Dylan Barnes and Jacob Visser

CSci 455, Spring 2016, Dr. Hassan Reza, Ph.D.

DC# Manager Project Design

DC# Manager Project Design Table of Contents

[Revision History 3](#_Toc446874227)

[1 Introduction 4](#_Toc446874228)

[1.1 Goals and Objectives 4](#_Toc446874229)

[1.2 Project Overview and Scope 4](#_Toc446874230)

[1.2.1 Core Features 4](#_Toc446874231)

[1.2.2 Additional Features 5](#_Toc446874232)

[1.3 Software Context 5](#_Toc446874233)

[1.4 Major Constraints 5](#_Toc446874234)

[2 Data Design 5](#_Toc446874235)

[2.1 Internal Software Data Structure 5](#_Toc446874236)

[2.2 Global Data Structure 6](#_Toc446874237)

[2.3 Database Description 6](#_Toc446874238)

[2.3.1 Relational Schema Diagram 7](#_Toc446874239)

[3 Architectural and Component-Level Design 7](#_Toc446874240)

[3.1 System Structure 7](#_Toc446874241)

[3.2 Primary Classes 8](#_Toc446874242)

[3.2.1 Form1 8](#_Toc446874243)

[3.2.2 LoginForm 8](#_Toc446874244)

[3.2.3 Program 9](#_Toc446874245)

[3.2.4 DCS\_MAKE 9](#_Toc446874246)

[3.2.5 DCS\_PARSE 9](#_Toc446874247)

[3.2.6 DCS\_REMOVE 9](#_Toc446874248)

[3.2.7 DCS\_SEARCH 9](#_Toc446874249)

[3.2.8 DCS\_STORE 10](#_Toc446874250)

[3.2.9 DamerauLevenshtein 10](#_Toc446874251)

[3.2.10 GlobalConnectionString 11](#_Toc446874252)

[3.3 Auto-Generated Classes 11](#_Toc446874253)

[4 Interface Design 11](#_Toc446874254)

[4.1 Command-Line Interface 11](#_Toc446874255)

[4.2 User Interface 11](#_Toc446874256)

[4.3 Library 11](#_Toc446874257)

[5 Restrictions, Limitations, and Constraints 11](#_Toc446874258)

[6 Testing 12](#_Toc446874259)

[6.1 Types of Testing 12](#_Toc446874260)

[6.1.1 White Box Testing 12](#_Toc446874261)

[6.1.2 Black Box Testing 12](#_Toc446874262)

[6.1.3 Feature Testing 12](#_Toc446874263)

[6.2 Performance Bounds 12](#_Toc446874264)

[6.3 Critical Systems 13](#_Toc446874265)

[6.4 Test Cases 13](#_Toc446874266)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| VERSION | TEAM MEMBER | DESCRIPTION | DATE |
| 0.1.0a | Dylan Barnes | Initial Document Layout and Introduction  Add Sub-Clause 1.1 *Goals and Objectives*  Add Sub-Clause 1.2 *Project Overview and Scope*  Add Sub-Clause 1.3 *Software Context*  Add Sub-Clause 1.4 *Major Constraints*  Add Sub-Clause 2.1 *Internal Software Data Structure*  Add Sub-Clause 2.2 *Global Data Structure*  Add Sub-Clause 2.3 Database Description  Add Figure 2-3 in Sub-Clause 2.3.1  Add Sub-Clause 3.1 *System Structure*  Add Sub-Clause 3.2 *Primary Classes*  Add Sub-Clause 3.3 *Auto-Generated Classes*  Add Sub-Clause 4.1 *Command-Line Interface*  Add Sub-Clause 4.2 *User Interface*  Add Sub-Clause 4.3 *Library*  Add Sub-Clause 6.1 *Types of Testing*  Add Sub-Clause 6.2 *Performance Bounds*  Add Sub-Clause 6.3 *Critical Systems*  Add Sub-Clause 6.4 *Test Cases* | 03/20/2016 |

# Introduction

The purpose of this software document is to provide a low-level overview of DC# Manager, as well as an insight into the structure and design of each component. Since the Agile methodology will be used to develop this software, it is expected that this document will continue to evolve over time. Topics covered include the following:

* Class hierarchies and interactions
* Data flow and design
* Database Description
* Abstract Code and SQL
* Design constraints and restrictions
* Interface design
* Testing procedures

By the time the reader has finished this document, they should have a thorough understanding of the inner workings of DC# Manager.

