Project Report/Documentation

Financial Verification System

Steven Miller
Joseph Kingston
Michael Kingston
Isaac Kingston

Table of Contents

Brief Motivation

Identity theft and fraud involving money have been two consistent threats that have lingered around us for some time. If someone's personal or bank information falls into the wrong hands, he/she could become a victim who could potentially lose a lot of money, or consistently pays bills for services or items that they aren't receiving. For instance, say an online company got a hold of a person's credit card information, and assume that this company begins charging this person monthly for services he/she will not receive. This victim would not be able to identify this fraud unless he/she consistently balanced his/her credit card bill, and verified that each transaction that appeared on the bill was valid. One way to solve this problem is for people to become increasingly paranoid about every transaction they make, and ensure that no "surprises" appear on their bank statements or credit card bills. This becomes increasingly tedious and time-consuming for those who make a lot of transactions, or even businesses who are obviously very interested in securing their financial information on a large scale.

Introduction

The intensions of this project were to provide a way for a person to obtain his/her personal bank information, compare this data with physical documents, and make this process be as automated as possible. These physical documents range between checks, receipts, or any document that can serve as proof of a transaction. In order to accomplish this, the following components or ideas needed to be implemented: a database to store all of this information, a scanner to scan the physical documents, OCR software to read the data from the physical documents, software programs to extract the bank information, and several other programs to make this system work together securely. With these intensions in mind, we have created a system which incorporates all of these features as well as others that have been added to provide more functionality for users.

Hardware and Software Design Overview

The sections that follow discuss the design and implementation of the hardware and software that is involved with this project. All of the system components can be categorized into one of the following below:

Software

- Bank Website: created by us to for legal purposes. Real banks do not want this type of program to access their information. This does, however, give us complete control of what is extracted from the "bank's" website.
- Bank Information Extraction Integration Program: used to extract the user's bank information from our bank website.
- Data Encryption Program: used to encrypt and decrypt sensitive or personal information like usernames and passwords that are required to access transaction information.
- Database User Interface: allows users to access and modify their transaction data.

- Database and VPN Software: Ingres is the database being used, and a mixture of ShrewSoft and Cisco Client is the VPN software that's used.
- OCR Software: reads the text from the fields of physical documents.
- Physical Document Extraction Integration Program: used to integrate all of the components of the Physical Document Extraction (PDE) component.

Hardware

- Iron Key: Used within the BIE component to securely store sensitive information.
- MagTek Excella Check Scanner: used exclusively to obtain the image of physical documents.
- Touch Screen: used to provide a user interface for the PDE component of this project.

Component Overview

This project consists of three main components: the Bank Information Extraction (BIE) component, the Physical Document Extraction (PDE) component, and the database. The diagram shown in Figure 1 consists of all of the main components that exist within this project as well as the integration requirements.

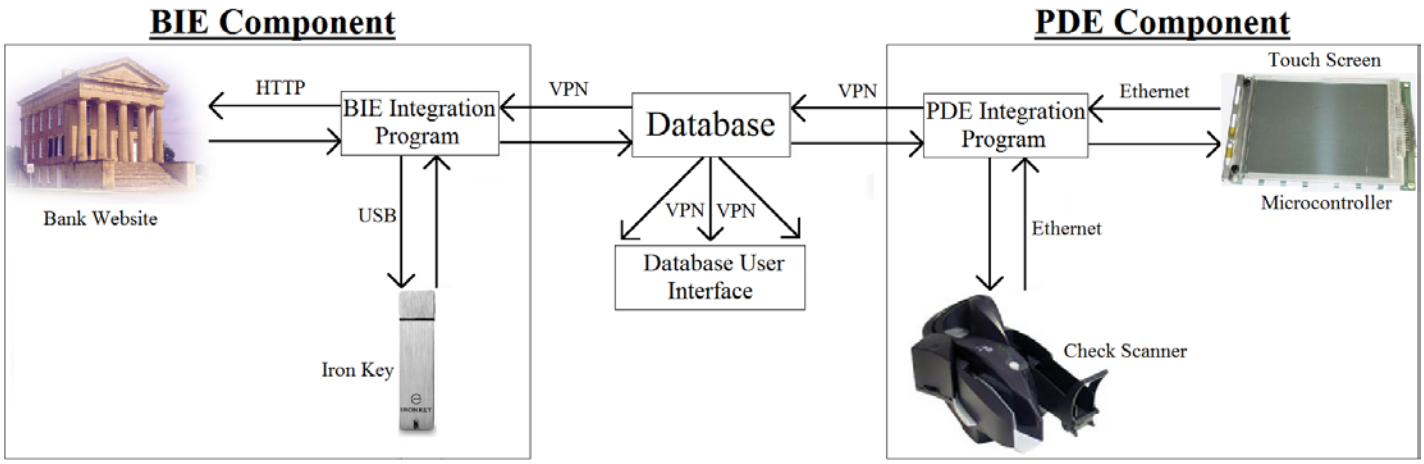


Figure 1. Project layout of all of the main components for this project.

The primary goal of the BIE component is to extract the bank information from the bank website using the HTTP protocol through the internet. All of the required sensitive information, such as usernames and passwords, is stored on the Iron Key where the integration program can easily extract this data. The data on the Iron Key determines what and whose bank information will be extracted.

The primary goal of the PDE component is to provide an interface in which a user can scan documents, have the system read the data, and send this information to the database. Note that the OCR software required in order to read the data is embedded within the integration program. More details will be explained in that portion of this document. The user will interface with the

PDE component directly through the touch screen. All of the required functionality that is needed from the user is controlled in this manner. The user will be able to see all of the results, and fix errors if needed before they send this information to the database.

BIE Component

The BIE portion is responsible for extracting a user's bank information from his/her bank's website. This consists of two major hardware components and two major software components. The hardware components include an internet computer and an external flash drive while the software components include an encryption program and a session save/replay program.

Hardware components:

Internet Computer:

The internet computer can be thought of as the central hub for the BIE portion. Every part ties in with this component. First, all traffic from the outside world, either in or out, will run through this computer. This happens through the session save/replay program which is housed on this component. Also, the encryption/decryption program is housed on this computer.

External Flash drive:

From the start it was decided to try not to store any secure information on the Internet Computer. This was a central design concept because anything on the internet computer would be more susceptible to security breaches whether from the internet or locally from the machine. To account for this idea we included an external flash drive. The External Flash drive can be thought of as the security vault of the BIE portion. It will house the encrypted login names and encrypted passwords of the user's bank website. It will also contain the key files to decrypt the user names and passwords, but will not have the program to decrypt it. These user names and passwords will be encrypted. The type of flash drive we have decided to use is the iron key personal. The iron key provides a large arsenal of security options, a few of these will be discussed below.

The iron key offers abroad and local protection of data. Even if the iron key is physically stolen, the information cannot be retrieved. It accomplishes this through hardware encryption, self destruct policies, and physical potting of the hardware. The iron key provides hardware AES, military grade encryption through its crypto-chip architecture. If the iron key detects even an attempt of a security breach, such as a brute force attack, it will internally self destruct. If someone tries to open up the iron key and connect through the hardware then they would have to break through hardened epoxy potting without destroying the circuit.

Software components:

Encryption/Decryption Program:

The Encryption/Decryption Program is used for the sole purpose of securing user names and passwords for bank websites. It was added as another security layer in case the iron key was compromised. The ED program was custom programmed and is based on the RSA encryption scheme and Java programming language. Attached in the appendix is an in-depth easy to follow users/technical manual. The ED program operates through the Java console user interface.

Bank Information Extraction Program:
This program is the most important part of the BIE component. It is the part that brings together the BIE and PDE components. It is the method by which the bank information is obtained. This by far created the greatest road block in the project.

Security/Encryption Program

Here is a high level view of the encryption process. You start out with a txt file containing the data you want to encrypt. Next you go on to the encoding process. The purpose of the encoding process is to map the characters to numbers so that they can be encrypted. A mathematical program doesn't understand how to encrypt a string such as "hello", so we need to map it to something like "12343139". After this then the encryption/decryption keys are created. Next, is the padding phase. This phase is not traditional padding where one is just appending bits on to blocks to make block sizes appropriate lengths. What it does instead is it creates a bunch of random data, far more than the actual data, and inserts garbage into the encoded messages. It was added as another level of security. After this it encrypts the data. There is many more steps in the process, but this is the basic process.

Another feature of the ED program is the ability to customize the encryption process at every step. For instance, for the encoding process, one can specify a permissible bit size, from 0-infinity, approximately, 1 out of 4 bit sizes are not appropriate. So while 11, 12, 13 are permissible, than 14 and 10 are not. The mappings are generated random each time. An example of what an encoding mapping of 511 bits would look is shown in Figure 1.

When generating a key file one can specify how large of the encryption components will be: The user can specify the modulus size, the encryption/decryption exponent size, the key size, etc… There is no limit on the key sizes other than the time it takes to create keys of those lengths. The program can create 2048 bit key sizes in a couple of minutes. Once you get larger than this then the time to create the keys grows exponentially. The padding step can also be customized; you can decide how much garbage you want to insert into the encrypted message. The program encrypts using ECB – electronic Code Book meaning it breaks apart the message to into chunks to be encrypted, but is not susceptible to open ECB attacks because of the padding step.

Another addition that makes the program more customizable is the automatic vs custom encryption process. One can have the program create everything for you automatically, one can have the program do portions of it automatically, or one can customize every single part of the process themselves. The program also packages up all of the key information into a nice compressed text file so that the same keys can be used for a particular session.

