Creating simulation-based training tools for customer relationship management with service-oriented architecture

Jeffrey Guevin

Follow this and additional works at: http://commons.emich.edu/theses

Part of the Technology and Innovation Commons

Recommended Citation

http://commons.emich.edu/theses/344
Creating Simulation-Based Training Tools for Customer Relationship Management with Service-Oriented Architecture

by

Jeffrey Guevin

Thesis

Submitted to the Department of Computer Information Systems

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Information Systems

Thesis Committee:

Stevan Mrdalj, PhD, Chair

David Dobrzykowski, PhD

July 13, 2011

Ypsilanti, Michigan
For A.G., without whom I could do nothing worth doing.
ABSTRACT

Simulation-based training offers significant benefits for employee education, but many software packages do not offer tools that can easily accommodate this type of training. This paper argues that the use of Service-Oriented Architecture can make it possible to create software-assisted simulated business environments as a method of training employees. I present the advantages and challenges of simulation as a training paradigm. I then discuss the obstacles to the creation of simulations for CRM training and offer web services as a possible solution. I describe in detail a software application that makes use of the web services interface of a CRM system. This project successfully demonstrates that web services can enable the use of simulation-based training when software does not directly include such tools.

Keywords: Customer Relationship Management, Simulation, Service-Oriented Architecture, Web Services
TABLE OF CONTENTS

Dedication ........................................................................................................................................... ii

Abstract ........................................................................................................................................... iii

List of Figures ......................................................................................................................................... vi

Chapter 1. Introduction ....................................................................................................................... 1

Chapter 2. Simulation as a Training Method ...................................................................................... 5

  Experiential Learning ...................................................................................................................... 5

  Challenges with Developing Simulations for Software Training .................................................. 8

  Open Source Software ..................................................................................................................... 10

  Application Programming Interfaces (APIs) ................................................................................. 11

Chapter 3. Extending an Application Using Service-Oriented Architecture .................................. 12

  Service-Oriented Architecture ....................................................................................................... 12

  Implementing SOA ......................................................................................................................... 16

  Web Services .................................................................................................................................. 16

  A Methodology for Web Services Implementation ......................................................................... 19

Chapter 4. The Sugar Simulation Manager (SSM) Application ....................................................... 25

  Overview ....................................................................................................................................... 25

  Sugar CRM’s Web Services ............................................................................................................ 26

  Namespace-Defined Messages and Data Types ........................................................................... 29

  SSM Application Design ................................................................................................................ 31
Design of Other SSM Functions.................................................................33

Chapter 5. Using SSM for Simulation-Based Training.............................40

Initial Data Load..................................................................................41

User Training ....................................................................................51

Analysis of Existing Data.................................................................51

Data Manipulation............................................................................58

Summary .........................................................................................62

Chapter 6. Conclusion..........................................................................64

Contributions of the Study.................................................................65

Limitations of Study and Future Opportunities for Research...............66

Final Remarks ..................................................................................70

References .......................................................................................72

Appendix: Sugar Simulation Manager Code .......................................75

A. SugarClient.cs................................................................................75

B. Form1.cs ......................................................................................91

C. Sugar Simulation Manager Name Files ........................................97
LIST OF FIGURES

Figure 1. SOA's Features and Benefits ................................................................. 14

Figure 2. An illustration of web service architecture .............................................. 18

Figure 3. Sugar CRM's soap.php interface ............................................................ 27

Figure 4. A portion of the Sugar CRM WSDL ....................................................... 28

Figure 5. The Login Transaction ........................................................................ 30

Figure 6. The Sugar Simulation Manager interface .............................................. 41

Figure 7. SampleLoad.csv .................................................................................. 42

Figure 8. Sample Name Files .............................................................................. 43

Figure 9. Inserting Random Data with the "Small" preset .................................. 44

Figure 10. Five records of random data inserted for each type ......................... 45

Figure 11. SugarCRM results of a data load ......................................................... 45

Figure 12. Links for Selecting Templates ............................................................... 46

Figure 13. The results of the Developing an Account Preset ................................ 47

Figure 14. The Contacts page in SugarCRM after using Developing an Account .......... 48

Figure 15. Bug report data ................................................................................... 48

Figure 16. Using the Bug Tracker to list bug reports ......................................... 49

Figure 17. Bug report details ............................................................................... 49
Figure 18. The "My Sugar Feed" dashlet within SugarCRM ........................................50

Figure 19. Get Record Count results ............................................................................52

Figure 20. Partial sample data used for Get Record Count example ..........................52

Figure 21. The User Status Report .............................................................................53

Figure 22. Partial data used for the User Status Report example .............................54

Figure 23. List of Jeff Guevin's meetings used for User Status Report example .........54

Figure 24. The User Comparison Report ....................................................................55

Figure 25. Meeting Data for User Comparison Report example ..............................55

Figure 26. The Missed Cases Report .........................................................................56

Figure 27. Data used for Missed Cases Report example ........................................56

Figure 28. SugarCRM interface corresponding to Missed Cases Report example ....57

Figure 29. The Account Activity Report .....................................................................58

Figure 30. Partial data used for Account Activity Data example ..............................58

Figure 31. Links for Utilities .......................................................................................59

Figure 32. After using Delete All: SSM (above) and SugarCRM (below) .................60

Figure 33. The percentage selection dialog for Randomize Cases .............................61

Figure 34. Data changed using Randomize Cases .....................................................61

Figure 35. SugarCRM interface showing randomized case data ...............................61
Figure 36. Data changes resulting from Randomize Opportunities, before and after ............62

Figure 37. SugarCRM screen showing affected Opportunities........................................62
Chapter 1. Introduction

Training employees in the use of software is an aspect of modern business that is expensive and time- and resource-consuming, but nonetheless critical in order for employees to learn to use new software effectively. This is especially the case in enterprise software rollouts, where the software can be tremendously costly, large numbers of employees are involved in training, an organization's fundamental business processes can be affected, and an implementation failure can put the company's well-being at risk. Moreover, failure is common: For example, in 2002, over half of Customer Relationship Management implementations failed to meet organizational goals (Rigby, Reichheld, and Schefter, 2002). Given these concerns, what can organizations do to ensure the success of their large IT rollouts? According to Bose (2002), training—for all levels of employee—is the most important facet of the implementation of IT projects. Additionally, training is an often overlooked method of making an enterprise software implementation a fiscal success (Schaffernoth, 2010).

One well-understood way to improve the effectiveness of employee education is through the use of simulation-based training. This can take many forms—from simple role-playing, to computer games that emulate certain aspects of an employee's daily tasks, to comprehensive simulations of the entire business environment. Effective simulation-based training will allow the facilitator to craft training sessions that will address trainees' weaknesses, respond to changing needs, and possibly include repeating and varying particular business scenarios in order to train users in specific tasks.

However, such comprehensive training tools are not included with most software, and even when some training materials are included, the particularities of a given
organization may require customization to achieve an optimal training environment for employees. Thus, for an organization to reap the benefits of simulation-based training, it may be necessary to develop custom software to meet its specific needs. How can an organization best create such training tools? The process of manipulating a third-party application in order to customize a training program can be difficult if that application does not make its source code available for modification or does not expose programming interfaces to extend the software. Instead, some software applications and systems are using the methodology known as Service-Oriented Architecture (SOA) to permit the software's functions to be extended, used in new contexts, and used together with other functions to compose larger-scale solutions. By breaking a system down into its functional components, SOA allows software to be leveraged in ways which even the software's authors may not have imagined. SOA-based systems operate by exposing the functionality of their components through some externally-accessible interface—common examples are CORBA and web services. Due to the popularity of web services, as well as the relative ease with which software can be developed to work with them, this project will involve the creation of software to work with the web services-based interface of an established product.

The application that will form the basis for this project is SugarCRM, a popular Customer Relationship Management product that is released under a hybrid open-source and commercial licensing structure. SugarCRM also boasts a huge community of users, driven both by paying customers and by users of the free Community Edition of the software. Besides an active bulletin board community of over 250,000 (SugarCRM.com, 2011), thousands of developers have contributed nearly 900 plugins and other related projects, free or commercial, to the software repository SugarForge. Such projects include connectors to Microsoft Outlook or Gmail, extension to integrate inventory management into the CRM
workflow, and software to add social networking and collaboration features (SugarForge.org, 2011). SugarCRM enables third-party development with a well-documented plugin framework, as well as a wide-ranging API for Web Services integration using SOAP (Simple Object Access Protocol) and/or REST (Representational State Transfer). It is this latter feature, and the large community that has explored and documented the use of Sugar's Web Services, that motivated the selection of SugarCRM for this thesis.

The original contribution of this thesis is twofold: first, to examine the novel questions of whether Service-Oriented Architecture makes possible the creation of simulation-based training tools which can be customized to an organization's specific needs; and second, to inform this question by showing specifically how applications that are built using a Service-Oriented Architecture provide an original solution to this problem.

Specifically, I will show how a simulation-based approach to Customer Relationship Management (CRM) training can help to form an important part of a successful CRM implementation and how SOA, using web services, enables simulation-oriented training tools to extend the functionality of SugarCRM. I accomplish this by developing the Sugar Simulation Manager, a C#-based application that uses SugarCRM's web services interface to enhance the functionality of SugarCRM. These added features will enable a training facilitator to manipulate SugarCRM's data in ways that contribute to his or her ability to manage simulation-based training sessions in a fashion that would be considerably more difficult using only SugarCRM's built-in functions. These features will include methods for creating and manipulating data and for analyzing existing data and will conform to the following lifecycle:

1. Loading of initial CRM data
2. User training, resulting in manual creation/editing of data

3. Analysis by trainer of existing data

4. Creation/editing of data, followed by return to stage 2

Based upon my review of the existing literature on simulation-based training,\(^1\) I take this approach to the challenges of managing training sessions for existing software to be a unique one.

In Chapter 2, I present the advantages of simulation as a training paradigm, introduce CRM systems, and discuss the difficulties in their implementation. In Chapter 3, I introduce the theoretical underpinnings of Service-Oriented Architecture and discuss the implementation of SOA using web services. In Chapter 4, I describe in detail a software application that makes use of the web services interface of a CRM system. In Chapter 5, I offer several examples of how this application could be used in a simulation-based training session. Finally, in Chapter 6, I consider whether this project can be called a success, and I suggest an alternative approach that might have been used in creating the Sugar Simulation Manager.

---

\(^1\) Databases searched include ACM Digital Library, Emerald Library, and IEEE Computer Society Digital Library.
Chapter 2. Simulation as a Training Method

Experiential Learning

The simulation of real-world scenarios has long been used fruitfully as a method for business training (Naylor, 1971). By presenting trainees with situations that resemble those with which they might expect to be confronted, a training program creates an environment for learning that has the advantages described by theorist David A. Kolb in his writings about "experiential learning" (Kolb, 1984). Participants can be expected to retain learned information better, learn immediately how to apply that information to the real challenges they will face in the workplace (as well as learn what kinds of challenges they are preparing to face), and engage creatively with the learning process—all benefits that traditional, rote learning often does not offer. Kolb describes four phases of learning:

- Concrete experience
- Observations and reflections
- Formation of abstract concepts and generalization
- Testing implication of concepts in new situations

All of these are understood to be stages in a repetitive model, wherein "every experiment gives the participant the ability to reflect on his actions" (Villegas, Rapp and Saven, 1997). Working from this framework, Villegas et al. describe a four-phase model of a training session:

- Theoretical instruction: Participants are given background information by a facilitator.
- Introduction to the game: The rules of the game or simulation are given.
• Playing the game: Participants "learn by doing" and "learn by experience of others."

• Group discussions: The "organizational learning" stage.

The Group Discussion is the most important stage, because

the participants must discuss the results of a game together and they must analyse
the possible connections of those results with the real problems in order for the
individual learning to have an impact on the company. […] The dialogue and
communication between participants during these meetings might help them to
clarify aspects of different problems in the game. (Villegas et al., 1997)

Simulation as a training method does not necessarily encompass all aspects of
experiential learning as defined by Kolb, but can rather be said to be an implementation of
an important facet of it—namely, the requirement that "the learner must be willing to be
actively involved in the experience" (Villegas et al., 1997). Studies that have focused on
simulations in business environments have supported Kolb's theories. Gopinath and Sawyer
(1999) write

We suggest that experiential learning processes, and specifically business simulations,
may help achieve the desired higher level conceptual processes of relativism and
comprehension through application, analysis, synthesis, and evaluation.

Their study, which had learners participate in a large-scale simulation of an athletic
footwear business, suggested that not only learning but strategic-level thinking benefited
from the simulation approach:
Our results show that the participants in the simulation exercise were able to understand the importance of focusing on strategy and taking a long-term perspective to their decision making. (Gopinath & Sawyer, 1999)

In addition to such formal studies, simulation-based training and education has enjoyed great popularity in the business world. In "Pretending to Learn," Dolezalek (2003) recounts several anecdotes of practical successes that are attributed to the use of different types of simulations and business games. For instance, the consumer products company Kimberly-Clark engaged in an ambitious simulation-based training program for its 15,000 employees, introducing them to the ins and outs of supply-chain management. They found that this process generated enthusiasm amongst its employees, a feeling of understanding of the company's goals as well as its difficulties, and a tangible benefit of $275M in cost savings gained by employee-generated waste and redundancy reduction.

Indeed, the experiential learning paradigm allows learners to "not only learn, but also understand what they're taught" and "to get so engaged in the learning experience that they're truly discovering for themselves what you want them to discover" (Dolezalek, 2003). Moreover, it offers the additional benefits of real-world compatibility, as in the case of American Express's experience in training call center employees, where the call center simulation environment resembled the real-world call center so closely that there was a "strong correlation between trainees' performance in the simulator and their performance on the call center floor" (Dolezalek, 2003). Not only were those employees who possessed an apparent natural ability to perform well able to show this talent during the simulation, but the intense simulation/training process apparently resulted in cost savings for the company. Employees who were unhappy with the simulated work ended up quitting the real job earlier.
than they would have otherwise—shortening the "unnecessary retention" time from 14 weeks to 18 days. For this reason, American Express even began using the simulation approach in its hiring process (Dolezalek, 2003).

A common theme in literature related to experiential learning is that the "debriefing" stage of the learning process is critical\(^2\) (Crookall, 1995; Villegas et al., 1997). In other words, once the simulation has concluded, participants are to be asked to discuss their experience with one another and with the facilitator. Learned material is reviewed, the process is critiqued, and actions taken during the simulation are analyzed. This can be particularly useful in an iterative simulation, where participants will have the opportunity to improve on their past performance. In fact, Dolezalek noted an HR planning director at an automotive supply company, who believed that "the debrief that followed the simulation was as important as the simulation itself" (Dolezalek, 2003). Software that enables simulation-based training can facilitate this valuable stage of the learning process. In Chapter 4, I will discuss the use of the Sugar Simulation Manager application in simulation-based training sessions for Customer Relationship Management software (specifically SugarCRM). Moreover, Chapter 5 will discuss further how participant debriefing can be incorporated into such a training session.

**Challenges with Developing Simulations for Software Training**

In the Information Age, with computer and software use being so integral to so many aspects of the business environment, employee training will naturally turn to what these technologies have to offer. This is often the case whether the purpose of the training is to familiarize trainees with a software package (where the software used in the training will

\(^2\) This corresponds to the "Group Discussion" in Villegas's model.
often be the very target of the training) or whether the training is meant to impart skills or knowledge that are not necessarily "about" the software itself (where the training software is useful to the educational process, but is itself peripheral to the end goals). Since many types of business training take the form of "games," software as a primary training tool can provide additional benefits, such as implementing scoring to grade participants' actions, providing "replayability," added interactivity, or allowing for rapid modification of the training scenarios.