## Goals and Objectives

DC# Manager is a database management system with numerous uses. It is part of a larger open source software suite, known as Data Control #. Main features involve storing data from various file formats from either local or online sources, sifting through the obtained data using an intuitive search feature, portability due to lack of language dependencies, and the ability to export data from the database into a useful file format.

If the reader is interested in finding out more details about the requirements, they should refer to the *DC# Manager Project Analysis and Specification* document for a more in-depth requirements list.

## Project Overview and Scope

DC# Manager is a database management system. Therefore, the core features are those functions which involve database manipulation, such as updating the database and performing queries. There are also additional features which allow for ease of use and additional function. Below are two list, providing a general overview of these features.

### Core Features

The core features listed below are required to be a part of DC# Manager before the initial version is released.

1. Query content within the database
   * Individual tables/views as well as joined tables
   * Display the results in an appropriate manner
   * Allow additional queries to be performed
   * Ability to reset back to an overall table view
2. Add or Edit information in the database
   * Users with proper permissions should be able to add information to the database
   * Users with proper permissions should be able to edit information in the database
   * Adding/Editing should be easy enough for non-technical users to perform
   * Core functionality involves individual, manual transactions, not bulk
3. Portability
   * In order to stand out amongst similar programs, DC# Manager must be portable
   * This requires various forms of interface access
   * Command-Line, GUI, and API/Library
4. User Access
   * Ability to enter a username/password that is verified by a server

### Additional Features

The following additional features are not guaranteed to be in the application by the completion date, but will be added after core functionality has been accomplished.

1. Connect to a cloud-based database
   * In addition to connecting to a local database, DC# Manager should be able to connect to a cloud-based database as well.
2. Self-Assistance in GUI
   * Users should be able to create a new account through the GUI
   * Users should be able to change their password through the GUI
3. Add information to the database in bulk
   * Users should be able to upload a data file, which in turn is broken down into data and stored in the database
   * DC# Manager should be able to properly label the incoming data
4. Output database data in bulk
   * Users should be able to export queried data in various type
   * This would most likely require a separate interface form

## Software Context

DC# Manager will be released on GitHub as open source software after the completion of the Spring 2016 semester at UND. This will allow the original developers, as well as others that are interested, to contribute to the software after the initial release.

Future development will consist in the completion of the additional feature that could not be completed earlier, as well as development of new features that may prove useful.

## Major Constraints

Currently, the largest constraint on DC# Manager is that of time. The development team consists of two programmers, both attending school full time and work part time. In addition to these, the development team is also learning database systems, the C# language, and Visual Studios as they go along. Due to this learning curve and lack of free time, project programming times are quite constrained.

# Data Design

## Internal Software Data Structure

The internal data structure of DC# Manager can be broken down into two subcategories: front-end and back-end.

The front-end work involves everything the user will interact with. This primarily involves the interfaces, which varies between a GUI, command-line interface, and library. The GUI must be designed simplistically, as it will be the primary source of interaction for the standard non-technical user. The GUI will also allow for additional functionality that may not be feasible in a command-line interface. However, the command-line interface will still attempt to mirror the functionality of the GUI, as this will be the method that other applications use to interact with DC# Manager. Together, these interfaces serve as a way to access the core and additional features of DC# Manager.

The back-end work involves the functionality and interactions with the database. The functionality is relating to the features listed in sub-clause 1.2. In addition to this, extensive work will be required to manipulate content in a cloud based database, as per our first additional feature. This additional work will come in the form of SQL, as SQL is what the developers are familiar with, and there is extensive documentation on this robust language. However, due to the nature of the C# language, there are moments when TSQL will be used to interact with the database as well.

## Global Data Structure

The global data structure of DC# Manager is best characterized by the database that is connected to DC# Manager. The default database used by DC# Manager is a cloud-based Azure database. This database contains all data that will be used by DC# Manager. This data is accessed and manipulated via SQL queries on the back-end and interface features on the front-end. The user never has direct, unprotected access to the database contents.

## Database Description

Due to the dynamic nature of DC# Manager, there is an infinite number of possibilities for database structure. There are also two ways for users to modify the database, either by manually editing/adding a record, or bulk upload a data file. The latter is the only type of data modification functionality that will create new tables.