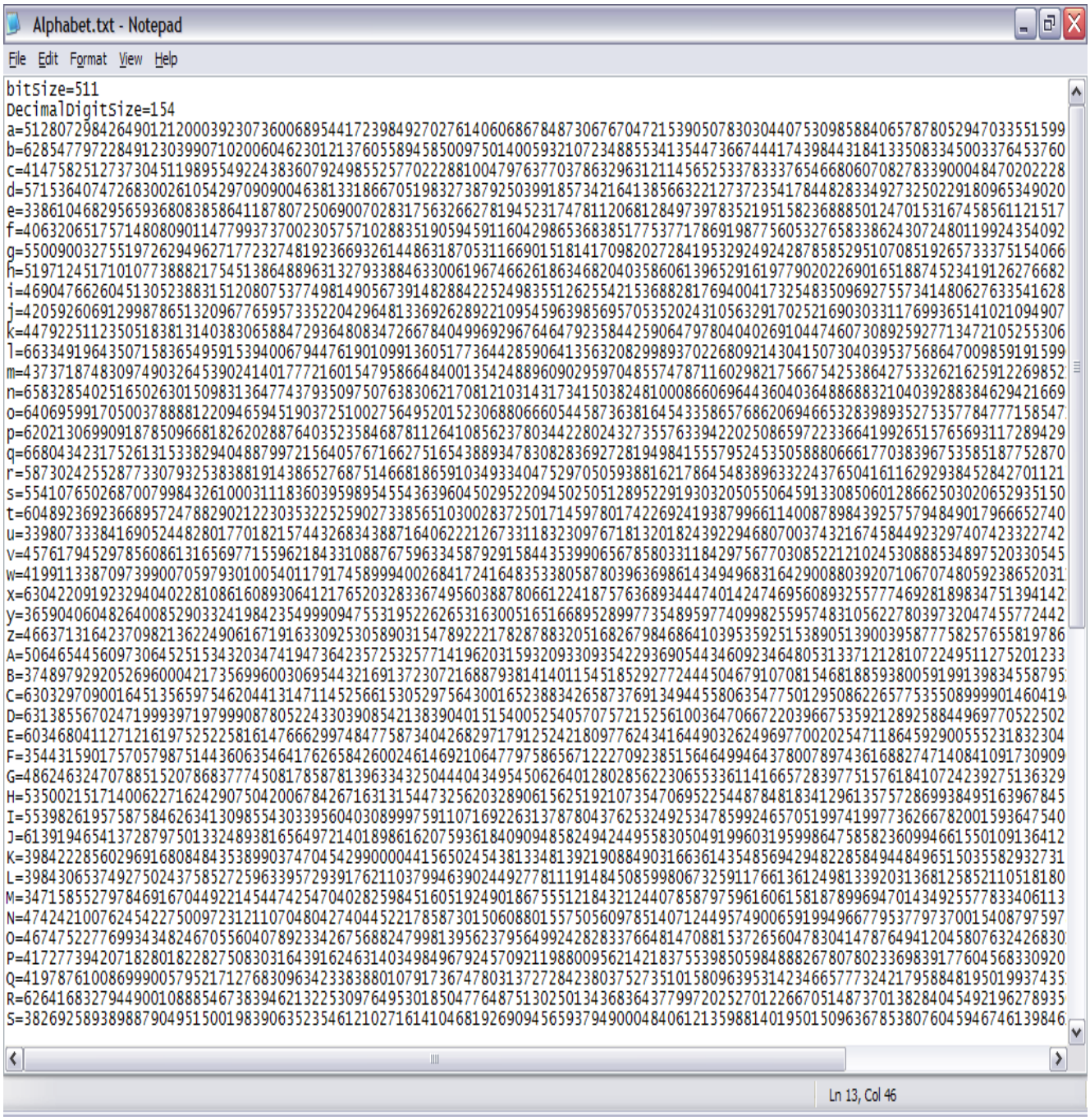


Figure 1. Sample alphabet mapping for the encryption program.

Bank Extraction Program

There were many ways researched and discovered to accomplish the goal of retrieving a user's bank information. A brief description along with the pros and cons of each method researched are listed below:

- OFX protocol – Open Financial Exchange

- Commercial software such as quickbooks
- Open source programs like gnu cash or KmyMoney
- Screen Scraping
- Session Playback

OFX protocol – Open Financial Exchange:
The Open Financial Exchange was collaborated effort by companies such as Intuit and Microsoft to streamline a standard of the transfer of financial information. It has been adopted and implemented into most of the larger financial organizations. With this protocol one could automatically connect to their bank and download their information in a nicely organized text/excel file. This by far would be the most efficient, the most cost effective, the most dynamic, it would provide the most control, and would be easy to maintain. The problem with this is cooperation from financial institutions. Despite the "Open" phrase in the name, the Open Financial Exchange is not open for particular implementations. So even though the entire OFX language is accessible, manuals are readily available, and bank clearance codes can be found online, then one has to get permission from the financial instution to open the gate for communications to their servers. Financial Instutions are very paranoid about letting single users have this access. It is a big security risk, in their eye's, and isn't worth the time and effort to them. They don't see any reason to grant this access to anyone else because there are already other pieces of commercial software that do this like quickbooks. Some institutions also have proprietary contracts with commercial software companies such as Quickbooks to where they won't let anyone else in unless they are using Quickbooks. Banks that we attempted to get setup in this area were Bank of Utah and Chase bank.

Commercial software:
Using existing software would be the easiest to implement of all options. It would also be the easiest to maintain because you wouldn't have to maintain it. It also provides many shortcomings. For one, it would eliminate a lot of the motivation for the project in the first place. It would get rid of the BIE portion almost entirely. Other cons are that you wouldn't have half as much control which could mean loss of functionality/customizability, privacy, and security. As far as privacy and security, then who knows who is looking at your information? It wouldn't be the most cost effective either. These pieces of software can range in the hundreds of dollars. One would also be subject to licensing fees and yearly dues.

Open source programs:
Implementing this would also be an easy option. This avenue would provide many of the same advantages and disadvantages as with using commercial software. It also comes with many more disadvantages. Many of these open source programs implement border-line illegal techniques to get their data. Many impersonate other commercial pieces of OFX software as to gain access to their information. Two that do this are Gnu Cash and

KmyMoney. They can't even package the bank downloader information in their standalone product because they will get in trouble. Rather they offer plug-ins that do the borderline illegal work. For the above two programs mentioned then there is a German site that you can pull the plug-ins from. One last interesting note to mention is while researching a forum on how to pull your data, then it was mentioned that you had to tell the banks you were using QuickBooks when setting up the connection with one of the open source programs that were looked at.

Screen Scraping:
This method would also be a fairly easy one to implement. The biggest problem with this would be maintenance of the program. If a bank changed even the slightest things on its website then one would have to adjust their program accordingly. This could be very time consuming especially if you are having to update multiple banks all the time. For this reason, this option was discarded.

Session Playback:
This is by far the hardest of the above mentioned methods to implement. This saves a users internet session and then replays it back. So for instance, if someone starts their browser in Google, navigates to their bank website, logs in, and extracts their bank information, then one can package this session into a nice file and replay it at anytime. The replay would go through the same things, it would start at Google, navigate to and through the bank, and download their bank info. This has a big advantage over screen scraping in that there is close to zero maintenance involved. If a bank changes their site and it affects navigation of a previous saved session then the user can go through and manually record a new session to be played back at anytime. Since the banks are the ones updating these files that can be downloaded then each time you replay the session then it will go through the same process, but give you the updated file.

How Session Playback works:
The Session Playback program consists of 3 crucial entities: A Firefox browser, a self-contained proxy server, and the destination website. The reason an actual browser such as Firefox is used is so that we wouldn't have to write a fully fledged html/http compliant program. To do this would be a tedious and difficult amount of work. With this in place then one can see what they are saving as it is being saved and can see the replay of was navigated through. The only work as far as the browser is concerned is to configure it to work with a proxy. This can be done by going into Tools→Options→Advanced→Settings and then clicking on Manual Proxy configuration. The destination website is just there to grab ones data from. The proxy is where real work takes place.

The Session Save/Replay program entities can be diagrammed at a high level as shown in Figure 2.

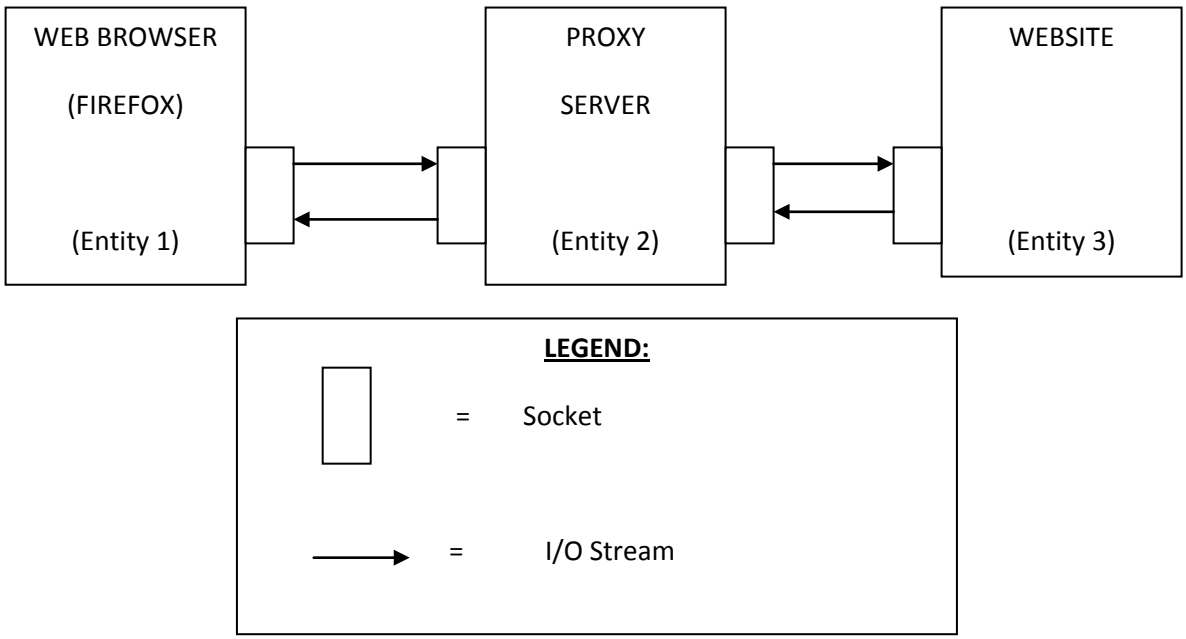


Figure 2. Simple diagram of the Session Save/Replay Program.

The Session Save/Replay program, from a programmer's perspective, can be broken up into 2 parts or concepts: Session Saving and Session Replaying. In the program itself, these two parts are broken up into two separate packages (so two separate programs) with a small bundle of classes pertaining to each. These two packages are V2Http and ReplayV2.

The classes that pertain to V2Http

- proxy
  - The central part of the program. (where main is called)
- clientProxyThread
  - A thread to handle everything coming into the proxy from the browser and everything going out of the proxy to the destination website.
- serverProxyThread
  - A thread to handle everything coming into the proxy from the destination website and everything going out of the proxy to the browser.
- Converter
  - Handles byte and string conversions.

The V2Http class works as follows: V2Http starts the program by calling the class proxy. Proxy initializes all of the connection setup and user permissions. From here it creates two sockets associated with the proxy entity; one to handle the I/O between the proxy and the browser and one to handle the I/O between the proxy and the website. After this it calls the clientProxyThread and the serverProxyThread respectively. These two threads are running simultaneously. It also, passes the created sockets into these threads. Once this is done it waits until the two created sockets are closed and then starts the process over.

