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](#_Toc448239614)

[1 Introduction 4](#_Toc448239615)

[1.1 Goals and Objectives 4](#_Toc448239616)

[1.2 Project Overview and Scope 4](#_Toc448239617)

[1.2.1 Core Features 4](#_Toc448239618)

[1.2.2 Additional Features 4](#_Toc448239619)

[1.3 Software Context 4](#_Toc448239620)

[1.4 Major Constraints 4](#_Toc448239621)

[2 Data Design 4](#_Toc448239622)

[2.1 Internal Software Data Structure 4](#_Toc448239623)

[2.2 Global Data Structure 4](#_Toc448239624)

[3 Architectural and Component-Level Design 4](#_Toc448239625)

[3.1 System Structure 4](#_Toc448239626)

[3.2 Primary Classes 4](#_Toc448239627)

[3.2.1 DCS\_CSV 4](#_Toc448239628)

[3.2.2 DCS\_JSON 5](#_Toc448239629)

[3.2.3 DCS\_XML 5](#_Toc448239630)

[3.2.4 DCS\_ALL 5](#_Toc448239631)

[3.2.5 Form1 5](#_Toc448239632)

[3.3 Auto-Generated Classes 5](#_Toc448239633)

[4 Interface Design 5](#_Toc448239634)

[4.1 Command-Line Interface 5](#_Toc448239635)

[4.2 User Interface 5](#_Toc448239636)

[4.3 Library 5](#_Toc448239637)

[5 Restrictions, Limitations, and Constraints 5](#_Toc448239638)

[5.1 Limitations to Parsing 5](#_Toc448239639)

[5.1.1 Reliability 5](#_Toc448239640)

[5.1.2 Robustness 5](#_Toc448239641)

[6 Testing 6](#_Toc448239642)

[6.1 Types of Testing 6](#_Toc448239643)

[6.1.1 White Box Testing 6](#_Toc448239644)

[6.1.2 Black Box Testing 6](#_Toc448239645)

[6.1.3 Feature Testing 6](#_Toc448239646)

[6.2 Performance Bounds 6](#_Toc448239647)

[6.3 Critical Systems 6](#_Toc448239648)

[6.4 Test Cases 6](#_Toc448239649)

# 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 |

TODO GLOSSARY?

# Introduction

This Project Design Document is intended to provide an overview and description of DC# Converter’s low-level design, looking into how each individual component is structured as well as how the overall software package is connected. Design information contained in this document includes: data structure used, software architecture chosen, user interface design details, and test cases along with their intended results.

## 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,

## Project Overview and Scope

### Core Features

### Additional Features

## Software Context

## Major Constraints

# Data Design

## Internal Software Data Structure

## Global Data Structure

# Architectural and Component-Level Design

## System Structure

TODO finish up this portion, talk about PIPE and FILTER?? Back end work

The GUI combines an event based system with the main system’s functionality. *Form1* and the auto generated classes are used to display GUI. *Form1* reacts to user events and calls functions on the back end.



Figure : Data conversion process overview. TODO

## Class Documentation

The DCS Converter is broken into a variety of classes. The data-oriented classes, DCS\_CSV, DCS\_JSON, and DCS\_XML, are responsible for handling data related to CSV files, JSON files, and XML files, respectively. DCS\_ALL contains higher level parsing and output functions that are common to the three data-oriented classes. The Form1 class is responsible for the user interface and event handling. Each of these classes will be elaborated on below. For more information on individual classes and methods, please refer to the attached Class Documentation Document NAME document for details.

### DCS\_ALL

DCS\_ALL is a helper class for events called from the GUI. There are three methods, which are responsible for:

* Determining file type of the input file to be parsed and calling the appropriate functions from DCS\_CSV, DCS\_JSON, and DCS\_XML.
* Calling the correct output method, much like the first. It will automatically determine the file type and call the corresponding methods.
* Creating a data table from the parsed data and returning it for the user interface to display.

These methods are named parseFile(…), outputFile(…), and objToDataTable(…), respectively. Please refer to the attached Class Documentation Document NAME document for details.

### DCS\_CSV

This class performs parsing and output operations that are specific to CSV files. Please refer to the attached Class Documentation Document NAME document for details.

### DCS\_JSON

This class performs parsing and output operations that are specific to JSON files. Please refer to the attached Class Documentation Document NAME document for details.

### DCS\_XML

This class performs parsing and output operations that are specific to XML files. Please refer to the attached Class Documentation Document NAME document for details.

### Form1

Form1 contains all code relating to the main GUI form. Functions within are responsible for handling GUI events, such as button presses, file dialog boxes, and showing tables of parsed content.

## Auto-Generated Classes

In addition to the manually created classes listed in Section 3.2 of this document, there are numerous classes auto-generated by Visual Studio in order to properly format the interface as well as compile the application. For the sake of brevity, these will not be explained in further details within this document.

# Interface Design

## Command-Line Interface

## User Interface

## Library

# Restrictions, Limitations, and Constraints

## Limitations of Methodology

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.

## Limitations to Parsing

Due to the time and experience limitations we have faced, our product is not perfect. There are still some conversion issues that we face. These issues are mainly due to the wide amount of variance in XML, JSON, and CSV files. Though we cannot account for all situations we have made efforts towards providing reliable data conversion as well as the appropriate feedback if data is not correctly parsed.

### Reliability

Further testing is needed to determine whether the system performs reliably. See Section 6: Testing for details on testing types, critical systems and future test cases. Tests will be completed with the release of DC# Converter version 1.0.

### Robustness

In the event that files cannot successfully be parsed, the system gracefully handles internal errors. The system makes use of active fault detection. Exception handling is used when opening files, parsing data, converting data between types, and writing to file. The system recovers from faults by discarding parsed data and generating a report, which is written to a log file (TODO). The user is notified of the error via a message box.

TODO validation of files section?

TODO HUMAN ERRORS ID10T

# 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