However, much software does not include tools for training beyond the documentation and/or tutorials that are usually provided. A large educational industry exists to fill this gap—for instance, one source estimated that $18B was spent on IT training in 2009, much of it on training for front-office software like Microsoft Office (Austin, 2009). But not all software is granted equal weight by the training industry—we naturally expect the most popular applications to receive more attention from training companies and consultants. Unusual types of software, or less popular applications, will have fewer dedicated educational offerings. As a consequence, organizations using niche software while still hoping to reap the benefits of simulation-based training may need to employ special measures in order to offer a robust, manageable simulation to their employees.

Why might it be difficult to create a simulated environment for such products? Traditional commercial software, which is only released as binary (executable) files, with the source code remaining proprietary (and usually secret), can be difficult to extend with third-party software (Zivtech, 2011). However, other software provides ways for third parties to extend or enhance the usefulness of the product.
Open Source Software

One way in which this is possible is through open source software, wherein the underlying source code is made available, generally via a free download. There are many different licenses for open source software—ranging from the very permissive MIT License (wherein reuse of the source code is essentially unrestricted, including for profit) to the "viral" Gnu Public License (GPL) (where any product that makes use of the GPL-licensed software must also be released under the GPL), with many variations, both free and fee-based.

With open source software, a third party would be able to download the full source code for a product, make changes to it, and compile its own version of the software with additional or changed features. In this manner, a training company could make a training-oriented version of the software. Note that in this scenario, the trainees would not be working directly with the software for which they are being trained, but instead with a modified version that approximates in look and feel, more or less, the target application. The features being added would be ones that would enable a training facilitator to manage sessions, guide participants through particular tasks, or modify the conditions under which the software is being used. In order for the simulation-based training to be valuable for the participants, however, the actual experience of using the software should be very similar (or identical) to the original version.

While the ability to customize the software for training has its advantages, there are several other problems with this approach. First, the process of learning about the target application's structure can require a significant investment of time, especially in the case where the source code may not be well-documented (as is notoriously the case with open-
source projects; OSS Watch Wiki, 2009). Second, by making changes directly to the application itself, there is the chance that bugs will be inadvertently introduced, making the "training version" of the software less reliable than its source, which might in some cases negatively impact the training for which it's intended (BerliOS, 2010). Third, by creating a "fork" from the original source code at a certain version level, it becomes difficult to incorporate ongoing changes from the main product into the forked version—a difficulty that will only get worse with time and that will require periodic programmer resources to manage (Gonzalez-Barahona, 2000).

**Application Programming Interfaces (APIs)**

As an alternative, a common way for developers to permit future third-party extensions to their product without releasing their source code (or, if the product is open source, without requiring third parties to modify the code in order to create such extensions) is to build an Application Programming Interface (API) to the software. This exposes a set of the internal software routines to other developers, allowing external programs to trigger certain functionality within the target application, retrieve data from it, and otherwise interact with it (Orenstein, 2000). There have historically been a number of common conventions for defining APIs, but with the growth in web applications in the last decade, the Web Services approach has become increasingly popular (Azami, Ibañez, Mathew and Oldham, 2005). In the next chapter, I will describe the Web Services specification, discuss its advantages and disadvantages, and argue for why Web Services present a highly useful solution to the problem of creating simulations for enterprise software like CRM systems.
Chapter 3. Extending an Application Using Service-Oriented Architecture

Service-Oriented Architecture

Service-Oriented Architecture (SOA) is a paradigm for large-scale application design in which discrete, interoperable services are knit together in a flexible, loosely-coupled framework that can permit a modular, pluggable approach to the ongoing development of an enterprise system. The primary feature of SOA is "service orientation," in which software systems are conceived of as a network of discrete, on-demand components, which expose their "functionality through standard, published, and discoverable interfaces" (Hoffman, 2006). Those "containers of related capabilities" will generally encompass one or more functions which perform tasks that are related in some unifying context. Those operations will be invoked by external software, perform a set of tasks internally, and return a response to the invoking software in a predefined fashion (Erl, 2005).

As defined by Thomas Erl (2005), SOA as an approach has the following components:

- Standardized Service Contract – Services define their available operations through a descriptive document, and this document serves as a "contract" to which the service is expected to conform.
- Service Loose Coupling – Services are related to one another only through the requirements of their advertised contracts, and do not contain additional dependencies.
- Service Abstraction – Services hide the details of the inner workings.
- Service Reusability – Services are positioned so as to promote reuse in different contexts—they should be "agnostic" vis a vis the context in which they are invoked.

- Service Autonomy – The logic which governs a service's function is controlled internally, with clearly defined boundaries separating the service from external applications (or other services).

- Service Statelessness – Previous invocations of a service should not impact subsequent invocations.

- Service Discoverability – Metadata about a service's availability permits services to be discovered by entities which might need to invoke them. This is frequently accomplished by way of a "service registry."

- Service Composability – Services can be tied together into "compositions" of multiple services. Individual services might solve smaller problems, but they can be leveraged together into a larger functional unit to provide solutions for larger (or different) problems.

See Figure 1 for Erl's illustration of how the components of SOA yield a number of benefits.
The benefits of SOA include easier reuse of components and an improved ability to modify an architecture in the future. The paradigm of application creation becomes one of composition, where existing components (and possibly newly-created ones) are re-ordered and integrated to solve new problems. According to Erl:
This results in a shift where more and more business requirements are fulfilled not by building or extending applications, but by simply composing existing services into new composition configurations. (Erl, 2005)

As listed in Figure 1, several of the key advantages of the SOA approach, according to Erl (2005), are:

- Increased Intrinsic Interoperability – Software designed in a service-oriented environment is better able to share information with other software.

- Increased Federation – Applications in an SOA environment are defined independently, but work in a united fashion through standardized interfaces.

- Increased Business and Technology Alignment – The abstraction of software into discrete units modeled on business needs allows the software to be reconfigured to meet ongoing business changes.

- Increased Return on Investment (ROI) – The emphasis on reuse means that initial investment in newer services can return benefits over and over, in differing contexts.

These are long-term advantages that organizations are increasingly finding to have real, concrete payoffs, and to be worth the initial required investment. According to industry analyst Judith Hurwitz,

"It's the type of technology that your real goal is reuse and the ability to loosely couple components together[…]You can't look at this for short-term benefit because in fact the real gain happens when change happens." (Brodkin, 2007)
Implementing SOA

Service-Oriented Architecture does not require any particular technology to implement its various design elements and has traditionally been associated with technologies like Common Object Request Broker Architecture (CORBA), Web Services, and Distributed Component Object Model (DCOM). CORBA predates SOA and defines a way to create tightly-connected, secure systems that can be distributed across different computers, including those on different platforms. The technologies behind CORBA can, in fact, be used as part of an SOA. DCOM is a Microsoft-specific technology for creating networked software components, which was intended to be a direct competitor to CORBA. Largely because of the difficulties encountered in implementing CORBA or DCOM, SOA is commonly seen as the successor to these technologies, though SOA does not attempt to define quite so specifically how the architecture is to be designed, and represents a paradigm rather than a specific platform. And although some organizations have continued to use those systems for their distributed computing needs, including for SOA implementations, it is web services, especially those using the Simple Object Access Protocol (SOAP) standard, that have become the most popular way to create service-oriented architectures (Rogue Wave Software, 2008).

Web Services

Web services are one of the most commonly-used methods of integrating disparate computing services. They were initially defined as applications that expose their functionality through a particular suite of protocols (XML for the data, SOAP for the messaging encapsulation, WSDL for the interface definition, and UDDI for service discovery) in a speech by Bill Gates in 2000, and encountered some initial resistance because of their association with Microsoft. By the end of 2000, web services had earned a
commitment from the five leading IT vendors (Oracle, HP, Sun, IBM, and Microsoft), and thus have become one of several de facto standards for integration over the last 10 years (Levitt, 2001).

The definition of web services is governed by the World Wide Web Consortium, which defines them as follows:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. (W3C Working Group, 2004)

The reuse of other existing, well-known standards into a coherent stack is part of web services' appeal. Web services standardize network communication over the HTTP protocol, and the data is usually encapsulated in XML (or more recently Javascript Object Notation (JSON); W3C Working Group, 2004). The interchange of messages is managed either through SOAP or the more recent Representational State Transfers (REST) protocols. Specific functions are defined formally through the Web Services Description Language (WSDL), which insulates the extension developer from needing to know about the internal processes of the target application, or even knowing the programming language in which it is written.

The manner in which these various protocols work together to connect web services with the client applications can be seen in Figure 2. A web service will be hosted by a Service Provider (in the lower left-hand corner of the diagram), which will publish the
specification to another Service Provider (above), which acts as the directory service for the web service using the Universal Description, Discovery and Integration (UDDI) protocol. The client application (represented here as the Service Requester) will find the service by querying the directory service. Once the desired web service has been located, the Service Requester will bind to the web service by interrogating its WSDL contract. The operations exposed by the web service can then be executed by the client application.


The combination of web services' language-agnostic nature and their use of common network and application protocols can greatly simplify and accelerate the process of developing applications to interact with the web services. There are many web server
packages available, most of them free, which means that the implementation of HTTP over TCP/IP is already included. Tools also exist to automate the creation of both web services and the client software that can interact with them—for instance, Microsoft's Visual Studio.NET product can automatically generate a web services interface to existing public methods (the "bottom-up approach"). Another common feature is the automatic generation of the stubs for client software to access web methods, given a WSDL contract from an existing web service (known as the "top-down approach").

For these reasons, web services have seen widespread use as a way to connect disparate applications. Countless industries have found advantage in exposing the functionality of applications as a way to keep components of software projects platform-neutral and flexible:

Web Services are built using a group of XML standards, which, by their very nature, are platform independent. This allows an organization to deploy the Web service on the server platform of its choice and to use the Web service from any application written in VB, Java or any number of other programming languages. These client applications can be Web-based or thick clients; the only requirement is the ability to manipulate XML and to post it to a URL on the Internet or the company's intranet. Taking this into consideration, it is no surprise that many corporations are using XML-based Web services as a means by which to integrate a multitude of applications built on disparate technologies. (Shin, Wagner, and Webber, 2003)

A Methodology for Web Services Implementation

While there are several approaches to implementing web services within an application, SSM makes use of what is probably the most common methodology in order to
interact with SugarCRM's web services. A simple template for working with a web service will serve to illustrate the various components and processes of this methodology. First, I must locate a web service. The UDDI protocol defines a registry for organizations to advertise the availability of their web services, information about the service's purpose, the organization publishing the service, and the technical details of using the service (UDDI XML.org, 2006). The use of UDDI is outside the scope of this example, so I will assume that a web service has been found which I wish to integrate with my own application. This hypothetical web service will expose a function, Cube, which will require the input of an integer, and returns the cube of the supplied value.

Next, to interact with this web service, I must first be aware of its definition. I do this by inspecting the service's published Web Services Definition Language (WSDL) document (note that this corresponds to the "bind" operation from Figure 2). Different frameworks have different standards for location the WSDL, but in Microsoft's .NET web services framework, a web service located at the URL http://localhost/MathService.asmx would expose its WSDL in response to simply appending the Querystring value "WSDL" to the end of the URL: http://localhost/MathService.asmx?WSDL. The resulting WSDL might look like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/soap/
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:tns="http://tempuri.org/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 targetNamespace="http://tempuri.org/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
<s:schema elementFormDefault="qualified"
 targetNamespace="http://tempuri.org/">
```

20
This document has several key components. The WSDL:Types section defines two elements: "Cube", which is the function itself, and "CubeResponse", which is the output.
which the client will receive upon invoking the function. Within the Cube element, the
MyNumber integer is defined:

```xml
<s:element minOccurs="1" maxOccurs="1" name="MyNumber" type="s:int" />
```

which establishes that a single occurrence of an integer value is expected as input to the
function. Similarly, CubeResponse defines the integer output CubeResult:

```xml
<s:element minOccurs="0" maxOccurs="1" name="CubeResult" type="s:int" />
```

which defines what the client application will expect as output.

In an actual SOAP transaction with this web service, the client would use an HTTP
POST operation to submit the required input information. For instance, if I were to wish to
retrieve the cube of the integer value 3, I would submit the following data:

```
POST /MathService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 341
SOAPAction: "http://tempuri.org/Cube"

<?xml version="1.0" encoding="utf-8"?>
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Cube xmlns="http://tempuri.org/">
      <MyNumber>3</MyNumber>
    </Cube>
  </soap:Body>
</soap:Envelope>
```

The first part of the message is the HTTP header information (in the same format
that is used with every web browser request to a standard web page). The second (XML)
portion of the message is the SOAP message, containing information about the standards
being used for the message formation, the method being invoked (<Cube>…</Cube>),
and the actual data being submitted (<MyNumber>3</MyNumber>).

The web service would hand this data off to the underlying application that will
perform the mathematical operation—which might be written in any programming language,
might access other components, access a database, or even call another web service. My
client application does not need to know anything about how this result is created, as long as
the response conforms to the structure specified in the WSDL contract. Once the operation
is completed and the response is available, the web service would return the results as
follows:

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 362

<?xml version="1.0" encoding="utf-8"?>
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CubeResponse xmlns="http://tempuri.org/">
      <CubeResult>27</CubeResult>
    </CubeResponse>
  </soap:Body>
</soap:Envelope>

The client application would parse the XML portion of this response and, based on the
WSDL, would know to look for the element <CubeResponse> and its subelement
<CubeResult> in order to locate the desired data.

Note that at no point did the client application need to know anything about the
underlying application's function or the architecture of the supporting network. Thus,
concerns that would have been encountered previous to web services, such as requirements for the network and transport layers, have essentially been made unnecessary as a result of the use of the openly-defined (and nearly-ubiquitous) TCP/IP, HTTP, and XML protocols.

In the next chapter, I will describe how a specific client program, the Sugar Simulation Manager, makes use of the web services exposed by the Sugar Customer Relationship Management application to enhance the functionality of that application.
Chapter 4. The Sugar Simulation Manager (SSM) Application

Overview

The Sugar Simulation Manager (henceforth SSM) application is meant to serve as an example of the type of application that can leverage a web services framework in order to provide more complete, realistic simulations for training of CRM users and managers. By using SSM's features to control and analyze the data contained within a SugarCRM instance in ways that SugarCRM does not itself make possible, a training facilitator will be able to more effectively manage a simulation-based training program. While this application does not represent (nor does it attempt to represent) the full spectrum of features that might be desired for such training situations, I will show that it will nevertheless serve as both (a) a featureful, genuinely useful application which could be used for real simulation-based training; and (b) a useful example of how web services can be leveraged for these and other purposes, representing a value both to the users who will develop new tools for use with CRM software, and to the creators of the CRM software, whose creation's appeal and usefulness is increased merely by providing SOA interfaces to it. In this chapter, I will discuss the design of SSM, showing how it implements the web services methodology described in Chapter 3.

The features of SSM fall into two categories: data creation and manipulation, and analysis of existing data. The typical use for the application would be modeled on the following lifecycle:

1. Loading of initial CRM data

---

3 Here, by "simulation" I mean a "simulated business environment," in which human agents (trainees) are an integral component. Software which simulates the actions of multiple individual users (for instance, in a "multi-agent simulation") is outside the scope of this project.
2. User training, resulting in manual creation/editing of data

3. Analysis by trainer of existing data

4. Creation/editing of data, followed by return to stage 2

Stages 1, 3, and 4 would be accomplished primarily by use of the SSM. Stage 2 would be managed primarily through direct use of the SugarCRM system in which users are trained, since stage 2 will in most cases constitute the activity that generates the most direct value for the organization conducting the training. It is conceivable that web-service-enabled applications like SSM could be developed which would be intended for direct management/manipulation of a production SugarCRM system, in which case the other stages of the training lifecycle would themselves be part of the training, and the users performing analysis or data manipulation tasks would themselves be trainees.

**Sugar CRM's Web Services**

Sugar CRM exposes a number of functions through its web services interface. These are normally accessed via a URL of the form http://[domain name]/[application instance]/soap.php. This presents a user-friendly web page, shown in Figure 3, which allows a developer to click on each of the exposed methods and receive detailed information about how to invoke the method and what sort of response will be expected—essentially a friendlier version of the WSDL contract.
In addition, by appending the querystring value "?wsdl" to the URL (e.g., http://[domain name]/[application instance]/soap.php?wsdl), the user can access the full WSDL specification for the web service, including all of the available methods, as shown in Figure 4. The WSDL is how developers of third-party applications (like SSM) determine the "contract" for the Sugar CRM web services and can design their applications with the assurance that the interface will not change in the future.
A number of useful functions are included in Sugar CRM's web services. Table 1 lists a selection of those of particular interest for this project.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login</td>
<td>Given a LoginRequest object (containing a username and password), authenticates a user and returns the LoginResponse object (see below).</td>
</tr>
<tr>
<td>Create_Session</td>
<td>Given a create_sessionRequest (containing a username and password), returns a create_sessionResponse (which contains a Session ID).</td>
</tr>
<tr>
<td>Create_Contact; Create_Lead; Create_Case, etc.</td>
<td>Given minimal initial data, creates an instance of the specified module type (Contact, Lead, Case, Account, or Opportunity) and returns the ID for the new object.</td>
</tr>
<tr>
<td>Set_Entry</td>
<td>Accepts an object identifier and a name-value list, permitting an existing object to have its properties modified.</td>
</tr>
<tr>
<td>Set_Relationship</td>
<td>Accepts two module instances, and creates a relationship between them. For instance, can create a relationship between a Lead and an Account.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Get Entries Count</td>
<td>Given a module name, returns the number of existing instances of that type.</td>
</tr>
</tbody>
</table>

Table 1. Key SugarCRM web service methods

Namespace-Defined Messages and Data Types

A necessary component of working with these functions is working with the data types defined as their input and output parameters. For instance, the Login method is defined as follows:

