Dylan Barnes, Ryan Kilbride, and Cameron Kerbaugh

CSci 463, Spring 2016, Dr. Hassan Reza, Ph.D.

DC# Converter Project Design

DC# Converter Project Design Table of Contents

[Revision History 3](#_Toc448239564)

[1 Introduction 4](#_Toc448239565)

[1.1 Goals and Objectives 4](#_Toc448239566)

[1.2 Project Overview and Scope 4](#_Toc448239567)

[1.2.1 Core Features 4](#_Toc448239568)

[1.2.2 Additional Features 4](#_Toc448239569)

[1.3 Software Context 4](#_Toc448239570)

[1.4 Major Constraints 4](#_Toc448239571)

[2 Data Design 4](#_Toc448239572)

[2.1 Internal Software Data Structure 4](#_Toc448239573)

[2.2 Global Data Structure 4](#_Toc448239574)

[3 Architectural and Component-Level Design 4](#_Toc448239575)

[3.1 System Structure 4](#_Toc448239576)

[3.2 Primary Classes 4](#_Toc448239577)

[3.2.1 DCS\_CSV 5](#_Toc448239578)

[3.2.2 DCS\_JSON 5](#_Toc448239579)

[3.2.3 DCS\_XML 5](#_Toc448239580)

[3.2.4 Form1 5](#_Toc448239581)

[3.3 Auto-Generated Classes 5](#_Toc448239582)

[4 Interface Design 5](#_Toc448239583)

[4.1 Command-Line Interface 5](#_Toc448239584)

[4.2 Graphical User Interface (GUI) 5](#_Toc448239585)

[4.3 Library 5](#_Toc448239586)

[5 Restrictions, Limitations, and Constraints 5](#_Toc448239587)

[5.1 Limitations to Parsing 5](#_Toc448239588)

[5.1.1 Reliability 5](#_Toc448239589)

[5.1.2 Robustness 5](#_Toc448239590)

[6 Testing 6](#_Toc448239591)

[6.1 Types of Testing 6](#_Toc448239592)

[6.1.1 White Box Testing 6](#_Toc448239593)

[6.1.2 Black Box Testing 6](#_Toc448239594)

[6.1.3 Feature Testing 6](#_Toc448239595)

[6.2 Performance Bounds 6](#_Toc448239596)

[6.3 Critical Systems 6](#_Toc448239597)

[6.4 Test Cases 6](#_Toc448239598)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| VERSION | TEAM MEMBER | DESCRIPTION | DATE |
| 0.1.0a | Dylan Barnes | Initial Document Layout and Introduction | 4/04/2016 |
|  |  |  |  |
| 0.1.1a | Ryan Kilbride | Fault tolerance, under Restrictions, Limitations, and Constraints | 4/12/2016 |
|  |  |  |  |
| 0.1.1b | Cameron Kerbaugh | Added content to Introduction | 4/12/2016 |
| 0.1.1c | Cameron Kerbaugh | Interface Design section | 4/13/2016 |

# Introduction

This Project Design Document is intended to provide an overview and description of DC# Converter’s low-level design, examining how each individual component is structured as well as how the overall software package is connected. Design information contained in this document includes:

* Software Structure
* Class Hierarchies
* Software Architecture Chosen
* User Interface Design Details
* Test Cases

## Goals and Objectives

DC# Converter aims to provide a portable, easy-to-use, multi-function conversion tool for converting between several popular data exchange formats, including XML, JSON, and CSV. It is a component of the larger DC# Software Suite, an open source software suite for managing and converting data files of various formats.

In order to provide the aforementioned portability and ease of use, both a GUI and command line functionality must be implemented. The GUI will also provide feedback to the user about the data contained in converted files, allowing them to more closely inspect the data files they are using. Additional details about the system requirements are available in the *DC# Converter Project Definition and Specification* document.

## Project Overview and Scope

DC# Converter is a file conversion system, and as such, the core features are those that allow and support reading, converting, and writing different file types while retaining the data within those files. Portability is also important to the system, as DC# Converter aims to provide a single-program solution that works for users of various programming languages, rather than being limited to use in programs made in C#. Additional functionality for ease of use and additional data manipulation and monitoring is also included.

### Core Features

The core features listed below are to be implemented before the initial release of the DC# Converter application.

1. Select input file for conversion
   * CSV, XML, and JSON file types are supported
   * Through file browser in GUI or full file path provided in Command-Line/Library calls
2. Select desired output type
   * CSV, XML, and JSON file types are supported
   * Drop-down menu in GUI, imbedded in output file name in Command-Line/Library calls
3. Portability
   * Accessible via GUI, Command-Line, and Library
   * Usable in other (non-C#) through system calls
4. Data View
   * Provides users with an overview of data contained in converted files
   * Primarily accessed via GUI
5. Conversion/Parsing options
   * Allow users to specify special conversion options
   * Available via GUI menus and additional arguments in Command-Line/Library calls

### Additional Features

The following additional features will be added as time allows, but may not be implemented prior to the initial application release.

1. Select remote files for conversion
   * Via a URL non-local files of the supported types can be selected for conversion
2. Interface with other members of the DC# Software Suite
   * DC# Converter will be adapted and optimized for use with the rest of the software suite

## Software Context

DC# Converter, as a component of the larger DC# Software Suite, will be released as open source software through GitHub after the culmination of the Spring 2016 semester. Through this release, the original developers, along with any additional interested parties will be able to continue development of the DC# Software Suite, adding additional functionality and expanding the system as needed.

## Major Constraints

The primary constraint on DC# Converter’s development is the lack of available time on the part of the developers. All three developers are full-time students, with additional projects and commitments. All core functionality will be fully implemented, but some additional features may not be completed by the initial release time.

# Data Design

## Internal Software Data Structure

## Global Data Structure

# Architectural and Component-Level Design

## System Structure

## Primary Classes

The DCS Converter is broken into three data-oriented modules and one TODO. The data-oriented classes, DCS\_CSV, DCS\_JSON, and DCS\_XML, TODO

### DCS\_CSV

#### Global Variables

#### Class Signatures

### DCS\_JSON

### DCS\_XML

### Form1

#### Global Variables

#### Class Signatures and SQL

## 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

DC# Converter has a command-line interface to allow technically adept users to access primary conversion functionality quickly and easily through the command line. Additionally, this interface also allows external applications, particularly those written in other languages, to access DC# Converter’s functionality through system calls.

This interface is what allows DC# Converter to operate as a language-independent application, which makes it more broadly useful and accessible than if it was limited to only being usable with other applications written in C#.

## Graphical User Interface (GUI)

The graphical user interface allows the average user to access DC# Converter’s conversion functionality, and also allows all users to utilize the more advanced features for modifying output and viewing the contents of parsed files.

## Library

DC# Converter will be included as a DLL in the open source distribution of the DC# Software Suite. This allows C# programmers to easily access the functions implemented in DC# converter for use in their applications. This is integral to the modular and reusable design desired for the entire DC# Software Suite.

# Restrictions, Limitations, and Constraints

## Limitations to Parsing

### Reliability

### Robustness

In the event that a file cannot successfully be parsed, the system gracefully handles internal errors. Any exceptions thrown internally are caught, and the user is notified via a message box.

# Testing

## 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

## Critical Systems

## Test Cases

Table 6-4 lists all currently planned test cases.

|  |  |
| --- | --- |
| FEATURE | CASES |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 6-4: Test Cases