The clientProxyThread works as follows: It creates I/O streams to the appropriate sockets. These streams consist of an input stream to the proxy entity from a local host connection (the browser), and an output stream to the website entity (based on a URL). It handles and manages all of the http traffic coming from the browser. This includes saving the information to file, checking and filtering the information for particular requests (i.e. http1.0 http1.1), and converting the information into an understandable format for a human being to read.

The serverProxyThread works as follows: It creates I/O streams to the appropriate sockets. These streams consist of an input stream to the proxy entity from a website (based on a URL), and an output stream to the browser (through local host). It handles and manages all of the http traffic coming from the website. This includes saving the information to file, checking and filtering the information for particular requests, and converting the information into an understandable format for a human being to read.

The classes that pertain to ReplayV2

- ReplayProxyV2
  - The central part of the program. (where main is called)
- ReplayclientProxyThread
  - A thread to handle everything coming into the proxy from the browser and everything going out of the proxy to the destination website.
- ReplayserverProxyThread
  - A thread to handle everything coming into the proxy from the destination website and everything going out of the proxy to the browser.
- Converter
  - Handles byte and string conversions.

ReplayV2 has a lot of the same functionality as V2Http, but also has many things that are unique to it. The timing issues are handled a little bit different and many of the subtle details that make the program work are different.

The ReplayProxyV2 class works as follows: ReplayV2 starts the program by calling the class ReplayProxyV2. This class initializes all of the connection setup and user permissions. It parses the saved data and stores appropriate information such as the saved URL's for future processing and the request information to replay back sessions. From here it creates two sockets associated with the proxy entity; one to handle the I/O between the proxy and the browser and one to handle the I/O between the proxy and the website. After this it calls the ReplayclientProxyThread and the ReplayserverProxyThread respectively. These two threads are running simultaneously. It also, passes the created sockets into these threads. Once this is done it waits until the two created sockets are closed and then starts the process over.

The ReplayclientProxyThread works as follows: It creates I/O streams to the appropriate sockets. These streams consist of an output stream to the website entity (based on a URL). One of the main differences between ReplayclientProxyThread and the clientProxyThread previously

discussed is that ReplayclientProxyThread is not recieiving information from the browser, but rather from the saved session from file.

The ReplayserverProxyThread works as follows: It creates I/O streams to the appropriate sockets. These streams consist of an input stream to the proxy entity from a website (based on a URL), and an output stream to the browser (through local host). It handles and manages all of the http traffic coming from the website. This includes saving the information to file, checking and filtering the information for particular requests, and converting the information into an understandable format for a human being to read.

GUI Interface:

To manage all of this by hand can be tedious and require a very extensive knowledge of the inner workings of the program. For this reason a user interface has been created. An example of the user-interface program is shown in Figure 3.

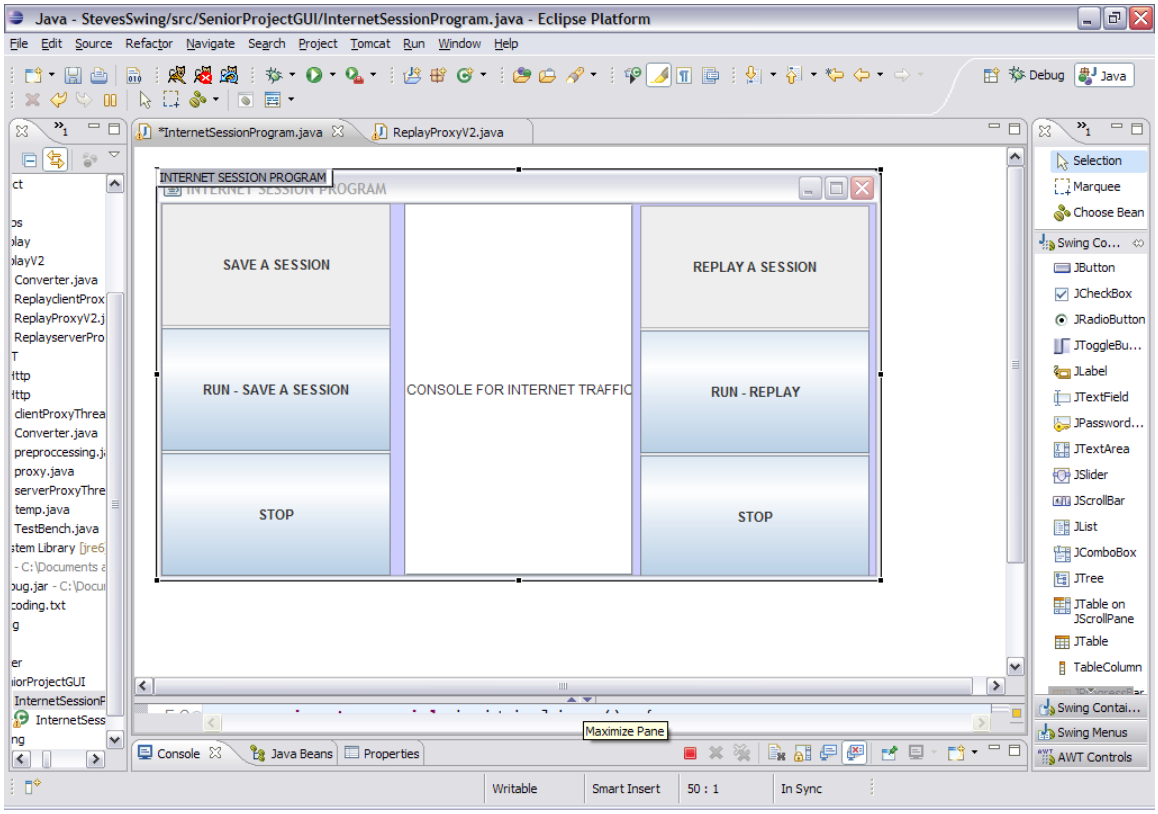


Figure 3. Example of the BIE Component GUI Interface.

In Figure 3, the Saved Session portion is on the left and The Replay Session is on the right. To save a session, just click on the appropriate "Run" button, and to stop the particular running program then click the appropriate "Stop" button. The incoming and outgoing internet traffic appears in the Text Area.

The program currently only supports http. This means that it will not work for anything that uses https. Most everything that deals with secure information, such as the banks, use https. To account for https in saving and replaying sessions would be a project within itself. If there was another semester available to work on it then it could be done, but there is not enough time. The thing that makes https hard is the randomness that is apart of the encryption process. Even though one could save the sessions, replaying them would not work in the programs current form. This is so because the website would recognize that the authentication/key information does not match up. The website would not recognize the requests. The fix for this is would be to not worry about saving the keys long term, but rather the request information. What one would do would be to automatically generate the session keys each time, do all of the encryption/decryption in the proxy, save the appropriate request and response information, and for the saved request/response data then encrypt and decrypt it using the automatically generated keys that are produced each time. To take care of the fact that our program doesn't supplement https, then we created a fake bank website to demonstrate our project through.

Database

An Ingres database is being used for this project. This database can only be accessed while connected to a VPN. The database can be seen below in Figure 4. Notice that transactions are tied to documents with a foreign key. Documents include checks and slips. The 'cardNumber' field in the document table holds the Payee's card number (for slips) and the MICR code for checks. Both the card number and MICR code can be used to map to the Payee if needed. The access table is used to store the user's login information. In the future it may be expanded to reference a customer's multiple accounts.

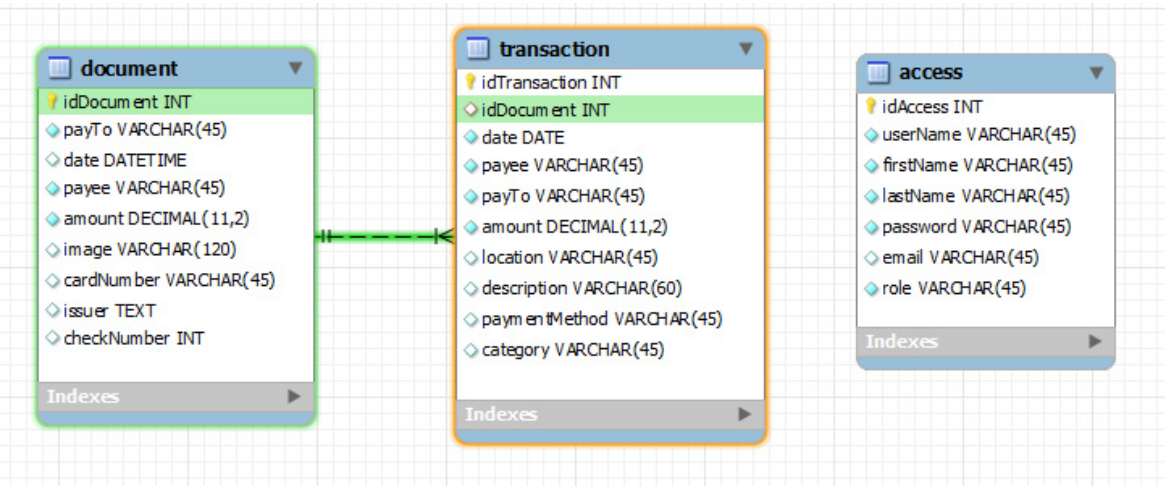


Figure 4: Database Schema

VeriFinance User Interface

VeriFinance is the application that facilitates the verification of bank transactions. It allows the user to match up each transaction with a scanned document. It presents graphs to summarize

financial information in a meaningful way. It also allows the user to modify the transaction table in order to correct an error.