For the sake of testing purposes, we have populated our default Azure database with content procured from *Fundamentals of Database Systems (7th Edition)* by authors Ramez Elmasri and Shamkant B. Navathe. Slight modifications have been made to the relational schema presented in the textbook, and the recreated relational schema diagram is displayed below.

### Relational Schema Diagram

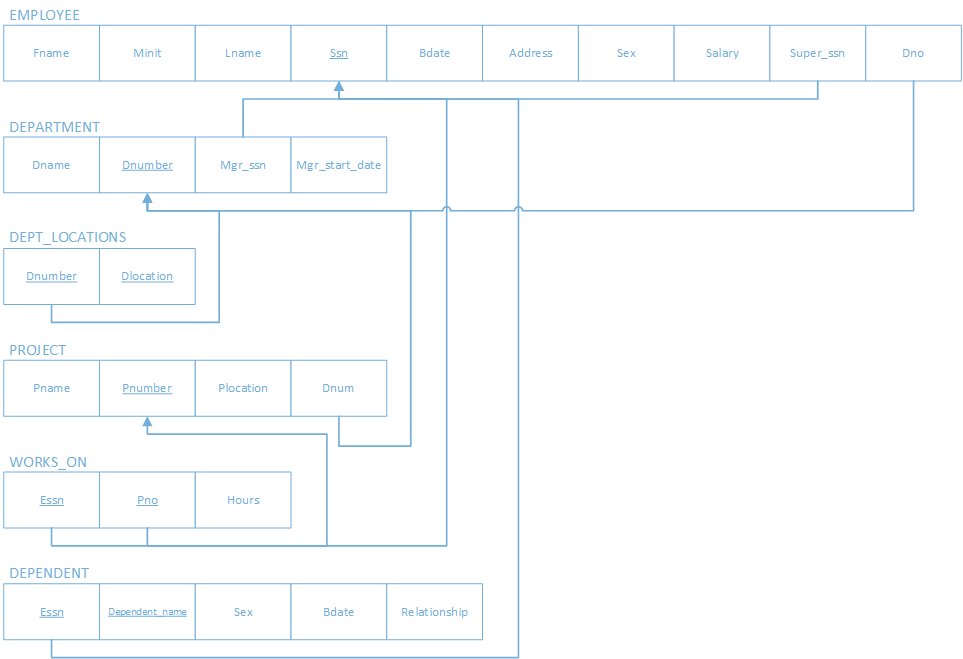


Figure 2-3: Relational Schema Diagram of the COMPANY database schema presented in the textbook “Fundamentals of Database Systems (7th Edition)”. This database schema is used throughout the development of DC# Manager for testing purposes.

# Architectural and Component-Level Design

## System Structure

The majority of the front-end work in DC# Manager exists solely in the *Form1* class, along with auto-generated classes that manage the design of the GUI. *Form1* is used to react to user events, which in turn calls functions that perform actions on the back-end.

The back-end work is split into multiple classes. However, the primary functionality exists in the following classes: *DCS\_MAKE, DCS\_PARSE, DCS\_REMOVE, DCS\_SEARCH,* and *DCS\_STORE*. Additional classes supplement these core classes. All classes are described in further detail below.

## Primary Classes

### Form1

Form1 contains all code relating to the main GUI form. Functions include those that respond to interface events, the backend portions of events, and manipulating the locally stored data pulled from the database.

#### Global Variables

* Columns : List<String>
* currentTable : String
* mySearch : DCS\_SEARCH
* conn : SqlConnection
* Ds : DataSet
* Da : SqlDataAdapter

#### Class Signatures and SQL

* **Form1(string username)**
  + "SELECT count(\*) FROM " + currentTable
* **Button1\_Click(object sender, EventArgs e) : void**
* **Button2\_Click(object sender, EventArgs e) : void**
* **Button3\_Click(object sender, EventArgs e) : void**
* **Clear(int i) : void**
* **clearProgressBar() : void**
* **comboBox1\_SelectionChangeCommitted(object selectionChangeCommited)**
* **fillTable(SqlDataAdapater dataAdapter) : void**
* **findList() : List<String>**
* **Form1\_Load(object sender, EventArgs e) : void**
* **getColumns() : void**
  + "SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME='" + table.Trim() + "'"
* **incrementProgressBar() : void**
* **initFillTable() : void**
  + "SELECT \* FROM" + currentTable
  + "SELECT count(\*) FROM" + currentTable
* **linkLabel1\_LinkClicked (object sender, LinkedLabelClickedEventArgs e) : void**
* **maxProgressBar() : void**
* **setTableSelector() : void**
  + "SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE='BASE TABLE' AND TABLE\_NAME<>'sysdiagrams'"

### LoginForm

LoginForm includes all functions related to interface features on the login screen.

