </children>
570             <HBox.margin>
571                 <Insets left="30.0" />
572             </HBox.margin>
573         </Pane>
574         <Label opacity="0.4" text="Creating ServiceNow
575             tables">
576             <HBox.margin>
577                 <Insets top="5.0" />
578             </HBox.margin>
579         </Label>
580     </children>
581     <VBox.margin>
582         <Insets top="40.0" />
583     </VBox.margin>
584 </HBox>
585 <HBox layoutX="50.0" layoutY="154.0"
586     styleClass="importProgressStep">
587     <children>
588         <Pane fx:id="uploadingDataPane"
589             prefWidth="60.0">
590             <children>
591                 <ProgressIndicator prefHeight="34.0"
592                     prefWidth="30.0" visible="false" />
593             </children>
594             <HBox.margin>
595                 <Insets left="30.0" />
596             </HBox.margin>
597         </Pane>
598         <Label opacity="0.4" text="Uploading data">
599             <HBox.margin>
600                 <Insets top="5.0" />
601             </HBox.margin>
602         </Label>
603     </children>
604     <VBox.margin>

```

```

597         <Insets top="40.0" />
598     </VBox.margin>
599 </HBox>
600 <HBox layoutX="50.0" layoutY="208.0"
        styleClass="importProgressStep">
601     <children>
602         <Pane fx:id="allDonePane" prefWidth="60.0">
603             <HBox.margin>
604                 <Insets left="30.0" />
605             </HBox.margin>
606             <children>
607                 <MaterialIconView glyphName="CHECK"
608                     visible="false" y="30.0" />
609             </children>
610         </Pane>
611         <Label opacity="0.4" text="All done!">
612             <HBox.margin>
613                 <Insets top="5.0" />
614             </HBox.margin>
615         </Label>
616     </children>
617     <VBox.margin>
618         <Insets top="40.0" />
619     </VBox.margin>
620 </HBox>
621 <JFXProgressBar fx:id="importProgressbar"
        prefWidth="787.0" progress="0.0"
        VBox.vgrow="ALWAYS">
622     <VBox.margin>
623         <Insets top="120.0" />
624     </VBox.margin>
625 </JFXProgressBar>
626 </children>
627 <padding>
628     <Insets bottom="20.0" left="40.0" right="20.0"
629         top="30.0" />
630 </padding>
631 </VBox>
632 </content>
633 </Tab>
634 </tabs>
635 </JFXTabPane>
636 </children>
637 </AnchorPane>
638 </content>
639 </Tab>
640 </tabs>
641 </TabPane>
642 </children>
643 <!-- comment: REXX program that stores all fx:id objects in .local~SAA.fxml
        directory -->
        <fx:script source="put_FXID_objects_into.my.app.rex" />
</AnchorPane>

```

Listing A.2.: SAA.fxml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import com.jfoenix.controls.JFXButton?>
4 <?import com.jfoenix.controls.JFXPasswordField?>

```



```

5 <?import com.jfoenix.validation.RequiredFieldValidator?>
6 <?import javafx.scene.control.Label?>
7 <?import javafx.scene.layout.AnchorPane?>
8 <?import javafx.scene.text.Font?>
9 <?language rexx?>
10
11 <AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
    minWidth="-Infinity" prefHeight="200.0" prefWidth="400.0"
    stylesheets="@SAA.css" xmlns="http://javafx.com/javafx/8.0.111"
    xmlns:fx="http://javafx.com/fxml/1">
12 <fx:script source="../../ooRexx/JFXAlert-controller.rxj" />
13 <children>
14 <Label layoutX="23.0" layoutY="14.0" text="Input Password">
15 <font>
16 <Font name="System Bold" size="16.0" />
17 </font>
18 </Label>
19 <JFXPasswordField fx:id="inputMasterPassword" layoutX="25.0"
    layoutY="92.0" prefHeight="25.0" prefWidth="350.0">
20 <validators>
21 <RequiredFieldValidator />
22 </validators>
23 </JFXPasswordField>
24 <JFXButton defaultButton="true" layoutX="255.0" layoutY="146.0" text="OK"
    onAction="call passwordEntered arg(arg())" />
25 <Label layoutX="25.0" layoutY="47.0" prefHeight="49.0" prefWidth="350.0"
    text="Please provide a master password to save and retrieve
    credentials." wrapText="true" />
26 </children>
27 </AnchorPane>

```

Listing A.3.: JFXAlert.fxml