In the VeriFinance application the user is greeted by the welcome screen seen in Figure 5, where they receive a brief explanation of this financial management system. The user can log in through the sidebar on the left or else create a new account if they wish. The design you see below was ported from a website theme created by Aquia Drupal. A download of this theme can be obtained from the Drupal community at http://drupal.org/project/acquia_marina

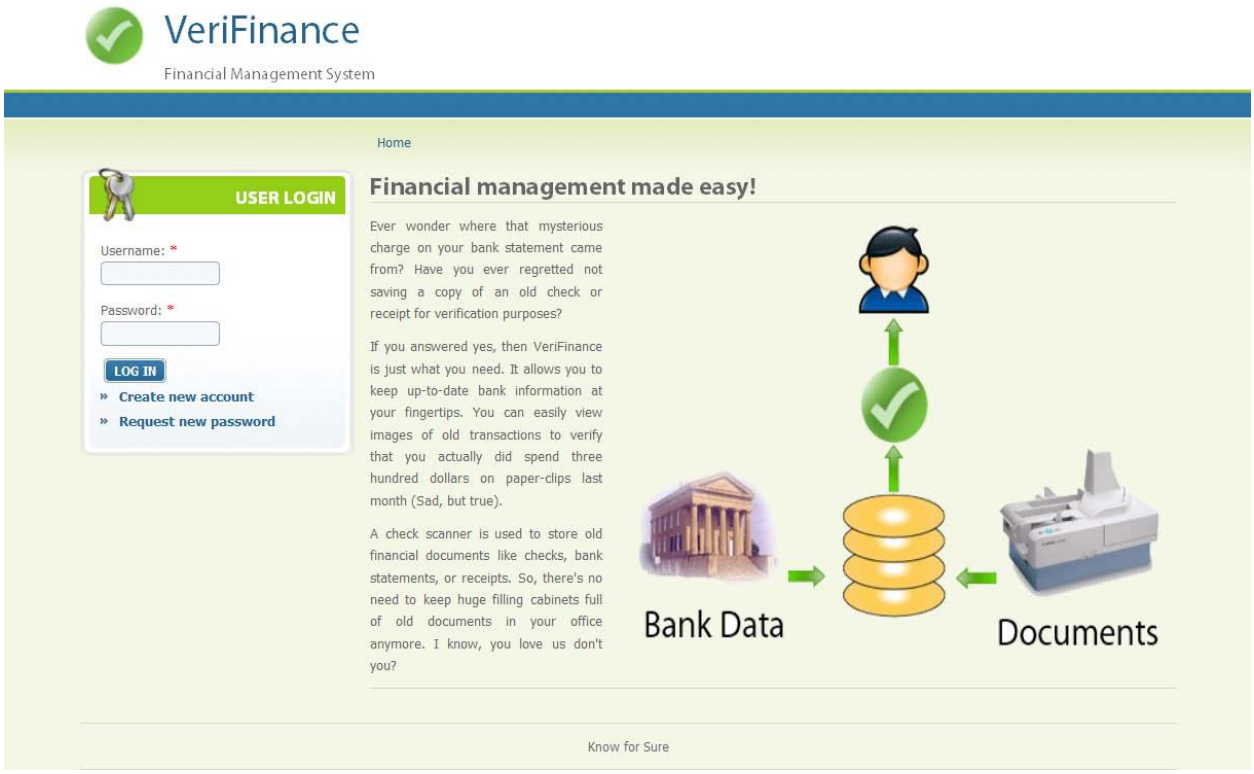


Figure 5: Welcome Page

Once the user is logged in they see a pie graph summary of their account information. The pie graph divides a user's expenses into categories defined by their bank. Each category in the graph is clickable to produce a second pie graph showing all expenses within the selected category. A back button is provided next to the graph that returns the user to the main screen. Users will love the beauty of this animated graph. Each category lights up as the cursor hovers over it. This high quality graphing library was provided by Open Chart. It is based on Javascript and Flash technology. It is an open source project that can be found at http://teethgrinder.co.uk/open-flash-chart-2/. Version 1 was used in this project, but Version 2 of open chart is now complete and may be used in the future to display bar graphs. Figure 6 below shows an example of the working pie graph that has been integrated within the database user interface.

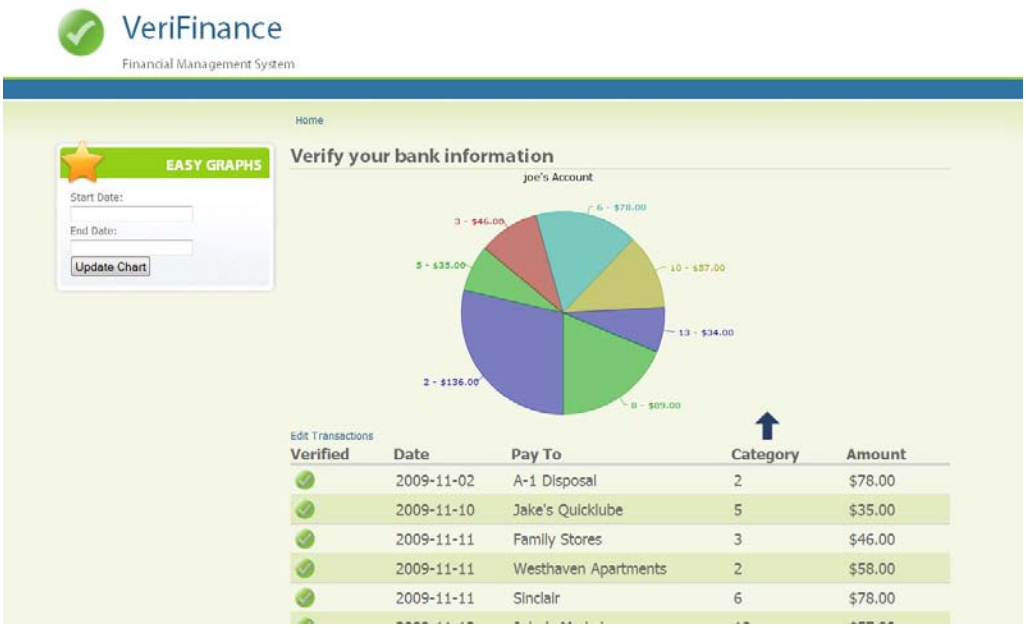


Figure 6: Transactions Pie Graph

If the user is overwhelmed by the amount of data visible he can limit the data to a specific date range. This is accomplished by using the date selectors in the 'Easy Graphs' box and then clicking the 'Update Chart' button. The table below the graph shows a list of all transactions. In the left most column are green check marks next to every transaction that has been verified by a scanned document. Rolling the cursor over the check mark shows a pop-up view of the actual document.

In order to increase the usability of this application it is possible to add and delete transactions. This is done by clicking the 'Edit Transactions' link which takes you to a new page displayed in Figure 7.

On the 'Edit' page errors can be corrected by deleting the error transaction and creating a new one with the desired changes. Input of transactions was designed to be as quick and easy as possible. The user enters the location where the expense was made into the 'Pay to' column. The 'Category' column uses a drop down menu of predefined categories (based on categories used by Wells Fargo). The date field contains a pop-up calendar to simplify date entry. Once all fields are populated the user can insert the transaction into the database by pressing the 'Return' key or by clicking the 'New Transaction' button. If desired multiple transactions can be deleted at one time by clicking multiple check boxes and pressing the 'Delete Checked' button. The 'Verify Transactions' button limits the table to view all transactions that have not been verified yet. In this way, the user can easily focus on the items of most concern.

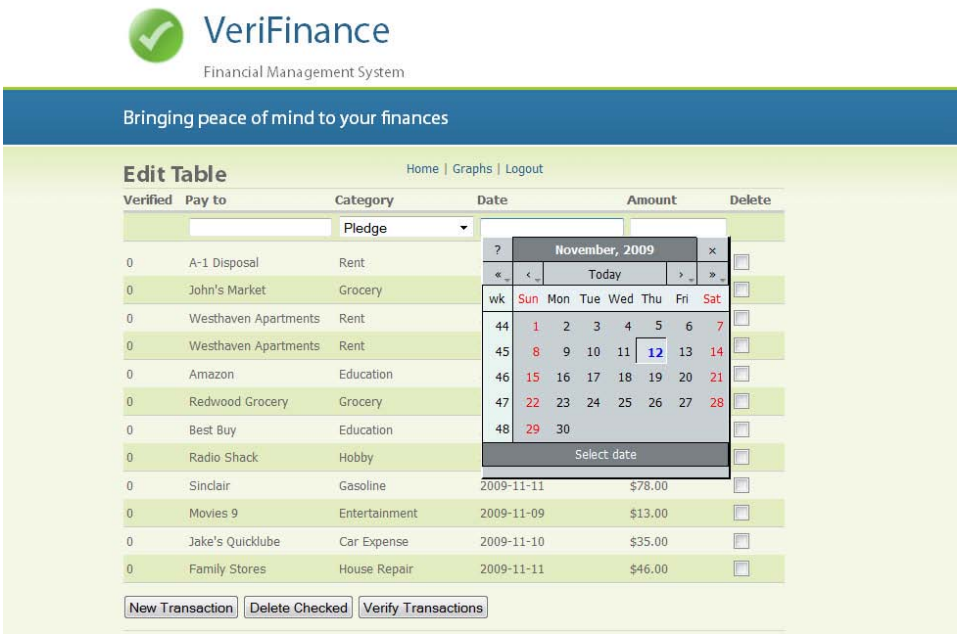


Figure 7: Edit Transactions

Security and Integration

<TO BE COMPLETED>

PDE Component

The PDE component needs to provide a way for a user to scan documents, have the system read the data, and send the correct information to the database. A Diagram of the PDE component is shown in Figure 8 below.

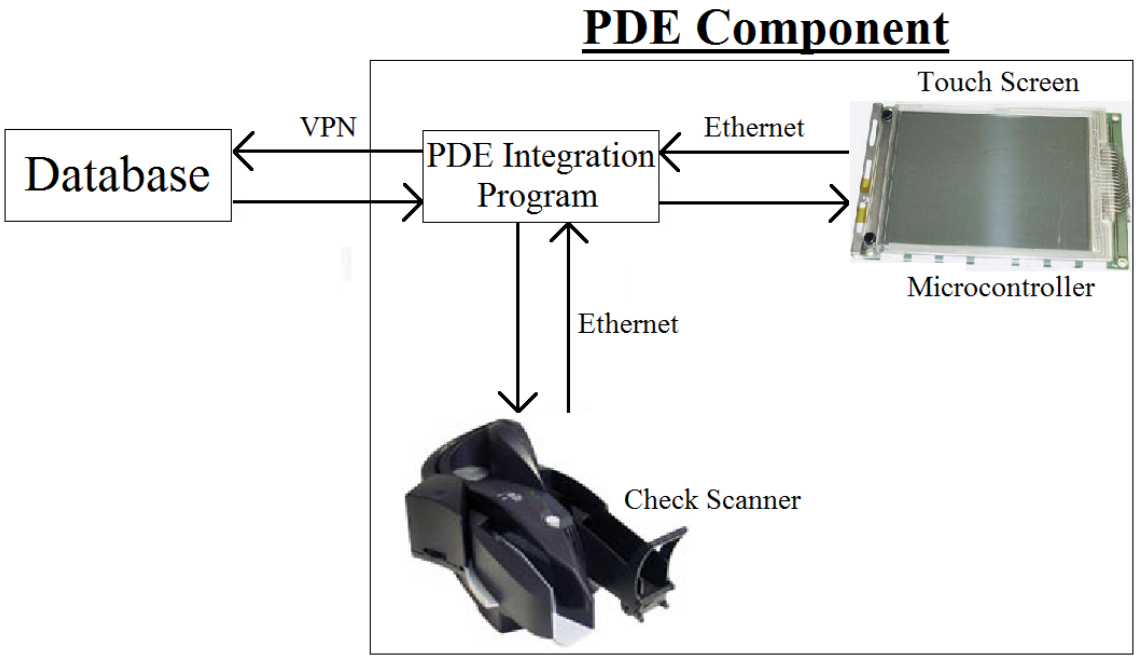


Figure 8. Basic layout for the PDE component.