#### Global Variables

* Username : String

#### Class Signatures and SQL

* **Button1\_Click(object sender, EventArgs e) : void**
* **LoginForm\_Load(object sender, EventArgs e) : void**

### Program

The Program class contains main, which runs all of the forms. In addition to this, it allows for command-line interaction and controls the order in which forms are executed.

#### Global Variables

* None

#### Class Signatures and SQL

* **Main() : void**

### DCS\_MAKE

DCS\_MAKE performs all actions involving outputting database data to an output file.

#### Global Variables

* elementList : List<String>
* fileType : String

#### Class Signatures and SQL

* **Create() : void**
* **getElementList() : List<String>**
* **getFileType() : String**
* **setElementList(List <String> input) : void**
* **setFileType(String input) : void**

### DCS\_PARSE

DCS\_PARSE performs all actions related to parsing data. It has been depreciated due to the development of the DCS\_CONVERTER library.

### DCS\_REMOVE

DCS\_REMOVE performs all actions related to removing information from the database.

#### Global Variables

* connectionString : String
* contentList : List<String>

#### Class Signatures and SQL

* **getConnectionString() : String**
* **getContentList() : List<String>**
* **setContentList(List<String> input) : void**
* **strip() : void**

### DCS\_SEARCH

DCS\_SEARCH performs all operations related to search functionality.

#### Global Variables

* connectionString : String
* form : Form1
* query : String
* results : List<String>

#### Class Signatures and SQL

* **didYouMean(String tempSQL, String tempField) : String**
* **findList() : List<String>**
  + "SELECT \* FROM " + form.currentTable + " WHERE CAST({0} AS char(20))=\'{1}\'", form.columns[0], form.searchBox1.Text
* **getConnectionString() : String**
* **getQuery() : String**
* **searchButton() : void**
  + "SELECT \* FROM " + form.currentTable
* **setQuery(String input) : void**
* **sortByDistance(List<String> unsortedList, string input) : List<String>**
* **suggestionField() : String**

### DCS\_STORE

DCS\_STORE performs all actions related to storing data in the database.

#### Global Variables

* connectionString : String
* conectent : List<String>

#### Class Signatures and SQL

* **Dump() : void**
* **getConnectionString() : String**
* **getContentList() : List<String>**
* **getContentList(List <String input) : void**

### DamerauLevenshtein

The DemaraueLevenshtein class includes an algorithm used in the DidYouKnow function in DCS\_SEARCH in order to suggest relative replacement options for a user search query.

#### Global Variables

* None

#### Class Signatures and SQL

* **DamerauLevenshteinDistanceTo(this string @string, string targetString) : int**
* **DamerauLevenshteinDistance(string string1, string string2) : int**

### GlobalConnectionString

GlobalConnectionString is currently used for security purposes in order to prevent hard-coded connection strings within the code. It is used within the Login Form.

#### Global Variables

* GlobalConnectionString : String

#### Class Signatures and SQL

* None

## Auto-Generated Classes

In addition to the manually created classes listed above, there are numerous classes auto-generated by Visual Studio in order to properly format the interface as well as properly compile the application. For the sake of brevity, these will not be explained in further details within this document.

# Interface Design

## Command-Line Interface

The command-line interface allows technical users to quickly and easily perform the core functionalities of DC# Manager. This interface is also useful to other applications, at it allows programs written in other languages to use DC# Manager functionality by supplying arguments and receiving output.

This interface is key to the development of DC# Manager, as it allows the application to be language independent. This is quite crucial, as it will expand the possible uses of DC# Manager.

## User Interface

The user interface will allow non-technical users and those wishing to use more advanced features to directly interact with DC# Manager in an intuitive way. The interface will support all currently existing core and additional features. For further details on the specifications of the GUI, please refer to the *DC# Manager Project Analysis and Specification* document.

## Library

DC# Manager will be packaged as a DLL file when the package is distributed as open source content. This will allow C# programmers to access any of the functions in DC# Manager, which in turn could be used to enhance the programmer’s own application.

# Restrictions, Limitations, and Constraints

As noted previously, the largest constraint on our project is time. We are limited to one semester to complete the initial version of the software, which must contain all of the core features, and as many additional features as possible.