```xml
<operation name="login">
  <input message="tns:loginRequest" />
  <output message="tns:loginResponse" />
</operation>
```

The input message type of "loginRequest" is also defined within the WSDL:

```xml
<message name="loginRequest">
  <part name="user_auth" type="tns:user_auth" />
  <part name="application_name" type="xsd:string" />
</message>
```

And furthermore, the "user_auth" type referenced within "loginRequest" is also defined:

```xml
<xsd:complexType name="user_auth">
  <xsd:all>
    <xsd:element name="user_name" type="xsd:string" />
    <xsd:element name="password" type="xsd:string" />
    <xsd:element name="version" type="xsd:string" />
  </xsd:all>
</xsd:complexType>
```

Here, the elements "user_name", "password", and "version" are simple strings, and so do not require further definition.
Note that the "login" function also returns an output message type of "loginResponse":

```xml
<message name="loginResponse">
  <part name="return" type="tns:set_entry_result" />
</message>
```

The data type "set_entry_result" is one that is used in many methods of the Sugar CRM web service and is defined thus:

```xml
<xsd:complexType name="set_entry_result">
  <xsd:all>
    <xsd:element name="id" type="xsd:string" />
    <xsd:element name="error" type="tns:error_value" />
  </xsd:all>
</xsd:complexType>
```

As we can see, we would expect an "id" value to be the result of our login attempt (or an error message). In practice, this represents the Session ID for the login. Using the above definitions, we can see what a login transaction might look like in Figure 5.

---

**Figure 5. The Login Transaction**
SSM Application Design

The SSM provides an interface to Sugar CRM's built-in web services by providing a simple Windows Forms-based control for data methods exposed by those services. The project is written in C#, using Microsoft Visual Studio 2005 and .NET 2.0. Each of the Sugar CRM methods that are needed is given a "proxy class," which allows the application to invoke those methods as though they were actually defined within the project. For instance, if a method used to create a Lead is found in the Sugar CRM WSDL as follows:

```xml
<operation name="create_lead">
  <input message="tns:create_leadRequest"/>
  <output message="tns:create_leadResponse"/>
</operation>
```

--this refers us to the definition for "create_leadRequest", which is:

```xml
<message name="create_leadRequest">
  <part name="user_name" type="xsd:string" />
  <part name="password" type="xsd:string" />
  <part name="first_name" type="xsd:string" />
  <part name="last_name" type="xsd:string" />
  <part name="email_address" type="xsd:string" />
</message>
```

Here we find defined the individual data elements that must be passed to the method in order for a Lead to be created. Likewise, we can see that the output is defined as "create_leadResponse", which shows that we can expense a string response:

```xml
<message name="create_leadResponse">
  <part name="return" type="xsd:string" />
</message>
```

To invoke this method, the application first creates an instance of the encompassing Sugar CRM SOAP definition:

```csharp
Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
```

and sets the URL property to the actual URL for the web service:

```csharp
```
Then the method can be invoked more or less directly:

```java
string wsr = ws.create_lead(SugarUserName, MD5(SugarPassword), FirstName, LastName, Email);
```

with the arguments being passed to `ws.create_lead` having been previously established in the program execution. The output string (wsr) represents the database ID value for the newly-created lead. This basic pattern holds for most of the other basic "create" methods exposed in Sugar CRM's web service.

In addition to creating objects within the Sugar CRM database, the SSM creates relationships between objects. For instance, in the process of loading initial data, when an Opportunity is created, the SSM will need an Account object to associate with the new Opportunity. Given a string "wsr" that represents the ID of the Opportunity, and a string "accountid" that represents the ID of the Account, SSM will invoke the Sugar CRM method "set_relationship", for which a data structure "set_relationship_value" is needed. This is defined in the WSDL as

```xml
<xsd:complexType name="set_relationship_value">
  <xsd:all>
    <xsd:element name="module1" type="xsd:string" />
    <xsd:element name="module1_id" type="xsd:string" />
    <xsd:element name="module2" type="xsd:string" />
    <xsd:element name="module2_id" type="xsd:string" />
  </xsd:all>
</xsd:complexType>
```

Where "module1" and "module2" will represent the types of the objects that will be associated, and "module1_id" and "module2_id" will represent the specific IDs of the objects. So to create this relationship, SSM executes the following:

```java
set_relationship_value srv = new set_relationship_value();
srv.module1 = "Opportunities";
```
Unlike the "create" methods, the username and password are not passed directly as arguments to the function. Instead, a user session must already have been created. This is accomplished through the use of the "login" method, which returns a string value "session_id". Once this session_id is available, the set_relationship function can finally be called:

```java
error_value result = ws.set_relationship(session_id, srv);
```

**Design of Other SSM Functions**

The other example SugarCRM functions from Table 1 are described below.

**Create_Session**

The Create_Session operation is defined as follows:

```xml
<operation name="create_session">
  <input message="tns:create_sessionRequest" />
  <output message="tns:create_sessionResponse" />
</operation>
```

The message types create_sessionRequest and create_sessionResponse are defined as follows. Note that all data elements are simple strings.

```xml
<message name="create_sessionRequest">
  <part name="user_name" type="xsd:string" />
  <part name="password" type="xsd:string" />
</message>
```

```xml
<message name="create_sessionResponse">
  <part name="return" type="xsd:string" />
</message>
```
The string "password" must be supplied in a hashed format using the MD5 algorithm. This is accomplished by using of the .NET library "MD5CryptoServiceProvider," which within SSM is called with the custom function MD5:

```csharp
public string MD5(string StringToHash)
{
    MD5 hasher = MD5CryptoServiceProvider.Create();
    byte[] data = hasher.ComputeHash(Encoding.Default.GetBytes(StringToHash));
    StringBuilder sBuilder = new StringBuilder();
    for (int i = 0; i < data.Length; i++)
    {
        sBuilder.Append(data[i].ToString("x2");
    }
    return sBuilder.ToString();
}
```

The string "return" in the response represents the unique ID of the authentication session that has been created.

**Create_Case**

The Create_Case operation is defined as follows:

```xml
<operation name="create_case">
    <input message="tns:create_caseRequest" />
    <output message="tns:create_caseResponse" />
</operation>
```

Since the datatype are all strings, this operation is very similar to Create_Lead (above). The "return" string represents the Unique ID of the Contact that was just created.

```xml
<message name="create_caseRequest">
    <part name="user_name" type="xsd:string" />
    <part name="password" type="xsd:string" />
    <part name="first_name" type="xsd:string" />
    <part name="last_name" type="xsd:string" />
    <part name="email_address" type="xsd:string" />
</message>
<message name="create_caseResponse">
    <part name="return" type="xsd:string" />
</message>
```
SSM's function, like the one used for Create_Lead, randomly assigns each new Contact to an existing Account.

```csharp
public bool create_contact(string FirstName, string LastName, string Email, string accountid)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    string wsr = ws.create_contact(SugarUserName, MD5(SugarPassword), FirstName, LastName, Email);
    //create relationship with random account
    set_relationship_value srv = new set_relationship_value();
    srv.module1 = "Contacts";
    srv.module1_id = wsr;
    srv.module2 = "Accounts";
    srv.module2_id = accountid;
    string session_id = get_user_auth();
    error_value result = ws.set_relationship(session_id, srv);
    ws.Dispose();
    return !(wsr.ToString() == "0");
}
```

Create_Case

Create_Case is nearly identical to Create_Contact.

```xml
<operation name="create_case">
<input message="tns:create_caseRequest" />
<output message="tns:create_caseResponse" />
</operation>

<message name="create_caseRequest">
<part name="user_name" type="xsd:string" />
<part name="password" type="xsd:string" />
<part name="name" type="xsd:string" />
</message>

<message name="create_caseResponse">
<part name="return" type="xsd:string" />
</message>
```

Likewise, the SSM function that invokes Create_Case is very similar.

```csharp
public bool create_case(string Name, string accountid)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    string wsr = ws.create_case(SugarUserName, MD5(SugarPassword), Name);
    //create relationship with random account
    set_relationship_value srv = new set_relationship_value();
```
srv.module1 = "Cases";
srv.module1_id = wsr;
srv.module2 = "Accounts";
srv.module2_id = accountid;
string session_id = get_user_auth();
error_value result = ws.set_relationship(session_id, srv);
ws.Dispose();
return (!wsr.ToString() == "0");
}

Set_Entry

Set_Entry is a SugarCRM operation that can be used for modifying existing objects of many types.

<operation name="set_entry">
<input message="tns:set_entryRequest" />
<output message="tns:set_entryResponse" />
</operation>

The message types are defined as follows:

<message name="set_entryRequest">
<part name="session" type="xsd:string" />
<part name="module_name" type="xsd:string" />
<part name="name_value_list" type="tns:name_value_list" />
</message>
<message name="set_entryResponse">
<part name="return" type="tns:set_entry_result" />
</message>

Note that the set_entryRequest object used as the input for the operation requires a part of the type "name_value_list." This data type is defined as follows:

<xsd:complexType name="name_value_list">
<xsd:complexContent>
<xsd:restriction base="SOAP-ENC:Array">
<xsd:attribute ref="SOAP-ENC:arrayType"
wsdl:arrayType="tns:name_value[]" />
</xsd:restriction>
</xsd:complexContent>
</xsd:complexType>
This datatype represents an array of name/value pairs, similar to the common "dictionary" objects in many programming languages.

In the "set_entryResponse" message, the "return" value is of the "set_entry_result" type that is described under the Login method (above). As with Login, it will indicate whether the operation was successful, or include an error message if it was not.

An example of SSM’s use of Set_Entry is in the delete_case method, which will delete an existing Case object in SugarCRM.

```java
public void delete_case(string case_id, Thesis.sugarcrm.sugarsoap ws) {
    /* given the GUID for a case, delete it */
    name_value[] fields = new name_value[2];
    fields[0] = new name_value();
    fields[0].name = "id";
    fields[0].value = case_id;
    fields[1] = new name_value();
    fields[1].name = "deleted";
    fields[1].value = "1";
    ws.set_entry(this.sessionId, "Cases", fields);
}
```

Here SSM instantiates an array of the "name_value" data type (which is defined by SugarCRM's WSDL). The first row in the array identifies the case by passing in the Case ID value. The second row identifies the property of the Case which will be updated—in this case, it sets the property "deleted" to "1". Finally, in the last line—