PDE Component Interface Program

The PDE component interface program is the "glue" that makes all of this work for this portion of the project. It consists of two main programs. One of them is a Java program, and the other is a C++ program, of which the details will be explained now.

Java Program

The Java program within the PDE interface controls the OCR software, the MagTek scanner, and database accessing. Java was chosen because we were all use to it, and it supported all of the functionality we needed for each of these three tasks. The implementation designs for each of these components are explained in the next few sections.

A2iA Check-Reader Software

In order to provide a way for the system to read scanned data, we needed a program that would be capable of reading English characters with a high accuracy rate. After looking into this for some time, we realized how difficult this would be so we began looking for some existing software. After realizing that this would cost too much ($10,000+) we decided to start talking with some financial firms that might be using such software. After talking with Fidelity Funding, they agreed to lend us some A2iA software, as well as a "trial version" dongle that is required to run this OCR software. The dongle will work for only 5,000 documents before it doesn't work anymore.

The exact OCR software that is used in this project is A2iA CheckReader International version 4.4.1. This software is not a trial version, but is the same software that is used by Fidelity Funding. The usage is controlled only by the number of documents that are read according to a dongle.

The following steps explain exactly how to run this software (after it has been installed).

Step 1: Plug in a valid dongle.
Step 2: open up a command prompt and type: `services.msc`
You should see a window like the one in Figure 9.

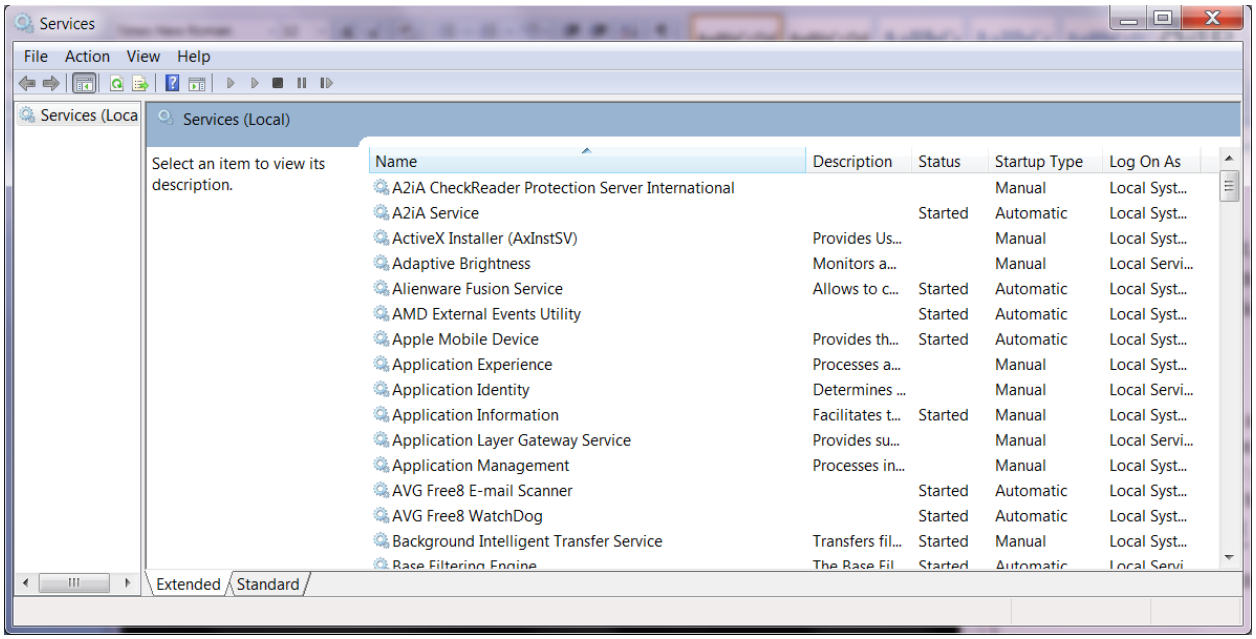


Figure 9: Services window allows you to start and stop program services.

Step 3: Start the "A2iA CheckReader Protection Server International" that is found in the services list.

Now the CheckReader software is available for programs to use. The way this software works is it needs to know exactly where to find the text that it is looking for. These settings are known as "fields". Because of this requirement, this means that for every type of document that needs to be used with this system, it must first be configured within this software. Thus, it is not practical to make it so this software can read text on all transaction documents like store receipts.

For this project, only two types of documents work with this software, checks and slips. A check is just any standard check that can be used in a transaction. Figure 10 shows a check as well as the required fields that need to be present for all checks. A slip is a document in which the user can write any and all receipt information on. This makes it so only one extra document is configured, but now the user can still insert all of their transaction information even if it doesn't involve a check. Figure 12 shows a slip as well as the required fields that need to be present for all slips.

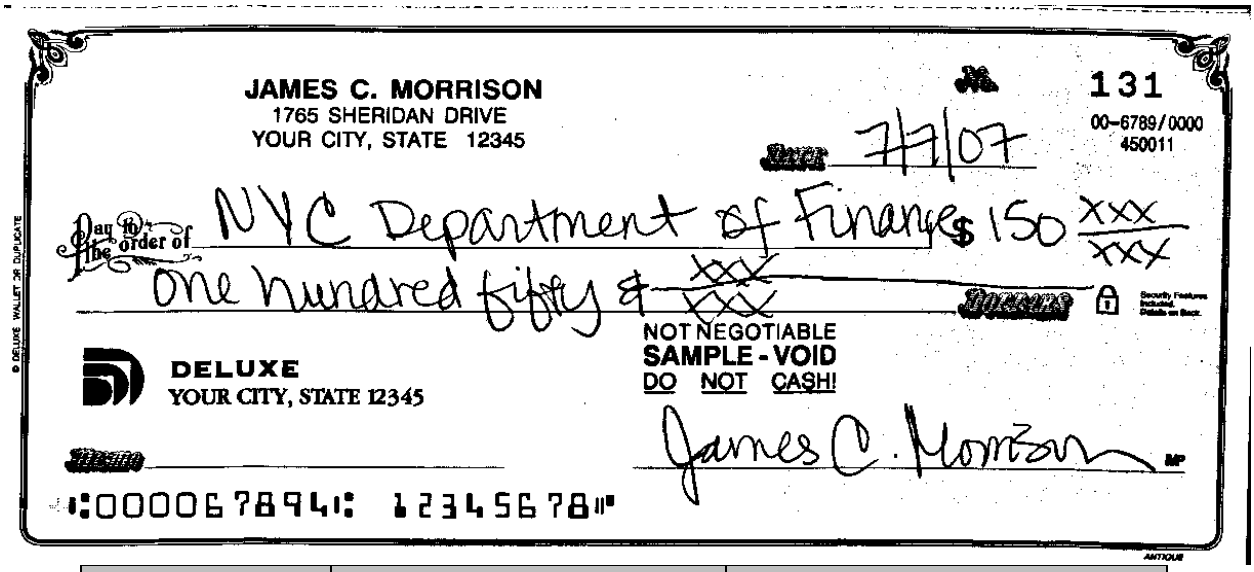
At a high level, the way that this A2iA software works is it attempts to read specific fields on a document that were configured by an administrator. For each field, the software outputs an answer of what it thought was in the field, and a confidence score between 1 and 1000. Currently, the Java program is configured such that if the confidence score is less than 700, the field will be considered invalid, and requires that a user specify what the field data is.

As previously mentioned, each document type must be configured within the A2iA software before any processing takes place. The tool that allows an administrator (or us) to configure documents is called the "A2iA Configuration Tool International". Within this tool, it is

incredibly easy to configure check documents. However, creating custom documents (like slips) take a little more work. The next few sections explain this in detail.

Check Configuration

The A2iA CheckReader software claims to work with all of the standard check documents that are commonly used in the United States. A sample check can be found in Figure 10 below.



Required Field	Field Type	Field Value in Sample Check
Pay To	Dictionary Word	NYC Department of Finance
Check Number	Numerical Character Array	131
MICR	MICR Code	000067894, 12345678
Date	Any Date String Format	7/7/07
Amount	Decimal AND Written	150XXX

Figure 10. The requirements for a check.

There are some very important key words that are used in Figure 10, and each one of them will be discussed shortly. It is very important to make design decisions first before attempting to configure anything. Once the administrator has a table like the one in Figure 10, he/she is ready to configure the document.

Before configuring the check document, it is important to understand what field types are available in order to make good design decisions. The field types that will be addressed in this section are the ones used for this specific configuration. These field types are: dictionary word, numerical character array, MICR code-line, date string format, and check amounts. Each one of these will be discussed in detail.

Dictionary Word

A field that requires a dictionary word requires a simple text file that is mapped to this field. This was a design decision because fields with dictionary words are more likely to have better accuracy when the OCR software reads the data. The OCR software uses the dictionary files to

make a more educated "guess" as to what exactly was read from such a field. Because these text files are simple, and can be accessed with a program, it becomes easy to provide a way for users (instead of administrators) to add words to dictionary files.

Numerical character array

This field type expects decimal numbers of any practical check number length. It is important to note that this array will only contain decimal numbers, and never alphabet characters. Telling the OCR software this in advance greatly increases the chance of it getting the right answer. This field type does not have a corresponding dictionary file mapped to it.

MICR Code

This field type expects to see MICR code that normally only appears at the bottom of checks. Because this code-line is always (for our purposes) printed, it becomes an easy field to extract the correct data from for this software. With this MICR information, it becomes incredibly easy to discover who the owner of the check is because this information is directly embedded within the MICR code.

Date String Format

This field can be any string of the form "month[- or /]day[- or /]year". In other words, as long has the month comes first, the day comes second, and the year comes last, this field should be able to accurately read any legible date that follows this format.

Check Amounts

This is actually two fields combined to give one result. Everybody knows that when someone writes out a check, he/she is required to write a decimal number for the amount, as well as write the amount out in words. There are field locations on each check for both entries. This software allows us to take advantage of both fields, and give the best educated answer based on what was interpreted from both the decimal and written amounts.

Once the designing is complete for checks, the check document type is ready to be configured. The following steps explain how to do this.

Step 1: Make sure that the A2iA software has been pre-configured and setup the way it was explained at the beginning of the "A2iA Check-Reader Software" section.

Step 2: Launch the "A2iA Configuration Tool". Make sure the "Dongle Server Name" is the loop back address "127.0.0.1" assuming the dongle is plugged in to the local computer. It has not been tested, but according to the A2iA documentation, this field just needs to be the IP address to the computer that has the dongle connected to it, and the local computer just needs to be on the same network.