However, time is not the only constraint of this project. There are also technical constraints as well. Our primary blocker involves working with tables dynamically. In particular, adding a table, and ensuring that table is fully editable upon creation, is quite a difficult task to complete at run time.

In addition to this, there have also been issues with creating new users. While this could be done by hard coding a connection string or user email, this poses a security risk. Therefore, more discussion must be done to determine what could be done to create a new user and supply the appropriate permissions.

Due to the use of the Agile methodology, we are witnessing and overcoming obstacles as we come across them. If these obstacles are significant, they will be added to this document as a restriction, limitation, or constraint.

# Testing

Each function and class will be tested individually to ensure they operate properly. Once functionality is added to the core application, the application as a whole will have regression testing performed.

## Types of Testing

### White Box Testing

When a class is being implemented, the developer of that class will test to ensure each functional component is working properly. The developer of the functionality is fully responsible for debugging their own code, as debugging another developer’s code can be time consuming.

### Black Box Testing

Black Box testing involves a majority of the testing. This testing is done once all functionality exists and the components have been assembled. Tests will be performed throughout the entirety of the application, ensure every possible situation that could be applied to the application succeeds without error.

### Feature Testing

The features will be tested through the use of broad test cases. The reader may find these test cases in sub-clause 6.4, *Test Cases*.

## Performance Bounds

The local front-end performance is negligible, as the majority of the interface is auto-generated on the back-end, leaving little need for optimization. In addition to this, if DC# Manager is connected to a local database, the performance will be negligible as well. Performance will be obtained by reducing the amount of database transactions required to perform an action.

The majority of performance issue will come through interaction with an online database. While many aspects of this performance is reliant on the user’s internet speed, there is still room for optimization. As with local databases, the primary form of performance improvement will come through reducing the transactions required.

The primary Azure database is currently at the lowest performance tier available, in order to cut monetary costs. However, if the software became popular once being open sourced, the possibility to expand the Azure database subscription may be discussed. However, the software is designed to work with any Microsoft database. Therefore, a technical user could change the database interacting with DC# Manager in order to improve performance and enhance customization.

## Critical Systems

The two most critical systems of DC# Manager are correct data parsing/output and the connected database must be updated correctly and contain no corrupted or false data.

The data parsing and output integrity will fall on the success of DC# Converter. As long as DC# Manager feeds the appropriate data to the DC# Converter classes, parsing and output should be performed properly.

Database integrity will be managed by the connected database and the update functionality of DC# Manager. Error testing will be put in place on the client-side, ensuring updates are performed and completed successfully. The Microsoft database itself will ensure that data is not corrupted or changed incorrectly. Therefore, between client and server side operations, data integrity shall be maintained.

## Test Cases

Table 6-4 lists all currently planned test cases.

|  |  |
| --- | --- |
| FEATURE | CASES |
| *Sign In* | Connection to database is established, existing user is verified. |
|  | Connection to database is established, non-existing user is verified to not exist. |
|  | Connection to database is established, incorrect information is entered, proper error message is displayed. |
| *Account Creation* | User with admin privileges shall be able to create a new user in the database. |
|  | A user shall not be created with the same username as an existing user. An error message shall be displayed. |
|  | User without admin privileges shall not be able to view the create account functionality. |
| *Forgot Password* | Existing user is able to obtain assistance in obtaining password. |
|  | Non-existent user shall not be able to obtain a password. |
| *Command-Line* | Users shall be able to access core functionality through the command line. |
| *Database Connection* | Connection to the database shall succeed if the database exists. |
|  | Connection to the database shall fail if the database does not exist. |
|  | Technical users shall be able to connect to a different database and still use all of the same functionality. |
| *Data Grid View* | A valid database shall populate the data grid with a default table. |
|  | An invalid database shall display an error to the user. |
|  | All tables in a valid database shall be able to merge. |
| *Manipulate Data with Permission* | Users with correct permissions shall be able to change the database data. (Update, Delete, Insert) |
|  | Users with incorrect permissions shall not be able to change the database data. |
| *Output Data Selection* | Users shall be able to output database data to an appropriate output format. |
|  | Users shall not be able to output invalid data. |
|  | Users shall not be able to output to an unsupported output format. |
| *Performance* | Time all actions to ensure the command-line application runs smoothly. |
|  | Time all actions to ensure the GUI application runs smoothly. |

Table 6-4: Test Cases