```java
ws.set_entry(this.sessionId, "Cases", fields);
```

—the second argument ("Cases") identifies which module the "id" value corresponds to. If a different module name, such as "Leads," were supplied here, then the Leads instance with the same "id" value would be deleted instead of a Case.
Get_Entries_Count

The Get_Entries_Count operation will return the number of undeleted instances of a SugarCRM module. The operation and its corresponding messages are defined as follows:

<operation name="get_entries_count">
  <input message="tns:get_entries_countRequest" />
  <output message="tns:get_entries_countResponse" />
</operation>

<message name="get_entries_countRequest">
  <part name="session" type="xsd:string" />
  <part name="module_name" type="xsd:string" />
  <part name="query" type="xsd:string" />
  <part name="deleted" type="xsd:int" />
</message>

<message name="get_entries_countResponse">
  <part name="return" type="tns:get_entries_count_result" />
</message>

Of note are the "get_entries_countRequest" parts "session," which is the Session ID for the current authentication session; "module_name," which is the module type being counted; "query," an open-ended parameter where database query parameters may be specified; and "deleted," which, when set to "1", will include deleted objects in the count returned. The response message "get_entries_countResponse" is of the type "get_entries_count_result," which is actually a simple data type very similar to the "set_entry_result" type described earlier.

<xsd:complexType name="get_entries_count_result">
  <xsd:all>
    <xsd:element name="result_count" type="xsd:int" />
    <xsd:element name="error" type="tns:error_value" />
  </xsd:all>
</xsd:complexType>

The "result_count" value will contain integer value of the number of matching records.

SSM uses Get_Entries_Count by way of the GetCount method.
public string[,] GetCount()
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    //create array for holding results
    string[,] results = new string[5,2];
    results[0, 0] = "Contacts";
    results[1, 0] = "Accounts";
    results[2, 0] = "Leads";
    results[3, 0] = "Opportunities";
    results[4, 0] = "Cases";

    get_entries_count_result cnt;

    for (int i = 0; i <= 4; i++)
    {
        cnt = ws.get_entries_count(this.sessionId, results[i, 0], "", 0);
        results[i, 1] = cnt.result_count.ToString();
    }

    return results;
}

The names of each of the module types (Contacts, Accounts, Leads, Opportunities, and Cases) are first placed into an array. Then for each of those module names, 
Get_Entry_Count is invoked, and the result is placed into the second column in each array 
row. The resulting string array is returned to the calling function.

In the next chapter, I will show that SSM successfully uses SugarCRM's web services interface to create, analyze, and manipulate SugarCRM data, and discuss how it can be used by a facilitator to control SugarCRM in a simulation-based training.
Chapter 5. Using SSM for Simulation-Based Training

Even though the Sugar Simulation Manager application is not meant to be a fully-featured tool for managing complex training sessions, it does show that such an application can be built by leveraging Sugar CRM's web services interface. And although it does provide tools for managing the data in a CRM system in ways which are not supported by the SugarCRM product, there are, to be sure, countless other tools that could be developed to enable the creation of a rich training environment. Nonetheless, it does offer a number of features that would be useful in such training. This chapter will show how the features of the SSM accomplish the goals of this project and will additionally describe ways in which these features might be used in different training scenarios to help create an immersive simulation for the participants. The framework I will use for evaluating SSM's features will be the following training lifecycle (introduced in Chapter 4):

1. Loading of initial CRM data
2. User training, resulting in manual creation/editing of data
3. Analysis by trainer of existing data
4. Creation/editing of data, followed by return to stage 2

Throughout this chapter I will refer to elements of the SSM Graphical User Interface (Figure 6).
Figure 6. The Sugar Simulation Manager interface

Initial Data Load

The initial loading of data is an essential, and perhaps primary, part of the process of creating a data environment for users to engage in training tasks. In this section, I will show how SSM makes it possible to load data in several different ways: from a single import file; with random data generated from "name files"; or by using either of two data loading "templates". This will show that SSM is able to use SugarCRM's web services to add data to the SugarCRM database, accomplishing the goals of the first phase of our training lifecycle.

The source of the data to be loaded can be either:
1) A comma-separated file with explicit values to be loaded, which allows for the same data to be loaded multiple times (so that all users may have the same initial data set, for instance). A sample file (SampleLoad.csv) is included with the application, as is a document describing the format of the file, as shown in Figure 7.

![SampleLoad.csv](image)

*Figure 7. SampleLoad.csv*

2) Random names which can be generated from sample "name files". These are named "MaleFirst.txt", "FemaleFirst.txt", "Surnames.txt", and "CompanySuffix.txt". Sample files with the most common given names and surnames in the USA are included with the project under Appendix C. MaleFirst.txt, FemaleFirst.txt, and Surnames.txt are used to generate records where an individual person is indicated (Leads, Contacts, and Opportunities). Surnames.txt and CompanySuffix.txt are used for generating Account (e.g., corporation) names, such as "Smith Engineering" or "Phillips Architects". Trainers can edit or replace these line-delimited files (which are shown in Figure 8) to meet their own needs, for instance in the case where Account names should all be from one particular industry.
If using the "random" option shown in Figure 9, the user can specify how many of each type of CRM entity (Lead, Contact, etc.) should be generated. If specific data are provided, then the number of entities will exactly correspond to what is provided in the .csv file.

Trainers should tailor the initial data load to reflect the needs of the simulation. For instance, if the training session will focus on developing opportunities into active accounts, then it could be useful to create a larger proportion of opportunities and leads to contacts and cases. For the most general cases, where careful tailoring to a particular training task is
not needed, "Small/Medium/Large" links are available to create an initial dataset of 25/100/500 of each object type (Case, Contact, Lead, Opportunity, and Account), and those pre-filled numbers can be edited manually before clicking the "Execute Data Load" button. Figure 9 shows the results after the "Small" link has been clicked.

Figure 9. Inserting Random Data with the "Small" preset

Figure 10 displays the table data results of clicking the "Execute Data Load" button if the user has manually entered "5" for each of the record types, and Figure 11 displays an example of associated Account, Contact, and Opportunity records created during the data load.
Figure 10. Five records of random data inserted for each type

Figure 11. SugarCRM results of a data load
The simulation/training approach for which SSM is intended makes it possible to tailor the data load for more specific training tasks as well. "Preset" data loads for two such tasks are included here, but while they could serve usefully in real-world CRM training sessions, they are not meant to represent the only (or necessarily best) data-loading options. Rather, they are meant to serve as models for the sorts of situation-specific data loads that could be useful in creating simulation-based training programs.

The first "template" included with SSM is for the task of "Developing an Account," shown in Figure 12, a sales-oriented training sequence.

![Figure 12. Links for Selecting Templates](image)

- **Templates**
  - Developing an Account
  - Managing a Support Backlog
  - Execute Data Load

This template will establish a number of initial Account objects, some of which will have a complete set of (randomly generated) contact, opportunity, and other historical information, and others which will contain very little, if any, of such data. See Figure 13 for an example: the account "Cantrell Designs" has been associated with two contacts, "Penny Rodriguez" and "Bernice Downs", and the account "Mejia Heavy Industries" is associated with "Jackie Craig" and "Stephanie Mathis". However, most of the other accounts that have been created do not have any contacts associated with them.
Figure 13. The results of the Developing an Account Preset

The trainees in a case like this will be expected to identify those accounts that need to have their sales potential developed and will simulate establishing contacts, finding sales opportunities, and creating Actions such as scheduled meetings, phone calls, and so on.

Figure 14 shows the Contacts screen from SugarCRM, which is one screen trainees would use to see which contacts and accounts are associated (SalesAgility.com, 2011).
The second included "template" is for the support-oriented task of "Managing a Support Backlog" (Figure 12), in which accounts will be generated with a number of outstanding bug reports or other customer complaints. Figure 15 shows the sample bug report data that would be generated using this option.

Figure 15. Bug report data

Figure 16 shows the result in the SugarCRM interface, using the Bug Tracker list to find outstanding reports.
When using this option, trainees would be required to analyze the support backlog, prioritize issues, assign tasks to support team members, and so on—in other words, to manage the support lifecycle in a simulation with a large number of accumulated problems that threaten the organization's customer relationships. Figure 17 shows the details of a bug report, with some manually entered "Activity" data that a trainee might enter in response to the bug.

A final note on processing the Initial Data Load: whichever type of data load the user creating the data load job chooses, he or she will specify the SugarCRM
username/password to be used to access the web services. The account name selected will be displayed within SugarCRM—for instance, within the "My Sugar Feed" dashlet, text like "[Account Name] created a new case [Case Name]" will be listed for each object created, as shown in Figure 18—so the user should consider this while creating the simulation script and choosing the account to be used. Multiple accounts could be used for different portions of the data load if the simulation script includes responding differently to data from different sources—for instance, if assignments are meant to come from different (simulated) managers. The trainee's own user accounts can be used for this purpose as well.

*Figure 18.* The "My Sugar Feed" dashlet within SugarCRM
User Training

As noted earlier, this portion of the training lifecycle takes place within the SugarCRM application and does not use SSM. For my purposes here, I will take the details of how trainees would be instructed to use SugarCRM during their training session to be outside the scope of this study.

Analysis of Existing Data

Multi-stage training sessions can benefit from a script that involves data manipulation in response to the trainees' previous actions. To assist with this, the SSM application provides some analytical functions that can supplement the built-in reporting tools included with SugarCRM. In this section, I will demonstrate how each of these SSM functions successfully assists a training facilitator with analyzing the data that exists within the SugarCRM database. I will also make several suggestions about how this analysis might fit into the overall training program.

The simple "Get Record Count" link is available under the "Utilities" section of the SSM interface. Using this utility will simply count the number of records that exist of each of the main record types (Contact, Account, Lead, Opportunity, and Case) and return the result with the "Status" window of SSM. Figure 19 shows the result that would be displayed if the data shown in Figure 20 were present in the SugarCRM database.
Several more sophisticated tools are also included in SSM. The "User Status Report" tool allows the trainer to receive a summary of each user's activities within CRM (see Figure 21). Included activities are:
• Created Tasks

• Logged Calls

• Scheduled Meetings

• Edits to customer information (Case, Contact, Lead, Opportunity, or Account)

• Attached Documents

• Marketing Campaign creation

• Assigned/Closed Support Cases

These results should be used by the trainer to highlight areas where trainees are meeting (or falling behind) expectations and can be used as a departure point for discussions with trainees about their simulated activity. For instance, if the training involves role-playing with customer calls but a trainee is not logging calls, this will be apparent to the trainer.

*Figure 21. The User Status Report*
Figure 22. Partial data used for the User Status Report example

Figure 22 displays a portion of the data used in the User Status Report example in Figure 21—note that the user "jguevin" is associated with two meetings and three cases, as reflected in the report. Figure 23 shows the SugarCRM interface with a listing of the meetings for the user "Jeff Guevin", used in the same User Status Report example.

Figure 23. List of Jeff Guevin's meetings used for User Status Report example

The "User Comparison Report," shown in Figure 24, generates a summary of all users' activities, using a simple scoring mechanism to show which users are entering the most CRM data, both total and broken down by type of data. Figure 25 shows the sample meeting data that existed when this report was run. This report should be used by the trainer to make sure low-scoring individual trainees understand the system well enough to use it effectively. However, care should be taken not to make data-entry a competition in its own right, since the quantity of data is not nearly as important as its quality, and trainees
should be learning to use the system appropriately and efficiently, not to flood the system with unnecessary information.

![User Comparison Report](image)

*Figure 24. The User Comparison Report*

<table>
<thead>
<tr>
<th>Username</th>
<th>Activity Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>jguevin</td>
<td>Meetings</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Cases</td>
<td>3</td>
</tr>
<tr>
<td>User3</td>
<td>Meetings</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Notes</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 25. Meeting Data for User Comparison Report example*

The "Missed Cases Report" shown in Figure 26 will display support cases that have not been responded to by any user, whether the case is assigned to a particular trainee or remains unassigned. Figure 27 shows the data that were in place when this example of the Missed Cases Report was run, and Figure 28 displays the SugarCRM interface which users would see in this case. This report will be useful to the trainer in showing support trainees
the proper procedures for responding to new (or unresolved) support cases, which have been overlooked or inadvertently neglected. The most effective response can vary greatly between organizations, depending on how each manages support issues, balances workload, and so on. Individuals with a relatively high number of missed cases may need to be given a specific workflow that involves periodically checking for new cases assigned to them. On the other hand, if a support queuing system is being used as part of the training, then it may be that all trainees will bear shared responsibility in the case where a backlog of unhandled cases exists at the end of the training stage.