Step 3: Create or open up a Table. The actual components of a new table will be configured later.

Step 4: Click on the "Forms" tab, and select the "defaultDefinition" label on the left side of the window. Under the "document type" option, select "check". Be sure to browse to the directory of check images that the user wishes to configure with (note: the user must have check images to work with in order to set the software settings correctly. After selecting the images to work with, select "Edit".

Step 5: This is where the main configuration takes place for the checks. Be sure to set the document type to US. This can be set by looking through the tree structure on the bottom-right until one of the options on the top-right has an option to set the document country. Select "US" s the country. Click on the "Process" button, which exists as one of the available buttons on the top of the window. Figure 11 shows what the window would look like with a successful process.

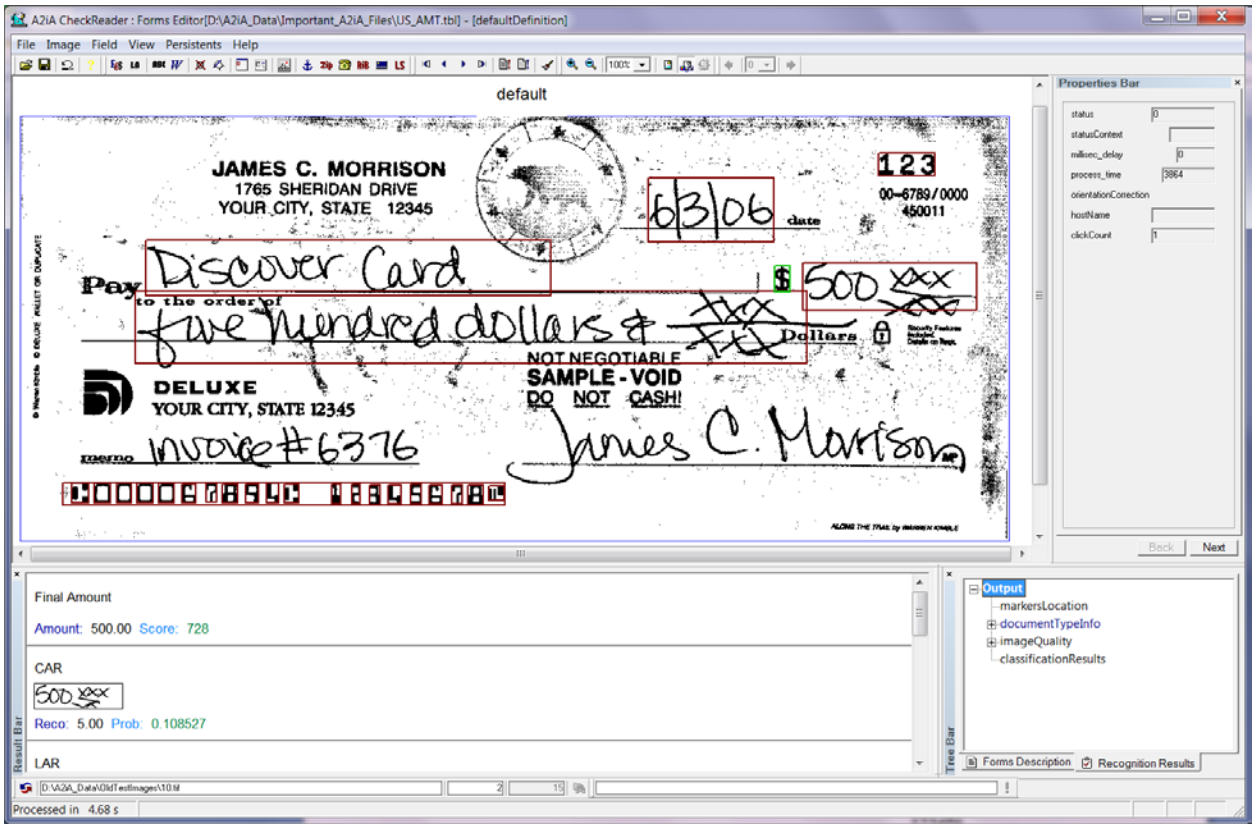


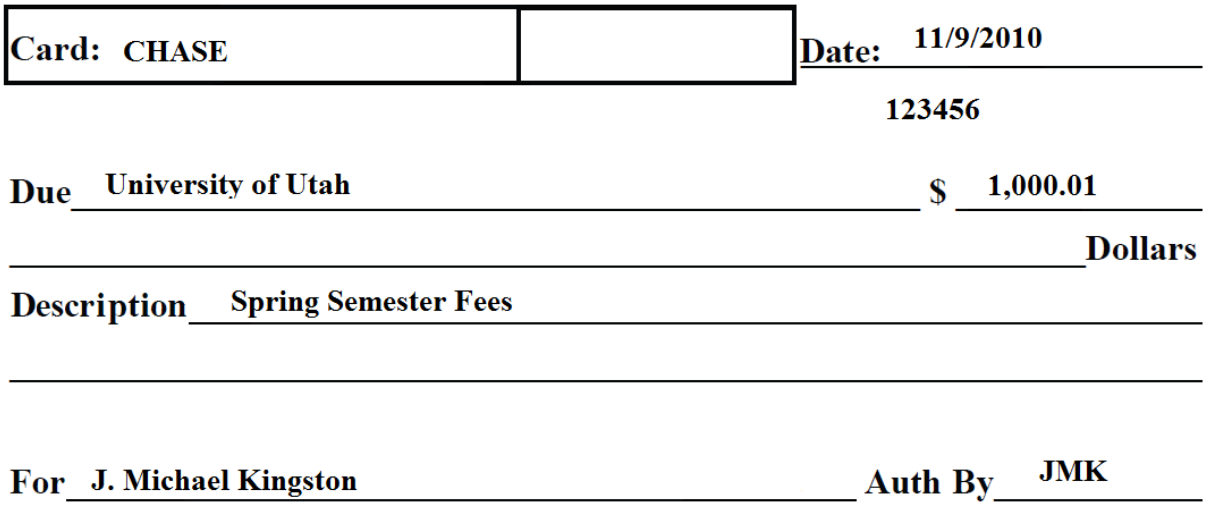
Figure 11. A2iA Configuration Tool: Current check has just been processed.

Notice in Figure 11, all of the fields that needed to be processed are surrounded with red rectangles. This indicates that the at least the correct fields were located. Also notice that processing results are shown on the bottom left. These are the results that would be sent to the database after the user approved. Note that the check document type has already been

configured in Figure 11. The way this was done was by navigating the tree structure on the bottom-left, and telling the configuration tool that the user wants to process the "check number", the "micr codeline", and the "date" in addition to the "payee" and the "amount" which would already be configured. And that's it. The check document type has now been configured. Save and close out of this configuration tool. Save in the original configuration tool (the window that should still be up), but it is not necessary to close this window unless now more configuration is needed.

Slip Configuration

Figure 12 below shows what the slip document type looks like, as well as the required fields that should have data.



Required Field	Field Type	Field Value in Sample Slip
Card	Dictionary Word	CHASE
Document #	Numerical Character Array	123456
Due	Dictionary Word	University of Utah
Date	Any Date String Format	11/9/2010
Amount	Currency Format	1,000.01
For	Dictionary Word	J. Michael Kingston
Auth By	2-3 characters	JMK

Figure 12. The requirements for a slip.

As with the check document type, there are some very important key words that are used in Figure 12. It is still very important to make design decisions first before attempting to configure anything, especially when working with a custom document. Once the administrator has a table like the one in Figure 12, he/she is ready to begin configuring a custom document.

Before configuring the slip document, it is important to understand what field types are available in order to make good design decisions. The field types that will be addressed in this section are the ones used for this specific configuration. These field types are: dictionary word, numerical

character array, date string format, currency format, and 2-3 characters. Each one of these will be discussed in detail.

Dictionary Word

A field that requires a dictionary word requires a simple text file that is mapped to this field. Thus, for each of the 3 fields that require a dictionary, a separate dictionary file is required. Again, fields with dictionary words are more likely to have better accuracy when the OCR software reads the data. The OCR software uses the dictionary files to make a more educated "guess" as to what exactly was read from such a field. As mentioned before, because these text files are simple, and can be accessed with a program, it becomes easy to provide a way for users (instead of administrators) to add words to dictionary files.

Numerical character array

This field type expects decimal numbers of length 6. Yes, this means that for each slip document, the document number must be 6 digits. It is important to note that this array will only contain decimal numbers, and never alphabet characters. Telling the OCR software this in advance, as well as the array size greatly increases the chance of it getting the right answer. As with other character arrays, this field type does not have a corresponding dictionary file mapped to it.

Date String Format

This field can be any string of the form "month[- or /]day[- or /]year". In other words, as long has the month comes first, the day comes second, and the year comes last, this field should be able to accurately read any legible date that follows this format.

Currency Format

This field requires that the amount is written on the right-hand side of the dollar sign that should be present on the slip document. Furthermore, it should contain a decimal point to define the cents. Although it isn't absolutely required to have the decimal point, it helps to improve the accuracy because the software is looking for it and if it can't find it, then this OCR software would have to make an educated decision on its own.

2-3 Characters

This field requires that there are two or three characters present. These characters must be alphabet characters instead of decimal values because this field is supposed to contain initials. Usually initials can be two or three characters long. Again, telling the OCR software beforehand what size and kind of data to read greatly increases the accuracy of the software output.

Once the designing is complete for slips, the slip document type is ready to be configured as a custom document. The following steps explain how to do this.

Repeat steps 1 through 3 in the previous section for configuring the check document type.
Step 4: Click on the “Forms” tab, and add a new label on the left side of the window. This can be called whatever, but for this project, it is called “slips”. Under the “document type” option, select “custom”. Be sure to browse to the directory of check images that the user wishes to configure with (note: the user must have slip images to work with in order to set the software settings correctly. After selecting the images to work with, select “Edit”.

Step 5: This is where the main configuration takes place for the checks. Be sure to set the document type to US. This can be set by looking through the tree structure on the bottom-right until one of the options on the top-right has an option to set the document country. Select “US” s the country. Now, each field must be configured appropriately. There are seven of them to define. Each field has already been defined in the design process, noting especially that dictionary fields are the same as “word” types in this software setup. Now for each one, locate and click the button on the top that correlates to the field type, place and size the field that appears on the document, and change the country codes for all of these fields to “US”. Now, for the dictionary fields, go through the small setup on the right (also navigates the tree stricture that’s located on the bottom-right) to define a dictionary. Locate the dictionary on the file system for each dictionary field, save this setup. Click on the “Process” button, which exists as one of the available buttons on the top of the window. Figure 13 shows what the window would look like with a successful process of the custom slip document type.

