# The ChocAn Simulator

Design Document

Table of Contents

[1Introduction 2](#__RefHeading__457_848818598)

[1.1Purpose and Scope 2](#__RefHeading__459_848818598)

[1.2Target Audience 2](#__RefHeading__463_848818598)

[1.3Terms and Definitions 2](#__RefHeading__465_848818598)

[2Design Considerations 3](#__RefHeading__467_848818598)

[2.1Constraints and dependences 3](#__RefHeading__469_848818598)

[2.2Methodology 3](#__RefHeading__471_848818598)

[3System Architecture 4](#__RefHeading__473_848818598)

[3.1Graph 1 4](#__RefHeading__475_848818598)

[4Detailed System Design 6](#__RefHeading__477_848818598)

[4.1Headers 6](#__RefHeading__479_848818598)

[4.2Files used 6](#__RefHeading__481_848818598)

[4.3Description of file structure(s): 7](#__RefHeading__483_848818598)

[4.3.1Member directory structure: 7](#__RefHeading__485_848818598)

[4.3.2Provider directory structure: 7](#__RefHeading__487_848818598)

[4.3.3Services list structure: 7](#__RefHeading__489_848818598)

[4.3.4Services for the week list structure: 7](#__RefHeading__491_848818598)

[4.3.5Member report structure: 8](#__RefHeading__493_848818598)

[4.3.6Provider report structure: 8](#__RefHeading__495_848818598)

[4.3.7Summary report structure: 9](#__RefHeading__497_848818598)

# Introduction

This document will be in charge of defining how the software requested by ChocAn will be implemented by Brian Breniser on behalf of CS 300 at Portland State University. It will outline purpose, scope, audience, definitions, design and architecture of the software product.

## Purpose and Scope

This document will describe the PSU portion of the ChocAn Software product. It will provided the technical background to create the software product, with the architectural background of the software needed.

## Target Audience

This document appeals to a variety of people:

* Brian Breniser
* CS 300 grader and professor
* Mirroring real world: ChocAn service provider

## Terms and Definitions

Provider: An employee of ChocAn (provides services)

Member: A customer of ChocAn (attends services)

# Design Considerations

This section will describe the constraints, dependences, and methodologies of this software system.

## Constraints and dependences

Constraints:

* Unix, or Unix-Like system (Unix, Linux, osx, or unix terminal emulators)
* Python 3.x installed (anything python 3 or higher is supported, up to 3.4 currently)

Dependences:

* Python 3.x installed
* Python json module installed (comes with 3.4 by default)
* Everything else will be python native

## Methodology

This application will be implemented, by the requirements, via a terminal application. It will take in commands one at a time, and process them accordingly. There will not be a functional date methodology, for implementation purposes we will use called functions to execute “Friday at midnight” kind of scenarios. I'm doing this because in a real application I would have this process running in a separate program calling the functions of this program on it's schedule instead.

I will be using the json module since I can easily import and export python dictionaries to text files without messing with disk I/O. This is to the advantage of the programmer since less code needs to be written, and more will be taken care of by standard libraries which are much more peer reviewed than this will ever be. The disadvantage comes with speed implementation, as using libraries will be slower, and exporting json formats would be slower than accessing files directly. But the trade off is small. It is much easier to implement, and not much slower to use.

# System Architecture

System architecture is an overview of of the structure of the application.

## Graph 1

After the program loads, a command line will be presented, allowing it to read user input. Depending on the commands provided, the application will act in a number of ways.

Adding a service entry: Add a customer service entry, include current date, service date, provider number, member number, service code, comments.

Ask provider directory: display the director of providers

Create reports: create 3 separate files, member report, provider report, and summary report

Customer or provider action: Add a customer/provider, delete a customer/provider, or modify a customer/provider

Arrows in the diagram lead to disk read/writes (using the json format)

# Detailed System Design

Here we will describe, in detail, including pseudo code, the implementation of the software.

## Headers

Import json

This is the only header needed and will import the json library for file use

## Files used

Since we need to keep everything on record in file format, we need two files:

* Provider directory
* Member directory
* Service list

Both are in json format.

The list of services provided will be kept in a list, in this file (after saving):

* Services for the week list

This list will be in the json file format

Reports will be held as files as well those reports are:

* Member report
* Provider report
* Summary report

Each of these reports will be in the json format as well.

## Description of file structure(s):

Pseudo code will be used to describe the format of these files, the hierarchy used will both describe the json format and the python dictionary format (which is what json imports into python as), thus, we can describe both with one description.

Provider directory structure:

### Member directory structure:

* Name (25 char)
* Number (9 digits)
* Street address (25 char)
* City (14 char)
* State (2 char)
* Zip code (5 digits)

### Provider directory structure:

* Name (25 char)
* Number (9 digits)
* Street address (25 char)
* City (14 char)
* State (2 char)
* Zip code (5 digits)

### Services list structure:

* Name (20 char)
* Code (6 digits)
* Fee (up to $999.99)

### Services for the week list structure:

* Current date and time (MM-DD-YYYY HH:MM:SS)
* Date services was provided (MM-DD-YYYY)
* Provider number (9 digits)
* Member number (9 digits)
* Service code (6 digits)
* Comments (100 characters)(optional)

### Member report structure:

* Name (25 char)
* Number (9 digits)
* Street address (25 char)
* City (14 char)
* State (2 char)
* Zip code (5 digits)
* Service list (repeatable):
  + Date of service (MM-DD-YYYY)
  + Provider name (25 char)
  + Service name (20 char)

### Provider report structure:

* Name (25 char)
* Number (9 digits)
* Street address (25 char)
* City (14 char)
* State (2 char)
* Zip code (5 digits)
* Service list (repeatable):
  + Date of service (DD-MM-YYYY)
  + Date and time of data received (MM-DD-YYYY HH:MM:SS)
  + Member name (25 char)
  + Member number (9 digits)
  + Service code (6 digits)
  + Fee to be paid (up to $999.99)
* Total number of consultations with members (3 digits)
* Total fee for the week (up to $99,999.99)

### Summary report structure:

* Provider list (repeatable):
  + Provider name
  + Number of consultations each had
  + Total fee for the week (for that provider)
* Total number of providers who provided services
* Total number of consultations
* Total overall fee