![Missed Cases Report](image)

**Figure 26.** The Missed Cases Report

**Figure 27.** Data used for Missed Cases Report example
The "Account Activity Report," shown in Figure 29, will allow the trainer to chronologically view all CRM activity related to a particular account. See Figure 30 for an illustration of how the cases associated with the particular account are included in the report. This view can be instructive in showing how each trainee is managing their accounts—for instance, are they neglecting some in favor of others? Is support being emphasized over developing new sales possibilities? Are certain types of actions being taken only with particular accounts? This analytical perspective can also help a trainer see whether accounts being shared among multiple trainees are being managed properly, since team-based account management has its own challenges which may need to be focused upon in training.
Figure 29. The Account Activity Report

<table>
<thead>
<tr>
<th>Account</th>
<th>Activity Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>STANLEY INDUSTRIES</td>
<td>Meetings</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Assigned Cases</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Meetings</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Created Tasks</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Logged Calls</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Attached Docs</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Campaign</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 30. Partial data used for Account Activity Data example

Data Manipulation

In most training scripts where the above analytical tools are being utilized, the trainer may want to respond to previous trainee activity by manipulating the existing data set en masse rather than simply editing records individually within the CRM system. This can involve several approaches, including (a) the wholesale deletion and recreation of the simulation scenario; (b) leaving existing data in place, but adding new data for trainees to work with; and (c) editing the existing data to allow trainees to "follow up" on their actions.
from the preceding stage. In this section, I will demonstrate how several SSM features accomplish these tasks using SugarCRM's web services.

For the first two of these approaches, the process is similar to that involved in the initial data load. First, deletions are performed using the "Delete All Records" link on the main SSM form, shown in Figure 31.

![Figure 31. Links for Utilities](image)

Figure 32 shows the results of using the "Delete All" link. Second, to add new data, the trainer can simply fill in the "Random Data Quantity" values for one or more object type (Case, Contact, etc.) and click "Execute Data Load". Existing data will not be affected, but brand new records will be created. Note that, since new Cases, Leads, Opportunities, and Contacts are associated with random Accounts, some of the Accounts created in the previous iteration will likely be selected for association with these newly-created objects.
For the third approach, two functions are provided to manipulate existing data. The first, "Randomize Support Cases" link, shown in Figure 31, will update existing Case objects by changing their status, priority, and assignment. The trainer can select the percentage of existing cases that will be affected, as shown in Figure 33, so that the data manipulation creates an effect that is realistic given the demands of the training script (for instance, a "support crisis" can be created by specifying that a high proportion cases are updated). If the "Randomize Assignment" option is selected, this can help to create situations in which trainees must cooperate with one another to share information about ongoing cases or accounts that are changing hands.
Figure 33. The percentage selection dialog for Randomize Cases

Figure 34 shows the case data in the SugarCRM database, before and after the Randomize Cases function is used, with the changed data outlined in red. Figure 35 shows an example of the SugarCRM interface for the data after the function is used.

Figure 34. Data changed using Randomize Cases

The second function, "Randomize Opportunities", shown in Figure 31, will perform much the same function as "Randomize Support Cases" but with an emphasis on sales rather than support. Randomly selected existing Opportunities will have their Opportunity

Figure 35. SugarCRM interface showing randomized case data

...
Amount, Sales Stage, and Probability changed, and trainees can be asked to respond appropriately to these changing circumstances. Figure 36 displays an example of opportunities that have had some data randomly modified with this function. The data present before the change are shown above and the modified data below, with the specifically changed data points outlined in red.

**Figure 36.** Data changes resulting from Randomize Opportunities, before and after

Figure 37 shows the SugarCRM Opportunity screen, displaying the records that were highlighted in Figure 36.

**Figure 37.** SugarCRM screen showing affected Opportunities

**Summary**

As I have now shown, the Sugar Simulation Manager demonstrates that it is possible to use web services to extend the functionality of SugarCRM in ways that facilitate the creation of simulation-based training sessions for that software system. Each of the SSM
functions used in this chapter have used SugarCRM's web services interface to accomplish their respective tasks, without requiring either direct access to the database⁴, or modifications to the SugarCRM code base. I have used a 4-stage model for a training lifecycle and shown how each of the components of SugarCRM fits into stages 1, 3, or 4 (not including stage 2, which is the stage at which trainees would directly use SugarCRM). Finally, I have included, along with each function, suggestions as to how a training program could most effectively make use of those functions towards the end of creating a successful simulation-based training program.

⁴ With the exception of several table indexes that I added manually to improve the performance of the SugarCRM database.
Chapter 6. Conclusion

The goal of this project is to seek an answer to the questions of whether, and how, Service-Oriented Architecture makes possible the creation of simulation-based training tools that can be customized to an organization's specific needs. This project should be considered a success, as it has demonstrated that it is possible to leverage the web services interface supplied by SugarCRM to create an application to manage simulation-based training sessions. By using the Service-Oriented Architecture approach, the Sugar Simulation Manager application allows a user to perform a number of functions that are not possible, or are considerably more difficult, with SugarCRM's built-in interface. These functions include the rapid creation of randomized objects using simply-defined text files, the random association of those objects with one another, the use of training templates to create an initial data load tailored to particular training needs, utilities to quickly remove all data or to count the number of records for each record type, and several reports to assist a training facilitator in evaluating trainees' performance.

SugarCRM is a project that has released a portion of its source code to the community under an open source license, which provides an alternative method of attempting to extend its functionality to accommodate the needs of a simulation-based training session. However, as I noted in Chapter 2, creating a "fork" of an open source project in order to add functionality to it carries with it burdens that can be significant, costly, and ongoing. Moreover, SugarCRM's particular open source strategy does not make available the entire product, but only certain "non-commercial" portions. Much of the software offered by SugarCRM is proprietary and thus not modifiable by third parties (Augustin, 2010). And SugarCRM is unusual in how much of the product is available—
other CRM solutions are entirely proprietary, so this sort of third-party development would be impossible. However, the SOA approach to enterprise architecture offers a method of functional extension for SugarCRM as well as other, competing products. The web services approach used in this project is just one example of the benefits the SOA approach can provide.

**Contributions of the Study**

The results of this project show that the Service-Oriented Architecture approach permits the development of software to extend the capabilities of SugarCRM to enable the creation of simulation-based training tools. I have demonstrated that the web services included with SugarCRM can be leveraged to create functions for SugarCRM that are not included within that product. Additionally, I have shown that a specific methodology for interacting with web services, described in Chapter 3, provides a practical approach to developing an application that can successfully utilize available web services.

Moreover, the source code for the Sugar Simulation Manager, included in the Appendix of this thesis, provides many concrete examples of the particulars of developing software for this purpose. While SSM is not intended to include every feature that would be needed in mature simulation-based training software, it is a functional C# application that does interact with SugarCRM's web services in the ways described in Chapter 5. In this way, I have also shown specifically how the goal of creating simulation-based training software can be realized. If new functions were added, and those already included were modified to make them more powerful and/or flexible, SSM could be built upon to take advantage of many other features of SugarCRM and could thus serve as the basis for a more fully-featured training product. Even though the specific features that such an extension to SSM might
have are beyond the scope of this project, the model provided here is a concrete contribution as a foundation for such future development.

**Limitations of Study and Future Opportunities for Research**

Additional uses for the SOA approach.

The somewhat narrow focus of this project on training for Customer Relationship Management systems should not lead one to conclude that it does not have implications beyond the stated scope. For example, three ways in which the consequences of this study might be more far-reaching are 1) the use of SOA with other types of CRM training, 2) the use of SOA in non-training contexts, and 3) the use of SOA with the creation of training tools for non-CRM applications. These elaborations on the contents of this thesis may serve as areas for fruitful ongoing research, taking the lessons learned from this project and extending them into related areas that were not covered here.

First, while I have restricted this study to the creation of tools for simulation-based training, there is nothing intrinsic to the SOA approach that would require that the sort of training being done be very similar to what is discussed here. Although the ability to manipulate a realistic training environment is a particular strength of the approach used, one can imagine that SugarCRM could, for instance, be extended to implement a scoring system, allowing a training regime that is much more game-like to be designed. With such a project, the ability to retrieve data from the SugarCRM database (and thus generate a "game score" for each trainee) would be essential. Using SugarCRM's web services to do so would take advantage of the level of abstraction they provide, keeping the developer from needing to know the details of the SugarCRM database's schema. This thesis shows that at least certain extensions to SugarCRM's functionality are possible, and it is reasonable that the SOA
approach would work in other cases as well as it worked in the case of the Sugar Simulation Manager.

Second, the SOA approach can lend itself to the development of applications that are not at all related to training. For instance, managers could use software that allows them to track a production SugarCRM installation to glean important information about the activity of support staff, salespeople, customers, and marketing campaigns. While SugarCRM contains some reporting features within the core application, the ability to add external analysis tools affords a company a level of flexibility and creativity that might add greatly to the overall value it gains from using SugarCRM. Although the focus within this project was the development of training tools, one of the practical implications of my work might be that other areas of use to an organization implementing SugarCRM could be subject to the sort of functional extension that SSM demonstrates.

Finally, Customer Relationship Management software is far from the only category of application that could be extended using the Service-Oriented approach. Even if we only consider training-oriented development, there are many enterprise-level applications that require significant resources to be brought to bear in training users. For instance, Enterprise Resource Planning and Supply Chain Management systems are often implemented in the same organizations that use CRM, and these complex, expensive systems share the sort of high risks and rewards that CRM presents. Given this, the implementation considerations I have discussed in this thesis might well apply to those types of systems as well, and likewise the ability to customize a training program using the SOA approach might provide similar benefits to those mentioned here. This thesis and the Sugar Simulation Manager application serve as a valuable demonstration of one kind of application of Service-Oriented
Architecture, and future work could pursue how SOA could be used to expand the usefulness of other types of software in ways that the software's creators might not even have considered.

**Other possible research directions.**

In addition to restricting this project only to SugarCRM and the creation of tools for simulation-based training, I also chose only one of several ways to implement SOA—in particular, SOAP-based web services. It is important to note that other alternatives exist. Here I will describe one of those alternatives that might have been pursued and gesture towards other research opportunities that might expand upon the work in this study.

An alternative approach that I might have taken with this project would be to use another technology while still adhering to the Service-Oriented Architecture paradigm. Recent versions of SugarCRM\(^5\) offer an alternative to the SOAP-based web services used here—"RESTful" web services are provided as well, and these offer much the same functionality for SugarCRM as the older SOAP-based ones do. REST (Representational State Transfer) is an architecture for network communication that takes a much simpler approach than the Remote Procedure Call style used by SOAP web services does, by doing away with the additional protocol specification of SOAP, and by limiting interactions to the standard HTTP methods of GET, POST, PUT, and DELETE. Whereas a traditional web service might provide custom functions like "Create" and "RetrieveByID", RESTful web services would instead use those HTTP methods combined with Uniform Resource Identifier logic to perform the same functions—e.g., a POST request to the URI

\(^5\) Versions 5.5 and later.