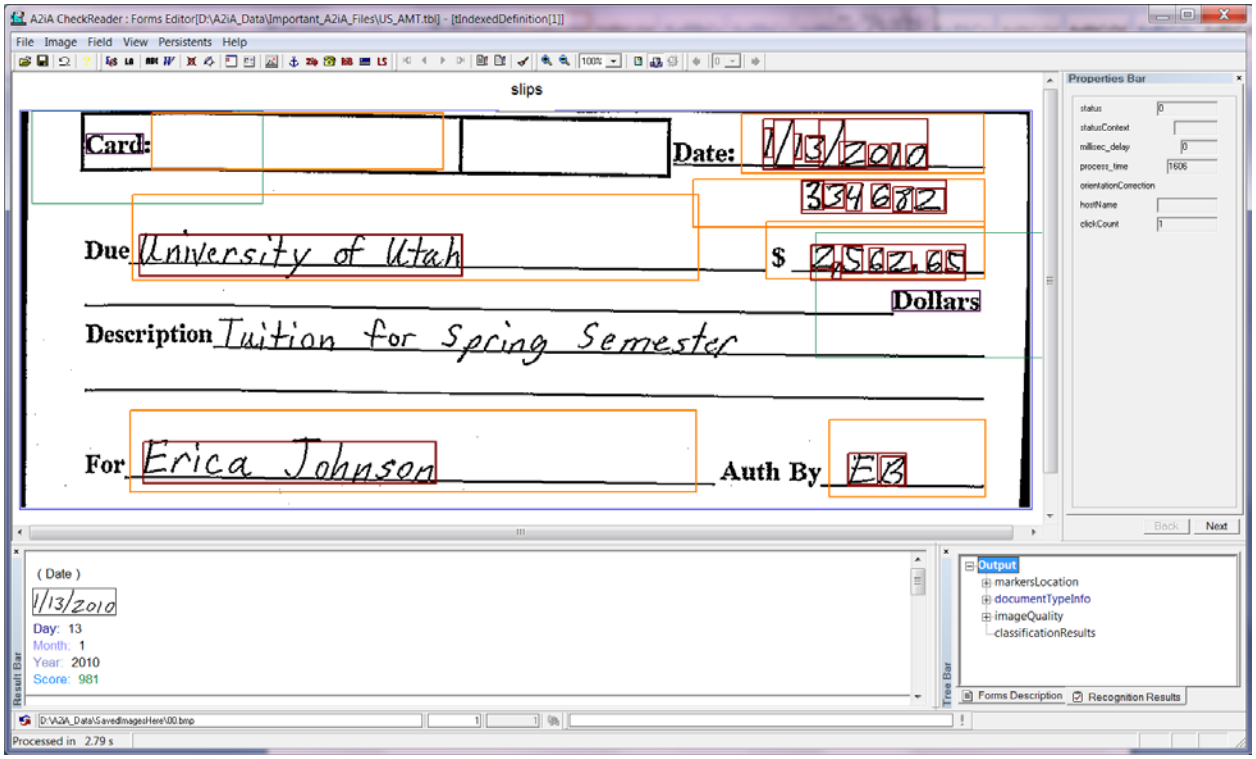


Figure 13. A2iA Configuration Tool: Current slip document has just been processed.

Step 6 (Optional): Notice in Figure 13, that the printed text “Card” and “Dollars” are surrounded by very close rectangles. This is because special markers were established to help increase the accuracy of the custom document reading. Markers are just special fields that are placed carefully on a document that search within the rectangle for a special word (like “Card” or “Dollars”). When it finds the word, it embeds a marker around it. Now, other fields like the “Due” or “For” fields can be located with respect to the marker location. This makes it so if the documents aren’t exactly the same size (maybe there is more space on the top, bottom, left, or right), the software can still easily find the fields with respect to the markers. Note that any and all markers and field locations can overlap. This doesn’t have a direct effect on the outcome.

Notice in Figure 13, all of the fields that needed to be processed are surrounded with red rectangles, which indicates that the software successfully located the valid data. This indicates that the at least the correct fields were located. Also notice that processing results are shown on the bottom left. These are the results that would be sent to the database after the user approved. Note that the slip document type has been completely configured in Figure 13, and thus every field has a result. Remember, each field can be specifically configured by navigating the tree structure on the bottom-left, and telling the configuration tool that the user wants to process each field that was needed in the design. Now the configuration setup for the slip document is complete. Save and close out of this configuration tool. Save in the original configuration tool (the window that should still be up), but it is not necessary to close this window unless now more configuration is needed.

Java Access to A2iA Software

Before working with Java, the A2iA software must be configured first. This process was explained in detail within the previous section. The dongle must also be available on the network, and the software should know where it’s at. As long as all of this is ready before hand, it becomes very easy to incorporate the Java code. Rather than explaining the code line by line, the complete code that successfully works with the A2iA software can be found in Appendix A. Instead, the most important ideas and concepts will be discussed.

There are three very important files that must be accessible to the Java program in order for the code to function properly. The first one is the “Parms” file located within the A2iA installation directory (under C:\Program Files\A2iA\...). This is the only file that is located in a static area, and should not change. When this file is invoked in a Java program, it allows Java code to make A2iA function calls that run and manage the software. This file is not changed. The second important file is the table file (default name is: “US_AMT.tbl”). This is where all of the configuration data has been stored. With this file alone, the A2iA software knows precisely what configurations have been made even without running the configuration tool. The file important file is the file which consists of the directory path to each image that needs to be processed. In other words, the A2iA software requires that it is given specific image addresses in order to process them. This file is always created dynamically, and can be generated as soon as the

image directory paths are known. The default name for this file is "FileList.txt". As long as the program can access these three files, the A2iA software will at least run within Java. Obviously if there are documents that require dictionary files, then these files must be available as well. Dictionary files were discussed in the previous section.

Another important concept is how to get the A2iA software into the correct state. For each batch of images, the software is initialized, processes the images, and then is closed down. During the initialization setup, the software has been configured so that it must know what type of document is being scanned before the processing phase. This software can be configured to determine the document type as well, but only two documents work with this system, and it becomes difficult to configure the software to determine between well known documents (like checks) and custom documents (like slips). It may be possible, but is something that is unknown right now. As a side note, it isn't hard to tell the software to determine the document type if the image belongs to a set of custom documents. This isn't an issue with this particular system because, again, there are only two documents that work with the system. The user is required to specify beforehand what type of document is being scanned.

MagTek Excella Check Scanner

A MagTek Excella check scanner the device used to obtain the document images. This scanner is capable of scanning 60 documents per minute. For the purposes of the project, this scanner must be controlled completely with a program. This is unusually easy with any language once the initial connection setup is made. The way that this scanner is being used is if the user were to pull up a browser, the user could type in a specific URL that would be sent directly to the scanner telling it what to do. In other words, the entire functionality of this scanner could be controlled exclusively with a specified URL, thus making the programming incredibly easy to create. This scanner does have an API, but is difficult to use in comparison with the URL control method.

As mentioned with the A2iA software, it is easiest to just look at the code to see how it works. So instead of explaining it line by line, only the most important ideas are discussed in this section. The complete Java code that controls the scanner correctly is available in Appendix B. One of the most important concepts to understand with this scanner is to understand how to connect to it. This is done simply with a URL object in Java. Once the URL object is created with the corresponding URL which tells the scanner to scan (or report the status), a connection is established in order to execute this command. This can happen in Java using an HTTPURLConnection object. Once the connection is established, the rest of this task involves sending appropriate XML to the scanner and closing down the connection.

The specific XML that is sent to the scanner is very important. This tells the scanner what format to put the images in, what compression type to use, and if the user wants two or more copies of this same image but in different image formats. More options include check endorsing and reading MICR code, but those options aren't relevant for this project. To sum this up, pretty

much anything that the user would need to tell the scanner can be placed within the XML code. The scanner will respond appropriately with XML code as well. Within this returned XML, the scanner reports where the image is saved (where exactly it can be retrieved), as well as any errors if there are any.

One important note with the current program design is that only one image is available from the scanner at a time. This means that the following cycle occurs when scanning documents: a document gets scanned and stored into a specified location. The Java program retrieves the image, and saves it to the local computer. Then the next document that is scanned overwrites the old image. This is not a direct problem other than it takes about two seconds to download a single image. Because the image is always being stored in the same place, this means that multi-threading is not an option. Although it is not certain if images can be stored in more than one location (by the scanner), this part of the project has not been explored the scanner is fast enough for the purposes of this project. Currently, it takes about three seconds to scan and download a single image.

Database Access from the PDE Component

The PDE component accesses the database in order to verify a user's username and password, and to store the document batch information. Within the design process, what was decided that users would be required to fix any fields that the OCR software had a difficult time reading. After this, the user could only decide whether to confirm and save the data, or to through it away and begin another batch. If the user decides to save the document information, then the Java program sends all the batch information to the database via a VPN connection.

C++ Program

The C++ program controls the touch screen's microcontroller that is used for the user interface of the PDE component. This portion had to be programmed in C++ because there was no Java support for this microcontroller.

EZLCD Touch Screen

The objective of the touch screen is to provide an interface for the user to interact with the document scanner. The touch screen will display screens to the user. The screens will prompt them their name and password to provide security to the document scanner. The screens will also display the status of the document scanner to the user and will give the user some control over the scanner.

Touch Screen Schematics

The ezLCD-101 is an all-in-one advanced color TFT LCD panel which includes:

- 640x480 pixel, 262144 color, 10.4" TFT LCD
- Embedded 32bit processor (Atmel AT32AP7000) with LCD Controller
- 4 Mega Bytes of embedded flash for storing custom fonts and bitmaps
- SD Card slot for storing bitmap, fonts and other user data up to 2 Giga Bytes

- Power supply, which generates all the voltages needed by the logic and the display itself
- Touch screen
- Interface drivers and other circuitry

The ezLCD-101 communicates with the outside world through several interfaces:

- RS232 Standard
- RS232 TTL
- USB
- I2C1
- SPI
- SD/MMC
- Ethernet2 (10 and 100 Mbit/s)

The ezLCD-101 firmware is in-field updatable and contains an extensive command set:

- Graphic commands
- Double buffering
- True Type and Open Type font rendering
- Bitmap font rendering
- Unicode support
- SD file I/O (FAT12, FAT16, and FAT32)
- Touch Screen commands

The ezLCD-101 may be in-field customized by:

- Adding custom fonts
- Adding custom bitmaps
- Customizing startup screen
- Modifying interface parameters like RS232 baudrate, Ethernet MAC and IP address, etc.
- Modifying pin functions

Communication

The touch screen runs and communicates through the Ethernet. Through the Ethernet the touch screen is able to be programmed. My program communicates with the touch screen through simple socket programming in C++. Through socket programming my program can up load the commands to the touch screen. When the touch screen turns on it will open a port and listen for incoming commands.