http://myhost.com/Case would create a new Case, and a GET request to
http://myhost.com/Case/123 would retrieve the Case with the ID value of "123".

The REST approach has some advantages, such as simplicity of architecture, less
bandwidth usage, and simplified security considerations. On the other hand, SOAP
provides developers with the reliability of the WSDL "contract", and REST does not have
the level of formal definition and "best practices" that SOAP does (Kuchtiak, 2006). But
since SugarCRM already provides both types of web service, the question of which to use
for a developer of an application like SSM might come down to other practical
considerations, such as for which type the development is easier. SOAP is known to be
more difficult to work with (and REST's stated advantage is simplicity; Kuchtiak, 2006), but
there are tools that remove much of the complexity of SOAP web services. The
development environment used to create SSM, Visual Studio 2005, automatically creates
proxy classes for web service-exposed methods, provides auto-completion of method names
and parameters, and imports definitions of the web service's custom data types, so a
developer does not necessarily need to look at the WSDL contract at all before beginning to
write code to work with an application like SugarCRM. Given the availability of such tools,
there is not a clear advantage, from a developer's perspective, to using REST. However, if
the SSM project were concerned with manipulating large volumes of data, say for many
thousands of records, or dozens or hundreds of trainees, then the simplicity of the RESTful
approach might prove useful. Web services in general are thought to be slower than other
methods of inter-platform communication, so if the lower bandwidth usage of the RESTful
approach means that the training application may be able to communicate more quickly with
SugarCRM, that in itself may provide a sufficient reason to use those web services instead.
Given these factors, future research might explore the relative advantages and disadvantages of these two methods of leveraging the Service-Oriented Architecture approach. One possibility would be to quantitatively measure the performance benefits or costs of REST vs. SOAP. Such a study might yield useful information about the merits of those two technologies, which could be of particular interest given SOAP's maturity and REST's relative newness (at least in the context of web services). Since the performance of SOAP and REST can be highly dependent on the web server software that implements them, this research could examine the same questions under various popular platforms, such as Apache and Microsoft's Internet Information Server.

Another area that might be fruitfully studied is the relative difficulty of developing tools in each case and the impact of the use of different tools or programming languages (C# vs. Python, for example). Developers need to choose early in the software development lifecycle which technologies to use, and concrete information about the purported ease of development of REST could be valuable in making such decisions. Comparisons of development efficiency on different platforms (and, as mentioned, using different languages) could usefully widen the scope of such a project.

Final Remarks

In conclusion, Service-Oriented Architecture presents opportunities for extending the functionality of enterprise software that can meet many specialized needs. The way in which web services were leveraged in this project to create an application that can manage simulation-based training sessions serves as an example not only of what can be accomplished technologically, but of the significant financial and organizational value those contributions can bring. Further development of a tool like Sugar Simulation Manager could
be of great significance to organizations that need to train their employees on software like SugarCRM, and the approaches taken in this project could be used to extend any number of enterprise applications in ways those applications' authors might never have imagined.
References

Augustin, Larry (2010). "Some Thoughts on Open – From SugarCRM CEO Larry Augustin"


http://cvibanez.myweb.uga.edu/csci8351/projSite/annotationReport%5B1%5D.v1.43.PDF

BerliOS (2010, November 23) "Why Do Linux Distributions Create Bad Forks,"
http://cdrecord.berlios.de/private/linux-dist.html


Prentice-Hall, Inc, Englewood Cliffs
http://blogs.oracle.com/milan/entry/web_services_rest_vs_soap


http://www.computerworld.com/s/article/43487/Application_Programming_Interface


http://www.meritalk.com/uploads_legacy/whitepapers/CORBAtoHydraExpressWPFINA L.pdf

SalesAgility.com, "CRM for Sales Personnel,"


SugarCRM.com, "The Power of Open Source,"
http://www.sugarcrm.com/crm/community/sugarcrm-community.html

http://www.sugarforge.org/projects/opensales/,
http://www.sugarforge.org/projects/qontext/


http://www.w3.org/TR/ws-arch/#whatis

https://www.zivtech.com/community/about-open-source-software-oss
Appendix: Sugar Simulation Manager Code

Note: Program code which is auto-generated by Visual Studio 2005 is not included here.

A. SugarClient.cs

using System;
using System.Data;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Xml.Serialization;
using System.Web.Services;
using System.IO;
using Thesis.sugarcrm;

namespace Thesis
{
    class SugarClient
    {
        public string SugarUserName;
        public string SugarPassword;
        public string SugarURL;
        public bool RandomNames;
        public string DataSourceFile;
        public int CountContact;
        public int CountLead;
        public int CountCase;
        public int CountAccount;
        public int CountOpportunity;
        public string ErrorMessage;

        // global variables for session-related info
        public string sessionId;
        public string error;
        public string user_guid;

        /* arrays for holding names for random generation */
        private string[] MaleFirst;
        private string[] FemaleFirst;
        private string[] Surname;
        private string[] SuffixFile;
        private string[] DataSource;

        int FindFieldID(Thesis.sugarcrm.name_value[] myarr, string fieldname)
        {
            for (int i = 0; i < myarr.Length; i++)
            {
                if (myarr[i].ToString() == fieldname)
return i;
} return -1;

public bool init()
{
    bool valid = true;
    if (RandomNames)
    {
        valid = LoadRandomFiles();
    }

    if (!TestURL(SugarURL))
    {
        valid = false;
    }

    return (valid);
}

private bool TestURL(string URL)
{
    /*
     * Make sure the web service URL is valid and accessible
     * before proceeding
     */
    return (true);
}

public void SetEntryTest()
{
    /* method for testing functions */
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    Thesis.sugarcrm.user_auth user_auth = new Thesis.sugarcrm.user_auth();
    user_auth.user_name = SugarUserName;
    user_auth.password = MD5(SugarPassword);
    Thesis.sugarcrm.set_entry_result login_results =
        ws.login(user_auth, "");
    string session_id = login_results.id;
    string user_guid = ws.get_user_id(session_id);
    Debug.Print (session_id + " : " + user_guid);

    Thesis.sugarcrm.get_entries_count_result en =
        ws.get_entries_count(session_id, "Leads", null, 0);
    Debug.Print (en.result_count.ToString());
    string[] fields = new string[] { "last_name", "first_name"};

    Thesis.sugarcrm.get_entry_list_result se =
        ws.get_entry_list(session_id, "Leads", "(leads.last_name LIKE 's%')", "last_name", 0, fields, 100, 0);
    for (int i = 0; i < se.entry_list.Length; i++)
Debug.Print(se.entry_list[i].id.ToString());
for (int j = 0; j < se.entry_list[i].name_value_list.Length; j++)
{
    if (se.entry_list[i].name_value_list[j].value != "")
    {
        Debug.Print(se.entry_list[i].name_value_list[j].name);
        Debug.Print(se.entry_list[i].name_value_list[j].value);
        //Debug.Print(se.entry_list[i].name_value_list.GetValue(FindFieldID(se.entry_list[i].name_value_list, fields[j])).ToString());
    }
}
ws.Dispose();

public void delete_contact(string contact_id,
Thesis.sugarcrm.sugarsoap ws)
{
    /* given the GUID for a contact, delete it */
    name_value[] fields = new name_value[2];
    fields[0] = new name_value();
    fields[0].name = "id";
    fields[0].value = contact_id;
    fields[1] = new name_value();
    fields[1].name = "deleted";
    fields[1].value = "1";

    ws.set_entry(this.sessionId, "Contacts", fields);
}

public void delete_account(string account_id,
Thesis.sugarcrm.sugarsoap ws)
{
    /* given the GUID for an account, delete it */
    name_value[] fields = new name_value[2];
    fields[0] = new name_value();
    fields[0].name = "id";
    fields[0].value = account_id;
    fields[1] = new name_value();
    fields[1].name = "deleted";
    fields[1].value = "1";

    ws.set_entry(this.sessionId, "Accounts", fields);
}

public void delete_opportunity(string opportunity_id,
Thesis.sugarcrm.sugarsoap ws)
{
/* given the GUID for an opportunity, delete it */
name_value[] fields = new name_value[2];
fields[0] = new name_value();
fields[0].name = "id";
fields[0].value = opportunity_id;
fields[1] = new name_value();
fields[1].name = "deleted";
fields[1].value = "1";
ws.set_entry(this.sessionId, "Opportunities", fields);

public void delete_lead(string lead_id,
Thesis.sugarcrm.sugarsoap ws)
{
/* given the GUID for a lead, delete it */
name_value[] fields = new name_value[2];
fields[0] = new name_value();
fields[0].name = "id";
fields[0].value = lead_id;
fields[1] = new name_value();
fields[1].name = "deleted";
fields[1].value = "1";
ws.set_entry(this.sessionId, "Leads", fields);
}

public void delete_case(string case_id,
Thesis.sugarcrm.sugarsoap ws)
{
/* given the GUID for a case, delete it */
name_value[] fields = new name_value[2];
fields[0] = new name_value();
fields[0].name = "id";
fields[0].value = case_id;
fields[1] = new name_value();
fields[1].name = "deleted";
fields[1].value = "1";
ws.set_entry(this.sessionId, "Cases", fields);
}

public void delete_all()
{
//delete contacts
//delete accounts
//delete leads
//delete opportunities
//delete cases
string[] fields = new string[] { "id" };

DataTable contacts = new DataTable();
DataTable accounts = new DataTable();
DataTable leads = new DataTable();
DataTable opportunities = new DataTable();
DataTable cases = new DataTable();
// separate function to retrieve a DataTable of each entity type. Take a query parameter[] to pass into the search
// in this case, we will use "" for the search so that we retrieve all records
// then loop through the DataTables to delete each one

Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
ws.Url = SugarURL;

contacts = GetTable("Contacts", null, fields, null);
foreach (DataRow x in contacts.Rows)
{
    current_id = x["id"].ToString();
    delete_contact(current_id, ws);
}
Debug.Print("contacts");

accounts = GetTable("Accounts", null, fields, null);
foreach (DataRow x in accounts.Rows)
{
    current_id = x["id"].ToString();
    delete_account(current_id, ws);
}
Debug.Print("accounts");

leads = GetTable("Leads", null, fields, null);
foreach (DataRow x in leads.Rows)
{
    current_id = x["id"].ToString();
    delete_lead(current_id, ws);
}
Debug.Print("leads");

opportunities = GetTable("Opportunities", null, fields, null);
foreach (DataRow x in opportunities.Rows)
{
    current_id = x["id"].ToString();
    delete_opportunity(current_id, ws);
}
Debug.Print("opportunities");

cases = GetTable("Cases", null, fields, null);
foreach (DataRow x in cases.Rows)
{
    current_id = x["id"].ToString();
    delete_case(current_id, ws);
}
Debug.Print("cases");

ws.Dispose();

// get a DataTable of the specified type
public DataTable GetTable(string type, string query, string[] fields, string OrderBy)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    DataTable results = new DataTable();
    int Offset = 0;
    int MaxResults = 10000;
    if (query == null) query = "";
    if (OrderBy == null) OrderBy = "";

    //Define the Columns
    foreach (string field in fields)
    {
        results.Columns.Add(field);
        Debug.Print("field: " + field.ToString());
    }

    //Get a list of entries
    get_entry_list_result entryList =
    ws.get_entry_list(this.sessionId, type, query, OrderBy, Offset,
    fields, MaxResults, 0);

    //Loop trough the entries
    foreach (entry_value entry in entryList.entry_list)
    {
        //Create a new DataRow
        DataRow meeting = results.NewRow();

        //Loop trough the columns
        foreach (name_value value in entry.name_value_list)
        {
            meeting[value.name] = value.value;
        }

        //Add the DataRow to the DataTable
        results.Rows.Add(meeting);
    }

    return results;
}

/* get a count of each entity type currently in the DB (not
including deleted records) */
public string[,] GetCount()
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    //create array for holding results
    //columns are hardcoded
    string[,] results = new string[5,2];
    results[0, 0] = "Contacts";
    results[1, 0] = "Accounts";
    results[2, 0] = "Leads";
results[3, 0] = "Opportunities";
results[4, 0] = "Cases";

get_entries_count_result cnt;
for (int i = 0; i <= 4; i++)
{
    cnt = ws.get_entries_count(this.sessionId, results[i, 0], ",", 0);
    results[i, 1] = cnt.result_count.ToString();
}
return results;

// attempt to authenticate the given username and password
// return true or false
public bool Authenticate()
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    Thesis.sugarcrm.user_auth user_auth = new Thesis.sugarcrm.user_auth();
    user_auth.user_name = SugarUserName;
    user_auth.password = MD5(SugarPassword);

    set_entry_result login_results = ws.login(user_auth, ",");

    //Check for errors
    if (Convert.ToInt32(login_results.error.number) != 0)
    {
        //An error occured
        this.error = String.Concat(login_results.error.name, ":
                                    ", login_results.error.description);
        //Clear the existing sessionId
        this.sessionId = String.Empty;
    }
    else
    {
        //Set the sessionId
        this.sessionId = login_results.id;
        this.user_guid = ws.get_user_id(this.sessionId);

        //Clear the existing error
        this.error = String.Empty;
    }

    //Return the boolean
    return (this.sessionId != String.Empty);
}

public bool LoadRandomFiles()
{
/*
* Loads a dictionary of names for use in randomly generating SugarCRM entities
* the files should be present in the application directory, and must be named
* "MaleFirst.txt", "FemaleFirst.txt", and "Surnames.txt"
*/
bool valid = true;
string MaleFirstFile = "MaleFirst.txt";
string FemaleFirstFile = "FemaleFirst.txt";
string SurnamesFile = "Surnames.txt";
string CompanySuffixFile = "CompanySuffix.txt";

if (!File.Exists(MaleFirstFile)) {
    valid = false;
} else {
    string MaleFirst = File.ReadAllText(MaleFirstFile);
    MaleFirst = MaleFirst.Split('
');
    MaleFirstFile.Close();
}

if (!File.Exists(FemaleFirstFile)) {
    valid = false;
} else {
    string FemaleFirst = File.ReadAllText(FemaleFirstFile);
    FemaleFirst = FemaleFirst.Split('
');
    FemaleFirstFile.Close();
}

if (!File.Exists(SurnamesFile)) {
    valid = false;
} else {
    string Surname = File.ReadAllText(SurnamesFile);
    Surname = Surname.Split('
');
    SurnamesFile.Close();
}

if (!File.Exists(CompanySuffixFile)) {
    valid = false;
} else {
    string CompanySuffix = File.ReadAllText(CompanySuffixFile);
    CompanySuffix = CompanySuffix.Split('
');
    CompanySuffixFile.Close();
}

if (valid) {
    // only proceed if all files exist
    return (valid);
    }

public bool LoadData()
{
    int male;
    Random rnd = new Random();
    string FirstName;
    string LastName;

    }
string Suffix;
string Email;
int amt;
string Phone;
string Website;
bool success;
string acc; // id of random account

if (RandomNames)
{
    try
    {
        /* create accounts */
        for (int i = 1; i <= CountAccount; i++)
        {
            /*
            male = rnd.Next(1);
            if (male == 1)
            {
                FirstName = MaleFirst[rnd.Next(MaleFirst.Length)].TrimEnd();
            }
            else
            {
                FirstName = FemaleFirst[rnd.Next(FemaleFirst.Length)].TrimEnd();
            }
            */
            LastName = Surname[rnd.Next(Surname.Length)].TrimEnd();
            Suffix = SuffixFile[rnd.Next(SuffixFile.Length)].TrimEnd().ToUpper();
            success = create_account(LastName + " " + Suffix, "555-555-5555", "www." + LastName + ".com");
            if (!success)
                throw new UnauthorizedAccessException("Permissions error creating account.");
        }

        /* create contacts */
        for (int i = 1; i <= CountContact; i++)
        {
            // randomly choose gender each time
            male = rnd.Next(1);
            if (male == 1)
            {
                FirstName = MaleFirst[rnd.Next(MaleFirst.Length)].TrimEnd();
            }
            else
            {
                FirstName = FemaleFirst[rnd.Next(FemaleFirst.Length)].TrimEnd();
            }
        }
    }
}
LastName = Surname[rnd.Next(Surname.Length)].TrimEnd();
Email = FirstName.Substring(0, 1) + LastName + 
"@email.com";
acc = get_random_account();
success = create_contact(FirstName, LastName, 
Email, acc);
if (!success)
    throw new
UnauthorizedAccessException("Permissions error creating 
contact.");
}

/* create leads */
for (int i = 1; i <= CountLead; i++)
{
    // randomly choose gender each time
    male = rnd.Next(1);
    if (male == 1)
    {
        FirstName = MaleFirst[rnd.Next(MaleFirst.Length)].TrimEnd();
    }
    else
    {
        FirstName = FemaleFirst[rnd.Next(FemaleFirst.Length)].TrimEnd();
    }

    LastName = Surname[rnd.Next(Surname.Length)].TrimEnd();
    Email = FirstName.Substring(0, 1) + LastName + 
"@email.com";
    acc = get_random_account_name();
    success = create_lead(FirstName, LastName, 
Email, acc);
    if (!success)
        throw new
UnauthorizedAccessException("Permissions error creating lead.");
}

/* create opportunities */
for (int i = 1; i <= CountOpportunity; i++)
{
    // randomly choose gender each time
    male = rnd.Next(1);
    if (male == 1)
    {
        FirstName = MaleFirst[rnd.Next(MaleFirst.Length)].TrimEnd();
    }
    else
    {
        FirstName = FemaleFirst[rnd.Next(FemaleFirst.Length)].TrimEnd();
    }
LastName = Surname[rnd.Next(Surname.Length)].TrimEnd();

    // Amount is a random number from 0 to 100,000
    amt = rnd.Next(100000);  
    acc = get_random_account();
    success = create_opportunity(FirstName + " " +
    LastName, amt.ToString(), acc);
    if (!success)
        throw new
        UnauthorizedAccessException("Permissions error creating opportunity.");
    }

    /* create cases */
    for (int i = 1; i <= CountCase; i++)
    {
        // randomly choose gender each time
        male = rnd.Next(1);  
        if (male == 1)
            {
                FirstName = MaleFirst[rnd.Next(MaleFirst.Length)].TrimEnd();
            }
        else
            {
                FirstName = FemaleFirst[rnd.Next(FemaleFirst.Length)].TrimEnd();
            }
        LastName = Surname[rnd.Next(Surname.Length)].TrimEnd();
        acc = get_random_account();
        success = create_case(FirstName + " " +
        LastName, acc);
        if (!success)
            throw new
            UnauthorizedAccessException("Permissions error creating case.");
        return (true);
    }

    catch (UnauthorizedAccessException ex)
    {
        ErrorMessage = ex.Message;
        return (false);
    }

    }/* use file to import data */
    
    else /* use file to import data */
    {
        if (!File.Exists(DataSourceFile))
        {
            ErrorMessage = "Data source file does not exist or is not accessible.";
            return(false);
        }
        StreamReader strm1 = new StreamReader(DataSourceFile);


string data = strm1.ReadToEnd();
DataSource = data.Split('n');
strm1.Close();

string[] curr;
//loop through each line and create the indicated record type
for (int i = 0; i < DataSource.Length; i++)
{
    curr = DataSource[i].Split(',');
    switch (curr[0])
    {
        case "Account":
            FirstName = curr[1];
            Phone = curr[2];
            Website = curr[3];
            create_account(FirstName, Phone, Website);
            break;
        case "Lead":
            FirstName = curr[1];
            LastName = curr[2];
            Email = curr[3];
            create_lead(FirstName, LastName, Email,
            amt.ToString(), null);
            break;
        case "Opportunity":
            FirstName = curr[1];
            amt = Convert.ToInt32(curr[2]);
            create_opportunity(FirstName,
            Email);
            break;
        case "Case":
            FirstName = curr[1];
            create_case(FirstName, null);
            break;
        default:
            ErrorMessage = "Unhandled line in data file encountered."
            return (false);
    }
    /* process completed, to return true */
    return (true);
}

/* create a contact record
* return true or false */
public bool create_contact(string FirstName, string LastName,
string Email, string accountid)
{
  Thesis.sugarcrm.sugarsoap ws = new
  Thesis.sugarcrm.sugarsoap();
  ws.Url = SugarURL;
  string wsr = ws.create_contact(SugarUserName,
  MD5(SugarPassword), FirstName, LastName, Email);
  //Debug.Print("contact: " + wsr.ToString());

  //create relationship with random account
  set_relationship_value srv = new set_relationship_value();
  srv.module1 = "Contacts";
  srv.module1_id = wsr;
  srv.module2 = "Accounts";
  srv.module2_id = accountid;
  string session_id = get_user_auth();
  error_value result = ws.set_relationship(session_id, srv);
  //Debug.Print("RESULT: " + result.description);  
  ws.Dispose();
  return (!(wsr.ToString() == "0"));
}

/* create a lead record
 * return true or false */
public bool create_lead(string FirstName, string LastName,
 string Email, string accountname)
{
  string session_id = get_user_auth();

  Thesis.sugarcrm.sugarsoap ws = new
  Thesis.sugarcrm.sugarsoap();
  ws.Url = SugarURL;
  string wsr = ws.create_lead(SugarUserName,
  MD5(SugarPassword), FirstName, LastName, Email);
  //Debug.Print(wsr.ToString());
  get_entry_result l = ws.get_entry(session_id, "Leads", wsr,
  null);

  //create relationship with random account (name only)
  name_value[] fields = new name_value[2];
  fields[0] = new name_value();
  fields[0].name = "id";
  fields[0].value = wsr;
  fields[1] = new name_value();
  fields[1].name = "account_name";
  fields[1].value = accountname;
  set_entry_result ser = ws.set_entry(session_id, "Leads",
  fields);
  ws.Dispose();
  return (!(wsr.ToString() == "0"));
}

// create a Sugar service session and return the session id
public string get_user_auth() {

Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
ws.Url = SugarURL;

Thesis.sugarcrm.user_auth user_auth = new Thesis.sugarcrm.user_auth();
user_auth.user_name = SugarUserName;
user_auth.password = MD5(SugarPassword);

Thesis.sugarcrm.set_entry_result login_results = ws.login(user_auth, "");
string session_id = login_results.id;
return session_id;

/* create an opportunity record
* return true or false */
public bool create_opportunity(string Name, string Amount,
string accountid)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    string wsr = ws.create_opportunity(SugarUserName,
    MD5(SugarPassword), Name,
    Amount,
    accountid);

    //create relationship with random account
    set_relationship_value srv = new set_relationship_value();
    srv.module1 = "Opportunities";
    srv.module1_id = wsr;
    srv.module2 = "Accounts";
    srv.module2_id = accountid;
    string session_id = get_user_auth();
    error_value result = ws.set_relationship(session_id, srv);
    //Debug.Print("RESULT: " + result.description);
    ws.Dispose();
    return (!(wsr.ToString() == "0");
}

/* create a case record
* return true or false */
public bool create_case(string Name, string accountid)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    string wsr = ws.create_case(SugarUserName,
    MD5(SugarPassword), Name);
    //Debug.Print(wsr.ToString());
    //create relationship with random account
    set_relationship_value srv = new set_relationship_value();
    srv.module1 = "Cases";
    srv.module1_id = wsr;
    srv.module2 = "Accounts";
    string session_id = get_user_auth();
    error_value result = ws.set_relationship(session_id, srv);
    //Debug.Print("RESULT: " + result.description);
    ws.Dispose();
    return (!(wsr.ToString() == "0");
}
srv.module2_id = accountid;
string session_id = get_user_auth();
error_value result = ws.set_relationship(session_id, srv);
//Debug.Print("RESULT: " + result.description);

ws.Dispose();
return !(wsr.ToString() == "0");

/* create an account record
 * return true or false */
public bool create_account(string Name, string Phone, string Website)
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    string wsr = ws.create_account(SugarUserName, MD5(SugarPassword), Name, Phone, Website);
    //Debug.Print(wsr.ToString());
    ws.Dispose();
    return !(wsr.ToString() == "0");
}

// get list of 100 accounts, and grab a random one
// returns the id
public string get_random_account()
{
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    string session_id = get_user_auth();

    string[] modules = new string[1];
    modules[0] = "Accounts";

    Random rnd = new Random();

    get_entry_list_result r = ws.get_entry_list(session_id, "Accounts", "", "", 0, null, 100, 0);

    int len = r.entry_list.Length;
    int ix = rnd.Next(len);
    string id = r.entry_list[ix].id;

    ws.Dispose();
    return id;
}

// get list of 100 accounts, and grab a random one
// returns the name
public string get_random_account_name()
{
    string accname;
    accname = "";

    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    string session_id = get_user_auth();

    string[] modules = new string[1];
    modules[0] = "Accounts";

    Random rnd = new Random();

    get_entry_list_result r = ws.get_entry_list(session_id, "Accounts", "", "", 0, null, 100, 0);

    int len = r.entry_list.Length;
    int ix = rnd.Next(len);
    string id = r.entry_list[ix].id;

    ws.Dispose();
    return id;
}
Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
ws.Url = SugarURL;

string session_id = get_user_auth();

string[] modules = new string[1];
modules[0] = "Accounts";

// generate random number and get random entry from list
Random rnd = new Random();

get_entry_list_result r = ws.get_entry_list(session_id, "Accounts", "", "", 0, null, 100, 0);

int len = r.entry_list.Length;
int ix = rnd.Next(len);

foreach (name_value x in r.entry_list[ix].name_value_list)
{
    if (x.name == "name")
    {
        accname = x.value;
    }
}

ws.Dispose();
return accname;

public void search(string Name)
{
    /* not currently useful */
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;
    Thesis.sugarcrm.contact_detail[] s = ws.search(SugarUserName, MD5(SugarPassword), Name);
}

public bool set_relationship(string Session, Thesis.sugarcrm.set_relationship_value Set_Relationship_Value)
{
    /* not functional */
    Thesis.sugarcrm.sugarsoap ws = new Thesis.sugarcrm.sugarsoap();
    ws.Url = SugarURL;

    return (true);
}

// hashing function to allow login to Sugar services
public string MD5(string StringToHash)
{
    MD5 hasher = MD5CryptoServiceProvider.Create();
byte[] data = hasher.ComputeHash(Encoding.Default.GetBytes(StringToHash));

StringBuilder sBuilder = new StringBuilder();
for (int i = 0; i < data.Length; i++)
{
    sBuilder.Append(data[i].ToString("x2"));
}
//Debug.Print(sBuilder.ToString());
return sBuilder.ToString();

namespace Thesis
{
    public partial class frmLoader : Form
    {
    }
}

B. Form1.cs

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace Thesis
{
    public partial class frmLoader : Form
    {
        public frmLoader()
        {
            InitializeComponent();
        }

        /* toggle the visibility of the password field */
        private void chkDisplayPassword_CheckedChanged(object sender, EventArgs e)
        {
            if (chkDisplayPassword.Checked)
            {
                txtPassword.PasswordChar = '\0';
            }
            else
            {
                txtPassword.PasswordChar = '*';
            }
        }

        private void btnLoad_Click(object sender, EventArgs e)
        {
            bool valid = true;
            /* check that options are filled out properly */
            if (radSourceFile.Checked && txtDataSourceFile.Text == "")
            {

            }
        }
    }
}
MessageBox.Show("No source file selected.");
valid = false;
return;
}
if (txtUsername.Text == "")
{
    MessageBox.Show("No username entered.");
    valid = false;
    return;
}
if (txtPassword.Text == "")
{
    MessageBox.Show("No password entered.");
    valid = false;
    return;
}
if (txtSugarURL.Text == "")
{
    MessageBox.Show("No URL entered.");
    valid = false;
    return;
}
if (radSourceRandom.Checked &&
    txtQtyContact.Text == "" &&
    txtQtyAccount.Text == "" &&
    txtQtyCase.Text == "" &&
    txtQtyLead.Text == "" &&
    txtQtyOpportunity.Text == "")
{
    MessageBox.Show("Random data selected, but no quantities entered.");
    valid = false;
    return;
}
/* end validation */

/* save settings in the application settings (registry) */
lblStatus.Text = "Saving settings...";
Thesis.frmLoader ActiveForm.Refresh();
    txtSugarURL.Text;

/* proceed with the data load */
lblStatus.Text = "Preparing Data Load...";
Thesis.frmLoader ActiveForm.Refresh();
SugarClient cs = new SugarClient();
/* set the configured values */
cs.SugarUserName = txtUsername.Text;
cs.SugarPassword = txtPassword.Text;
cs.SugarURL = txtSugarURL.Text;
cs.RandomNames = radSourceRandom.Checked;
cs.DataSourceFile = txtDataSourceFile.Text;
if (cs.RandomNames)
}

/* execute the data load */
valid = cs.init();
if (!valid)
{
    lblStatus.Text = "Initialization failed.\nIf generating randomly,\ncheck for existence of name files.";
    return;
}

valid = cs.LoadData();
if (valid)
    lblStatus.Text = "Data Load Complete.";
else
    lblStatus.Text = cs.ErrorMessage;
}

/* fill the record type fields with the given integer */
private void FillValues(string Amount)
{
    txtQtyContact.Text = Amount;
    txtQtyAccount.Text = Amount;
    txtQtyCase.Text = Amount;
    txtQtyLead.Text = Amount;
    txtQtyOpportunity.Text = Amount;
}

private void frmLoader_Load(object sender, EventArgs e)
{
}

/* when the file browser field is clicked, display a dialog * to select the file */
private void radSourceFile_CheckedChanged(object sender, EventArgs e)
{
    if (radSourceFile.Checked)
    {
       dlgDataSource.Filter = "Text or CSV File|*txt;*.csv";
       dlgDataSource.ShowDialog();
        txtDataSourceFile.Text = dlgDataSource.FileName;
        dlgDataSource.Dispose();
    }
else
{
    txtDataSourceFile.Text = "";
}

/* clear the file text box */
private void radSourceRandom_CheckedChanged(object sender, EventArgs e)
{
    if (radSourceRandom.Checked)
        txtDataSourceFile.Text = "";
}

/* delete all records in the database */
private void linkDeleteAll_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    if (MessageBox.Show("Are you sure you want to delete all records?", "Confirm Delete", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
        lblStatus.Text = "Deleting records...";
        Thesis.frmLoader.ActiveForm.Refresh();