Programming

When the touch screen is turned on it stands ready waiting for commands to be given. The touch screen has a simple CPU that processes commands given to it one at a time. Through simple commands the screens and functionality of the screens are able to be displayed on the touch screen. Examples of sets of commands that can be given to the touch screen are shown below.

```
BUTTON_DEF_LONG B5 hex (Command)
4 4 dec (Button No)
1 1 dec (Initial State: Button Up)
8 0 dec (Button Up Icon No MSB)
8 8 dec (Button Up Icon No LSB)
8 0 dec (Button Down Icon No MSB)
9 9 dec (Button Down Icon No LSB)
255 255 dec (Button Disabled Icon No MSB)
255 255 dec (Button Disabled Icon No LSB)
1 1 dec (Upper-left corner X MSB)
4 4 dec (Upper-left corner X LSB)
0 0 dec (Upper-left corner Y MSB)
170 170 dec (Upper-left corner Y LSB)
40 40 dec (Width of the Touch Zone)
30 30 dec (Height of the Touch Zone)
```

These set of commands display a simple button on the screen and activate hardware for the touch screen so the button can be used. In the C++ code we create a character string to store the hex commands in. Once we have the commands ready in the character string, we then send the commands to the touch screen through socket programming. At certain points of the program we will need to listen on the socket connection so that if any buttons are pressed, the touch screen alerts the program. When a button is press, the touch screen sends a 4 and the button number in hex. When a button is released the touch screen sends a 6 and the button number in hex.

Screens

Figure 14 below shows the C++ program as a simple state machine. This state machine diagram illustrates how each screen goes from one to the next.

Start screen

The start screen is a screen that displays a button that says start. The objective of the start screen is to active all of the hardware with respect to the touch pad. Once all of the hardware is activated the user will be able to interface with the touch pad. With this screen the user only needs to press the start button to activate all of the hardware. Once all of the hardware is activated the touch screen will then go to the login screen.

Login screen

The login screen is a security factor for the physical extraction process. This screen restricts access to unwanted personnel to your database and your document scanner. This screen prompts the user to type in their name and password. The curser starts at the name field prompting you to type in your name. Once you type in your name you will press enter. When enter is pressed the curser will then go to the password field prompting you to type in your password. When you type in your password your will press enter. When you press enter the touch screen will compile your name and password and send it to the server that controls the document scanner. The server will check the name and password to see if it is correct.

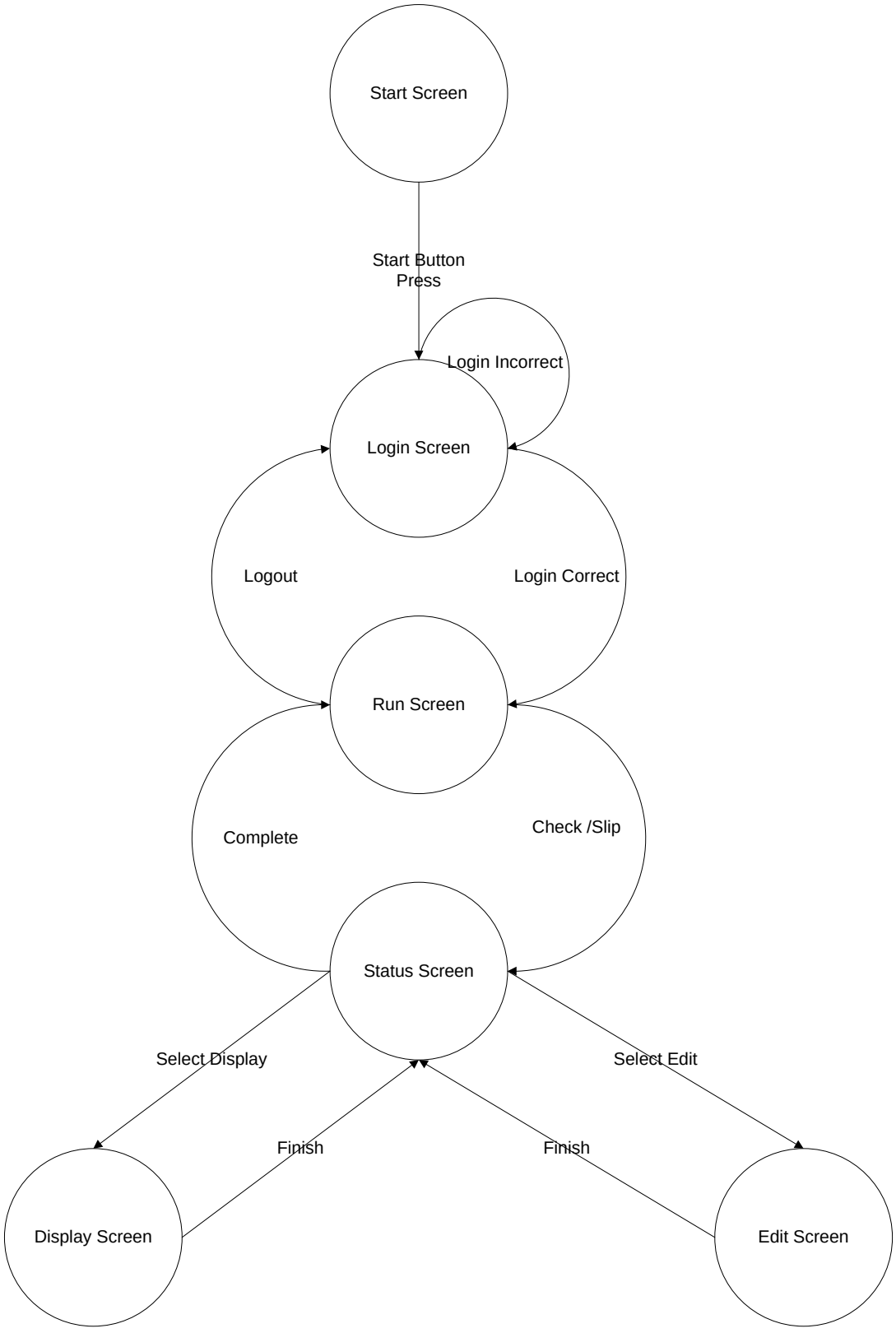


Figure 14. This state machine diagram shows how the C++ program goes to each screen.

If it is, then the user will gain access to the document scanner and the user will continue on to the Run Screen. If not then you will simply be rejected access. The escape button will end the program.

Run screen

The run screen will ask you what type of document you want to scan. You will be given two options checks or slips. Be sure to have your documents ready in the document scanner before you press any of the buttons. Once you press any buttons the document scanner will immediately begin to scan documents providing that you already have documents in the scanner. The run screen also has a logout button that will allow you to log out and return to the login screen.

Status screen

The status screen will display the condition of the documents that have been scanned. On the status screen you will see how many documents have been scanned, how many where interpreted by the OCR software correctly, how many where not interpreted correctly and what is the total amount of documents scanned correctly. On the status screen there are three buttons on the screen, complete, display, and fix. By pressing the complete button the information form the scanned documents will be dumped into the data base. Once the information is in the data base the touch screen will display the run screen. The display button will prompted the touch screen to go to the display screen. The Edit screen will prompt the touch screen to go to the edit screen.

Edit screen

The Edit screen will display an image of the document that was not read by the OCR software correctly. Under the image there is a field telling the user what field was not read correctly and prompt them to enter in the field. There will be a keyboard at the bottom of the screen allowing the user to enter in the data. Once the data is entered the user will press enter. When enter is pressed the touch screen will display the next field that was not read correctly. When all fields and fixed then the edit screen will then display the next document that was not read correctly. When all documents are read then the touch screen will display the status screen showing the new and changed documents.

Display screen

The display screen will show an image of the scanned document and will show the interpreted information that was read off of the document. At the bottom of the screen the document will show two buttons one being right and the other being left. Pressing the right document will display the next document. Pressing the left button will display the previous document. When you press the next button and you have reached the end of the batch of documents then the touch screen will then display the status screen.

Controlling the Document Scanner

The document scanner is being run by another program that is written in Java. In order for this C++ program to run the document scanner, we simply need to establish a socket connection to

the program that runs it and send in a command that tells it to start scanning documents. The Java program then runs the scanner, returns the information interpreted by the OCR software to the C++ program. This is so that the information can display on the screen. The Java program also sends in the status of the document scanner back to the C++ program if any errors arise.

Commands to send to the document scanner

Command Type: Verify User Name / Password
Description: Used to authorize a user name and password that was entered onto the screen
Command:
Touch Screen Server: `1 username password`
Document Scanner Server: `Yes/No`

Command Type: Initiate the Document scanner to run
Description: Used to tell the Document Scanner Server what types of documents are running in the machine. It also tells the Scanner Server to start running the document scanner. The two documents that can be scanned are checks or slips.
Command:
Touch Screen Server: `Slip/Check`

Command Type: Status
Description: Once the documents are scan the touch screen server will then ask the Document Scanner Server to return the status of the documents scanned. The Touch screen can also ask for the status of the scanner.
Command:
Touch Screen Server: `2 processed/scanner`
Document Scanner Server: `docs [# of docs] correct [# correct] bad [# incorrect] amount [batch total] problems [list of all problem documents in sequential order based on scanning order]`

Command Type: Display
Description: The touch screen asks for the information of the document being display on the screen. The scanner returns the information and a confidence present of each of the information. If it sends a confidence present less that 70 then we will consider this an error with the OCR software that will need to be fixed.
Command:
Touch Screen Server: `3 [doc # to display]`
Document Scanner Server: `/documents/ [image directory path] [doc #]/[doc # score] [date]/[date score] [amount]/[amount score] [payTo]/[payTo score] [payee]/[payee score] [initials]/[initials score] [card]/[card score]`

Command Type: Fix
Description: Used for the touch screen to send in the fixed information. The touch screen will send the image number and the field name fixed. The Scanner will send a response of a success or fail.
Command:

Touch Screen Server: *4 2 card/230*
Document Scanner Server: *success/fail*

Command Type: Reset
Description: Used for the touch screen to reset the Scanner server to a starting position.
Command:
Touch Screen Server: *5*
Document Scanner Server: *success/fail*

Command Type: Finish and close
Description: Used for the touch screen to close the Scanner server
Command:
Touch Screen Server: *6*

Appendix A

Not sure if the actual code should be placed here. It adds 15 pages to this document, and is harder to read in a non-programming environment.