        DeleteAll();
        lblStatus.Text = "Delete completed.";
        //Thesis.frmLoader.ActiveForm.Close();
    }
}

/* retrieve the current record count *
* the GetCount() method formats the results*
* and returns them to lblStatus */
private void linkGetRecordCount_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    lblStatus.Text = "Retrieving record count...";
    Thesis.frmLoader.ActiveForm.Refresh();
    lblStatus.Text = GetCount();
}

private void DeleteAll()
{
    SugarClient cs = new SugarClient();
    /* set the configured values */
    cs.SugarUserName = txtUsername.Text;
    cs.SugarPassword = txtPassword.Text;
    cs.SugarURL = txtSugarURL.Text;
    cs.Authenticate();
    cs.delete_all();
}

private string GetCount()
{
    string CountText;
    try
SugarClient cs = new SugarClient();
/* set the configured values */
cs.SugarUserName = txtUsername.Text;
cs.SugarPassword = txtPassword.Text;
cs.SugarURL = txtSugarURL.Text;
bool auth = cs.Authenticate();

if (!auth) throw new InvalidOperationException("Failed to login to SugarCRM web services.");

string[,] Count;
Count = cs.GetCount();

CountText = "Current Record Count\n";
for (int i = 0; i <= Count.GetUpperBound(0); i++)
{
    CountText += (Count[i, 0] + ": " + Count[i, 1] + "\n");
}
catch (Exception e)
{
    //something something
    CountText = "Error: " + e.Message;
}
return CountText;

/* preset value: 25 */
private void linkSmall_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    FillValues("25");
}

/* preset value: 100 */
private void linkMedium_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    FillValues("100");
}

/* preset value: 500 */
private void linkLarge_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    FillValues("500");
}

/* run the User Status report */
private void linkRptUserStatus_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    UserStatusForm frm = new UserStatusForm();
    lblStatus.Text = "Opening report...";
    this.Refresh();
System.Threading.Thread.Sleep(2000);
lblStatus.Text += "\nDone."
frm.Visible = true;
}

/* run the Account report */
private void linkRptAccount_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    ActForm frm = new ActForm();
    lblStatus.Text = "Opening report...";
    this.Refresh();
    System.Threading.Thread.Sleep(2000);
    lblStatus.Text += "\nDone."
    frm.Visible = true;
}

/* run the User Comparison report */
private void linkRptUserComparison_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    UserCompForm frm = new UserCompForm();
    lblStatus.Text = "Opening report...";
    this.Refresh();
    System.Threading.Thread.Sleep(2000);
    lblStatus.Text += "\nDone."
    frm.Visible = true;
}

/* run the Cases report */
private void linkRptCases_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    MissedForm frm = new MissedForm();
    lblStatus.Text = "Opening report...";
    this.Refresh();
    System.Threading.Thread.Sleep(2000);
    lblStatus.Text += "\nDone."
    frm.Visible = true;
}
C. Sugar Simulation Manager Name Files

Note that in the Sugar Simulation Manager project, these files are delimited by line feeds. In this appendix, they are comma-delimited in the interest of space.

**Surnames.txt**

SMITH, JOHNSON, WILLIAMS, JONES, BROWN, DAVIS, MILLER, WILSON, MOORE, TAYLOR, ANDERSON, THOMAS, JACKSON, WHITE, HARRIS, MARTIN, THOMPSON, GARCIA, MARTINEZ, ROBINSON, CLARK, RODRIGUEZ, LEWIS, LEE, WALKER, HALL, ALLEN, YOUNG, HERNANDEZ, KING, WRIGHT, LOPEZ, HILL, SCOTT, GREEN, ADAMS, BAKER, GONZALEZ, NELSON, CARTER, MITCHELL, PEREZ, ROBERTS, TURNER, PHILLIPS, CAMPBELL, PARKER, EVANS, EDWARDS, COLLINS, STEWART, SANCHEZ, MORRIS, ROGERS, REED, COOK, MORGAN, BELL, MURPHY, BAILEY, RIVERA, COOPER, RICHARDSON, COX, HOWARD, WARD, TORRES, PETERSON, GRAY, RAMIREZ, JAMES, WATSON, BROOKS, KELLY, SANDERS, PRICE, BENNETT, WOOD, BARNES, ROSS, HENDERSON, COLEMAN, JENKINS, PERRY, POWELL, LONG, PATTERSON, HUGHES, FLORES, WASHINGTON, BUTLER, SIMMONS, FOSTER, GONZALES, BRYANT, ALEXANDER, RUSSELL, GRIFFIN, DIAZ, HAYES, MYERS, FORD, HAMILTON, GRAHAM, SULLIVAN, WALLACE, WOODS, COLE, WEST, JORDAN, OWENS, REYNOLDS, FISHER, ELLIS, HARRISON, GIBSON, MCDONALD, CRUZ, MARSHALL, ORTIZ, GOMEZ, MURRAY, FREEMAN, WELLS, WEBB, SIMPSON, STEVENS, TUCKER, PORTER, HUNTER, HICKS, CRAWFORD, HENRY, BOYD, MASON, MORALES, KENNEDY, WARREN, DIXON, RAMOS,
REYES, BURNS, GORDON, SHAW, HOLMES, RICE, ROBERTSON, HUNT, BLACK,
DANIELS, PALMER, MILLS, NICHOLS, GRANT, KNIGHT, FERGUSON, ROSE,
STONE, HAWKINS, DUNN, PERKINS, HUDSON, SPENCER, GARDNER,
STEPHENS, PAYNE, PIERCE, BERRY, MATTHEWS, ARNOLD, WAGNER, WILLIS,
RAY, WATKINS, OLSON, CARROLL, DUNCAN, SNYDER, HART, CUNNINGHAM,
BRADLEY, LANE, ANDREWS, RUIZ, HARPER, FOX, RILEY, ARMSTRONG,
CARPENTER, WEAVER, GREENE, LAWRENCE, ELLIOTT, CHAVEZ, SIMS,
AUSTIN, PETERS, KELLEY, FRANKLIN, LAWSON, FIELDS, GUTIERREZ, RYAN,
SCHMIDT, CARR, VASQUEZ, CASTILLO, WHEELER, CHAPMAN, OLIVER,
MONTGOMERY, RICHARDS, WILLIAMSON, JOHNSTON, BANKS, MEYER,
BISHOP, MCCOY, HOWELL, ALVAREZ, MORRISON, HANSEN, FERNANDEZ,
GARZA, HARVEY, LITTLE, BURTON, STANLEY, NGUYEN, GEORGE, JACOBS,
REID, KIM, FULLER, LYNCH, DEAN, GILBERT, GARRETT, ROMERO, WELCH,
LARSON, FRAZIER, BURKE, HANSON, DAY, MENDOZA, MORENO, BOWMAN,
MEDINA, FOWLER, BREWER, HOFFMAN, CARLSON, SILVA, PEARSON,
HOLLAND, DOUGLAS, FLEMING, JENSEN, VARGAS, BYRD, DAVIDSON,
HOPKINS, MAY, TERRY, HERRERA, WADE, SOTO, WALTERS, CURTIS, NEAL,
Caldwell, Lowe, Jennings, Barnett, Graves, Jimenez, Horton,
SHELTON, BARRETT, O'BRIEN, CASTRO, SUTTON, GREGORY, MCKINNEY,
LUCAS, MILES, CRAIG, RODRIGUEZ, CHAMBERS, HOLT, LAMBERT,
FLETCHER, WATTS, BATES, HALE, RHODES, PENA, BECK, NEWMAN,
HAYNES, MCDANIEL, MENDEZ, BUSH, VAUGHN, PARKS, DAWSON,
SANTIAGO, NORRIS, HARDY, LOVE, STEELE, CURRY, POWERS, SCHULTZ,
BARKER, GUZMAN, PAGE, MUNOZ, BALL, KELLER, CHANDLER, WEBER,
LEONARD, WALSH, LYONS, RAMSEY, WOLFE, SCHNEIDER, MULLINS,
BENSON, SHARP, BOWEN, DANIEL, BARBER, CUMMINGS, HINES, BALDWIN,
GRIFFITH, VALDEZ, HUBBARD, SALAZAR, REEVES, WARNER, STEVENSON,
BURGESS, SANTOS, TATE, CROSS, GARNER, MANN, MACK, MOSS,
THORNTON, DENNIS, MCGEE, FARMER, DELGADO, AGUILAR, VEGA,
GLOVER, MANNING, COHEN, HARMON, RODGERS, ROBBINS, NEWTON,
TODD, BLAIR, HIGGINS, INGRAM, REESE, CANNON, STRICKLAND,
TOWNSEND, POTTER, GOODWIN, WALTON, ROWE, HAMPTON, ORTEGA,
PATTON, SWANSON, JOSEPH, FRANCIS, GOODMAN, MALDONADO, YATES,
BECKER, ERICKSON, HODGES, RIOS, CONNER, ADKINS, WEBSTER, NORMAN,
MALONE, HAMMOND, FLOWERS, COBB, MOODY, QUINN, BLAKE, MAXWELL,
POPE, FLOYD, OSBORNE, PAUL, MCCARTHY, GUERRERO, LINDSEY,
ESTRADA, SANDOVAL, GIBBS, TYLER, GROSS, FITZGERALD, STOKES, DOYLE,
SHERMAN, SAUNDERS, WISE, COLON, GILL, ALVARADO, GREER, PADILLA,
SIMON, WATERS, NUNEZ, BALLARD, SCHWARTZ, MCBRIDE, HOUSTON,
CHRISTENSEN, KLEIN, PRATT, BRIGGS, PARSONS, MCLAUGHLIN,
ZIMMERMAN, FRENCH, BUCHANAN, MORAN, COPELAND, ROY, PITTMAN,
BRADY, MCCORMICK, HOLLOWAY, BROCK, POOLE, FRANK, LOGAN, OWEN,
BASS, MARSH, DRAKE, WONG, JEFFERSON, PARK, MORTON, ABBOTT,
SPARKS, PATRICK, NORTON, HUFF, CLAYTON, MASSEY, LLOYD, FIGUEROA,
CARSON, Bowers, ROBERSON, BARTON, TRAN, LAMB, HARRINGTON,
CASEY, BOONE, CORTEZ, CLARKE, MATHIS, SINGLETON, WILKINS, CAIN,
BRYAN, UNDERWOOD, HOGAN, MCKENZIE, COLLIER, LUNA, PHELPS,
MCGUIRE, ALLISON, BRIDGES, WILKERSON, NASH, SUMMERS, ATKINS,
WILCOX, PITTS, CONLEY, MARQUEZ, BURNETT, RICHARD, COCHRAN,
CHASE, DAVENPORT, HOOD, GATES, CLAY, AYALA, SAWYER, ROMAN,
VAZQUEZ, DICKERSON, HODGE, ACOSTA, FLYNN, ESPINOZA, NICHOLSON,
MONROE, WOLF, MORROW, KIRK, RANDALL, ANTHONY, WHITAKER,
OCONNOR, SKINNER, WARE, MOLINA, KIRBY, HUFFMAN, BRADFORD,
CHARLES, GILMORE, DOMINGUEZ, ONEAL, BRUCE, LANG, COMBS, KRAMER,
HEATH, HANCOCK, GALLAGHER, GAINES, SHAFFER, SHORT, WIGGINS,
MATHEWS, MCCLAIN, FISCHER, WALL, SMALL, MELTON, HENSLEY, BOND,
DYER, CAMERON, GRIMES, CONTRERAS, CHRISTIAN, WYATT, BAXTER,
SNOW, MOSLEY, SHEPHERD, LARSEN, HOOVER, BEASLEY, GLENN,
PETERSEN, WHITEHEAD, MEYERS, KEITH, GARRISON, VINCENT, SHIELDS,
HORN, SAVAGE, OLSEN, SCHROEDER, HARTMAN, WOODARD, MUELLER,
KEMP, DELEON, BOOTH, PATEL, CALHOUN, WILEY, EATON, CLINE,
NAVARRO, HARRELL, LESTER, HUMPHREY, PARRISH, DURAN, HUTCHINSON,
HESS, DORSEY, BULLOCK, ROBLES, BEARD, DALTON, AVILA, VANCE, RICH,
BLACKWELL, YORK, JOHNS, BLANKENSHIP, TREVINO, SALINAS, CAMPOS,
PRUITT, MOSES, CALLAHAN, GOLDEN, MONTOYA, HARDIN, GUERRA,
MCDOWELL, CAREY, STAFFORD, GALLEGOS, HENSON, WILKINSON,
BOOKER, MERRITT, MIRANDA, ATKINSON, ORR, DECKER, HOBBS, PRESTON,
TANNER, KNOX, PACHECO, STEPHENSON, GLASS, ROJAS, SERRANO, MARKS,
HICKMAN, ENGLISH, SWEENEY, STRONG, PRINCE, MCCLURE, CONWAY,
WALTER, ROTH, MAYNARD, FARRELL, LOWERY, HURST, NIXON, WEISS,
TRUJILLO, ELLISON, SLOAN, JUAREZ, WINTERS, MCLEAN, RANDOLPH,
LEON, BOYER, VILLARREAL, MCCALL, GENTRY, CARRILLO, KENT, AYERS,
LARA, SHANNON, SEXTON, PACE, HULL, LEBLANC, BROWNING,
VELASQUEZ, LEACH, CHANG, HOUSE, SELLERS, HERRING, NOBLE, FOLEY,
BARTLETT, MERCADO, LANDRY, DURHAM, WALLS, BARR, MCKEE, BAUER,
RIVERS, EVERETT, BRADSHAW, PUGH, VELEZ, RUSH, ESTES, DODSON,
MORSE, SHEPPARD, WEEKS, CAMACHO, BEAN, BARRON, LIVINGSTON,
MIDDLETON, SPEARS, BRANCH, BLEVINS, CHEN, KERR, MCONNELL,
HATFIELD, HARDING, ASHLEY, SOLIS, HERMAN, FROST, GILES, BLACKBURN,
WILLIAM, PENNINGTON, WOODWARD, FINLEY, MCINTOSH, KOCH, BEST,
SOLOMON, MCCULLOUGH, DUDLEY, NOLAN, BLANCHARD, RIVAS,
BRENNAN, MEJIA, KANE, BENTON, JOYCE, BUCKLEY, HALEY, VALENTINE,
MADDOX, RUSSO, MCKNIGHT, BUCK, MOON, MCMILLAN, CROSBY, BERG,
DOTSON, MAYS, ROACH, CHURCH, CHAN, RICHMOND, MEADOWS,
FAULKNER, ONEILL, KNAPP, KLINE, BARRY, OCHOA, JACOBSON, GAY,
AVERY, HENDRICKS, HORNE, SHEPARD, HEBERT, CHERRY, CARDENAS,
MCINTYRE, WHITNEY, WALLER, HOLMAN, DONALDSON, CANTU, TERRELL,
MORIN, GILLESPIE, FUENTES, TILLMAN, SANFORD, BENTLEY, PECK, KEY,
SALAS, ROLLINS, GAMBLE, DICKSON, BATTLE, SANTANA, CABRERA,
CERVANTES, HOWE, HINTON, HURLEY, SPENCE, ZAMORA, YANG, MCNEIL,
SUAREZ, CASE, PETTY, GOULD, MCFARLAND, SAMPSON, CARVER, BRAY,
ROSARIO, MACDONALD, STOUT, HESTER, MELENDEZ, DILLON, FARLEY,
HOPPER, GALLOWAY, POTTS, BERNARD, JOYNER, STEIN, AGUIRRE,
OSBORNE, MERCER, BENDER, FRANCO, ROWLAND, SYKES, BENJAMIN,
TRAVIS, PICKETT, CRANE, SEARS, MAYO, DUNLAP, HAYDEN, WILDER,
MCKAY, COFFEY, MCCARTY, EWING, COOLEY, VAUGHAN, BONNER,
COTTON, HOLDER, STARK, FERRELL, CANTRELL, FULTON, LYNN, LOTT,
CALDERON, ROSA, POLLARD, HOOPER, BURCH, MULLEN, FRY, RIDDLE,
LEVY, DAVID, DUKE, ODONNELL, GUY, MICHAEL, BRITT, FREDERICK,
DAUGHERTY, BERGER, DILLARD, ALSTON, JARVIS, FRYE, RIGGS, CHANEY,
ODOM, DUFFY, FITZPATRICK, VALENZUELA, MERRILL, MAYER, ALFORD,
MCPHERSON, ACEVEDO, DONOVAN, BARRERA, ALBERT, COTE, REILLY,
COMPTON, RAYMOND, MOONEY, MCGOWAN, CRAFT, CLEVELAND,
CLEMONS, WYNN, NIELSEN, BAIRD, STANTON, SNIDER, ROSALES, BRIGHT,
WITT, STUART, HAYS, HOLDEN, RUTLEDGE, KINNEY, CLEMENTS,
CASTANEDA, SLATER, HAHN, EMERSON, CONRAD, BURKS, DELANEY, PATE,
LANCASTER, SWEET, JUSTICE, TYSON, SHARPE, WHITFIELD, TALLEY,
MACIAS, IRWIN, BURRIS, RATLIFF, MCCRAY, MADDEN, KAUFMAN, BEACH,
GOFF, CASH, BOLTON, MCFADDEN, LEVINE, GOOD, BYERS, KIRKLAND,
KIDD, WORKMAN, CARNEY, DALE, MCLEOD, HOLCOMB, ENGLAND, FINCH,
HEAD, BURT, HENDRIX, SOSA, HANEY, FRANKS, SARGENT, NIEVES, DOWNS,
RASMUSSEN, BIRD, HEWITT, LINDSAY, LE, FOREMAN, VALENCE, ONEIL,
DELACRUZ, VINSON, DEJESUS, HYDE, FORBES, GILLIAM, GUTHRIE,
WOOTEN, HUBER, BARLOW, BOYLE, MCMAHON, BUCKNER, ROCHA,
PUCKETT, LANGLEY, KNOWLES, COOKE, VELAZQUEZ, WHITLEY, NOEL,
VANG

FemaleFirst.txt

MARY, PATRICIA, LINDA, BARBARA, ELIZABETH, JENNIFER, MARIA, SUSAN,
MARGARET, DOROTHY, LISA, NANCY, KAREN, BETTY, HELEN, SANDRA,
DONNA, CAROL, RUTH, SHARON, MICHELLE, LAURA, SARAH, KIMBERLY,
DEBORAH, JESSICA, SHIRLEY, CYNTHIA, ANGELA, MELISSA, BRENDA, AMY,
ANNA, REBECCA, VIRGINIA, KATHLEEN, PAMELA, MARTHA, DEBRA,
AMANDA, STEPHANIE, CAROLYN, CHRISTINE, MARIE, JANET, CATHERINE,
FRANCES, ANN, JOYCE, DIANE, ALICE, JULIE, HEATHER, TERESA, DORIS,
GLORIA, EVELYN, JEAN, CHERYL, MILDRED, KATHERINE, JOAN, ASHLEY,
JUDITH, ROSE, JANICE, KELLY, NICOLE, JUDY, CHRISTINA, KATHY,
THERESA, BEVERLY, DENISE, TAMMY, IRENE, JANE, LORI, RACHEL,
MARILYN, ANDREA, KATHRYN, LOUISE, SARA, ANNE, JACQUELINE, WANDA,
BONNIE, JULIA, RUBY, LOIS, TINA, PHYLLIS, NORMA, PAULA, DIANA, ANNIE,
LILLIAN, EMILY, ROBIN, PEGGY, CRYSTAL, GLADYS, RITA, DAWN, CONNIE,
FLORENCE, TRACY, EDNA, TIFFANY, CARMEN, ROSA, CINDY, GRACE,
WENDY, VICTORIA, EDITH, KIM, SHERRY, SYLVIA, JOSEPHINE, THELMA,
SHANNON, SHEILA, ETHEL, ELLEN, ELAINE, MARJORIE, CARRIE,
CHARLOTTE, MONICA, ESTHER, PAULINE, EMMA, JUANITA, ANITA,
RHONDA, HAZEL, AMBER, EVA, DEBBIE, APRIL, LESLIE, CLARA, LUCILLE,
JAMIE, JOANNE, ELEANOR, VALERIE, DANIELLE, MEGAN, ALICIA,
SUZANNE, MICHELE, GAIL, BERTHA, DARLENE, VERONICA, JILL, ERIN,
GERALDINE, LAUREN, CATHY, JOANN, LORRAINE, LYNN, SALLY, REGINA,
ERICA, BEATRICE, DOLORES, BERNICE, AUDREY, YVONNE, ANNETTE, JUNE,
SAMANTHA, MARION, DANA, STACY, ANA, RENEE, IDA, VIVIAN, ROBERTA,
HOLLY, BRITTANY, MELANIE, LORETTA, YOLANDA, JEANETTE, LAURIE,
KATIE, KRISTEN, VANESSA, ALMA, SUE, ELSIE, BETH, JEANNE, VICKI, CARLA,
TARA, ROSEMARY, EILEEN, TERRI, GERTRUDE, LUCY, TONYA, ELLA,
STACEY, WILMA, GINA, KRISTIN, JESSIE, NATALIE, AGNES, VERA, WILLIE, CHARLENE, BESSIE, DELORES, MELINDA, PEARL, ARLENE, MAUREEN, COLLEEN, ALLISON, TAMARA, JOY, GEORGIA, CONSTANCE, LILLIE, CLAUDIA, JACKIE, MARCIA, TANYA, NELLIE, MINNIE, MARLENE, HEIDI, GLENGDA, LYDIA, VIOLA, COURTNEY, MARIAN, STELLA, CAROLINE, DORA, JO, VICKIE, MATTIE, TERRY, MAXINE, IRMA, MABEL, MARSHA, MYRTLE, lena, CHRISTY, DEANNA, PATSY, HILDA, GWENDOLYN, JENNIE, NORA, MARGIE, NINA, CASSANDRA, LEAH, PENNY, KAY, PRISCILLA, NAOMI, CAROLE, BRANDY, OLGA, BILLIE, DIANNE, TRACEY, LEONA, JENNY, FELICIA, SONIA, MIRIAM, VELMA, BECKY, BOBBIE, VIOLET, KRISTINA, TONI, MISTY, MAE, SHELLY, DAISY, RAMONA, SHERRI, ERIKA, KATRINA, CLAIRE

MaleFirst.txt

JAMES, JOHN, ROBERT, MICHAEL, WILLIAM, DAVID, RICHARD, CHARLES, JOSEPH, THOMAS, CHRISTOPHER, DANIEL, PAUL, MARK, DONALD, GEORGE, KENNETH, STEVEN, EDWARD, BRIAN, RONALD, ANTHONY, KEVIN, JASON, MATTHEW, GARY, TIMOTHY, JOSE, LARRY, JEFFREY, FRANK, SCOTT, ERIC, STEPHEN, ANDREW, RAYMOND, GREGORY, JOSHUA, JERRY, DENNIS, RYAN, ROGER, JOE, JUAN, JACK, ALBERT, JONATHAN, JUSTIN, TERRY, GERALD, KEITH, SAMUEL, WILLIE, RALPH, LAWRENCE, NICHOLAS, ROY, BENJAMIN, BRUCE, BRANDON, ADAM, HARRY, FRED, WAYNE, BILLY, STEVE, LOUIS, JEREMY, AARON, RANDY, HOWARD, EUGENE, CARLOS, RUSSELL, BOBBY, VICTOR, MARTIN, ERNEST, PHILLIP, TODD, JESSE, CRAIG, ALAN,
SHAWN, CLARENCE, SEAN, PHILIP, CHRIS, JOHNNY, EARL, JIMMY, ANTONIO,
DANNY, BRYAN, TONY, LUIS, MIKE, STANLEY, LEONARD, NATHAN, DALE,
MANUEL, RODNEY, CURTIS, NORMAN, ALLEN, MARVIN, VINCENT, GLENN,
JEFFERY, TRAVIS, JEFF, CHAD, JACOB, LEE, MELVIN, ALFRED, KYLE,
FRANCIS, BRADLEY, JESUS, HERBERT, FREDERICK, RAY, JOEL, EDWIN, DON,
EDDIE, RICKY, TROY, RANDALL, BARRY, ALEXANDER, BERNARD, MARIO,
LEROY, FRANCISCO, MARCUS, MICHEAL, THEODORE, CLIFFORD, MIGUEL,
OSCAR, JAY, JIM, TOM, CALVIN, ALEX, JON, RONNIE, BILL, LLOYD, TOMMY,
LEON, DEREK, WARREN, DARRELL, JEROME, FLOYD, LEO, ALVIN, TIM,
WESLEY, GORDON, DEAN, GREG, JORGE, DUSTIN, PEDRO, DERRICK, DAN,
LEWIS, ZACHARY, COREY, HERMAN, MAURICE, VERNON, ROBERTO, CLYDE,
GLEN, HECTOR, SHANE, RICARDO, SAM, RICK, LESTER, BRENT, RAMON,
CHARLIE, TYLER, GILBERT, GENE, MARC, REGINALD, RUBEN, BRETT,
ANGEL, NATHANIEL, RAFAEL, LESLIE, EDGAR, MILTON, RAUL, BEN,
CHESTER, CECIL, DUANE, FRANKLIN, ANDRE, ELMER, BRAD, GABRIEL, RON,
MITCHELL, ROLAND, ARNOLD, HARVEY, JARED, ADRIAN, KARL, CORY,
CLAUDE, ERIK, DARRYL, JAMIE, NEIL, JESSIE, CHRISTIAN, JAVIER,
FERNANDO, CLINTON, TED, MATHEW, TYRONE, DARREN, LONNIE, LANCE,
CODY, JULIO, KELLY, KURT, ALLAN, NELSON, GUY, CLAYTON, HUGH, MAX,
DWAYNE, DWIGHT, ARMANDO, FELIX, JIMMIE, EVERETT, JORDAN, IAN,
WALLACE, KEN, BOB, JAIME, CASEY, ALFREDO, ALBERTO, DAVE, IVAN,
JOHNNIE, SIDNEY, BYRON, JULIAN, ISAAC, MORRIS, CLIFTON, WILLARD,
DARYL, ROSS, VIRGIL, ANDY, MARSHALL, SALVADOR, PERRY, KIRK, SERGIO,
MARION, TRACY, SETH, KENT, TERRANCE, RENE, EDUARDO, TERRENCE, ENRIQUE, FREDDIE, WADE