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](#_Toc449037820)

[1 Introduction 4](#_Toc449037821)

[1.1 Goals and Objectives 4](#_Toc449037822)

[1.2 Project Overview and Scope 4](#_Toc449037823)

[1.2.1 Core Features 4](#_Toc449037824)

[1.2.2 Additional Features 5](#_Toc449037825)

[1.3 Software Context 5](#_Toc449037826)

[1.4 Major Constraints 5](#_Toc449037827)

[2 Data Design 5](#_Toc449037828)

[2.1 Internal Software Data Structure 5](#_Toc449037829)

[2.2 Global Data Structure 6](#_Toc449037830)

[2.3 File Creation 6](#_Toc449037831)

[3 Architectural and Component-Level Design 6](#_Toc449037832)

[3.1 System Structure 6](#_Toc449037833)

[3.2 Class Documentation 6](#_Toc449037834)

[3.2.1 DCS\_ALL 7](#_Toc449037835)

[3.2.2 DCS\_CSV 7](#_Toc449037836)

[3.2.3 DCS\_JSON 7](#_Toc449037837)

[3.2.4 DCS\_XML 7](#_Toc449037838)

[3.2.5 Form1 7](#_Toc449037839)

[3.3 Auto-Generated Classes 7](#_Toc449037840)

[4 Interface Design 7](#_Toc449037841)

[4.1 Command-Line Interface (CLI) 7](#_Toc449037842)

[4.2 Graphical User Interface (GUI) 8](#_Toc449037843)

[4.3 Library 8](#_Toc449037844)

[5 Restrictions, Limitations, and Constraints 8](#_Toc449037845)

[5.1 Limitations of Methodology 8](#_Toc449037846)

[5.2 Limitations to Parsing 8](#_Toc449037847)

[5.2.1 Reliability 8](#_Toc449037848)

[5.2.2 Robustness 8](#_Toc449037849)

[6 Testing 9](#_Toc449037850)

[6.1 Types of Testing 9](#_Toc449037851)

[6.1.1 White Box Testing 9](#_Toc449037852)

[6.1.2 Black Box Testing 9](#_Toc449037853)

[6.1.3 Feature Testing 9](#_Toc449037854)

[6.2 Performance Bounds 9](#_Toc449037855)

[6.3 Critical Systems 9](#_Toc449037856)

[6.4 Test Cases 9](#_Toc449037857)

[7 Glossary 10](#_Toc449037858)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| VERSION | TEAM MEMBER | DESCRIPTION | DATE |
| 0.1.0a | Dylan Barnes | Initial Document Layout and Introduction | 4/04/2016 |
|  |  |  |  |
| 0.1.1 | Ryan Kilbride | Fault tolerance, under Restrictions, Limitations, and Constraints | 4/12/2016 |
|  |  |  |  |
| 0.1.1b | Cameron Kerbaugh | Added content to Introduction | 4/12/2016 |
|  |  |  |  |
| 0.1.2 | Dylan Barnes | Added Sub-Clause 2.1  Added Sub-Clause 2.2  Added Sub-Clause 2.3 | 4/18/2016 |
|  |  |  |  |
| 0.1.2b | Dylan Barnes | Added Sub-Clause 6.1  Added Sub-Clause 6.2  Added Sub-Clause 6.3  Added Sub-Clause 6.4  Added Table 6-4 | 4/20/2016 |
|  |  |  |  |
| 0.1.3 | Cameron Kerbaugh | Updated Sub-Clause 1.1  Added Sub-Clause 1.2  Added Sub-Clause 1.3  Added Sub-Clause 1.4  Added Sub-Clause 4.1  Added Sub-Clause 4.2  Added Sub-Clause 4.3 | 4/20/2016 |
|  |  |  |  |
| 0.1.3b | Cameron Kerbaugh | Added Main Clause 7  Updated table formatting  Edited grammar | 4/21/2016 |
| 0.1.4 | Ryan Kilbride | Merged working branch with this document  Added Clause 3 and Sub-Clauses  Updated Clause 5  Added GUI to Glossary of Terms  Added CLI to Glossary of Terms  Refined first usage of Glossary terms in Sub-Clause 1.1 | 4/21/2016 |

# 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 eXtensible Markup Language (XML), JavaScript Object Notation (JSON), and Comma-Separated Values (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 graphical user interface (GUI) and command line interface (CLI) 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

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# Converter. Together, these interfaces serve as a way to access the core and additional features of DC# Converter.

The back-end work involves the functionality, such as parsing and converting data. The functionality is relating to the features listed in sub-clause 1.2. In addition to this, research will have to be performed to determine if it is viable to use pre-existing C# libraries to assist in the conversion of data. If external libraries are required, it will be our goal to ensure the number of libraries required is kept to a minimum to decrease the number of dependencies.

## Global Data Structure

DC# Converter does not require a global data structure. The GUI and command-line applications are initialized on a per-instance basis, while the library is dependent upon how users implement the classes. DC# Converter may be viewed as a pipe and filter system, where files are fed into the functions of the converter, and a new file comes out the other end. No data is stored within the application itself.

## File Creation

It is worth noting that every successful conversion results in the creation of a new file. If the user is interacting with DC# Converter through the GUI, a file explorer will be displayed allowing the user to easily specify a directory and file name. On the other hand, if a command-line interface is used, the user will be expected to provide the full output file path before performing a conversion. The size of the output file is dependent on the size of the input file, as well as the output file type. This variance in size is due to the syntax of the file type. For example, converting a CSV file to XML will result in a larger file than if you were to convert that same CSV file to JSON. As one would expect, the larger the input file is, the longer conversion may take.

# Architectural and Component-Level Design

## System Structure

DC# Converter acts as a pipe and filter system, in which files are parsed for data and the resultant dynamic object is converted to an output type. This functionality is common to the GUI, CLI, and the DLL library. The figure below details this conversion process. In addition to this functionality, the GUI combines an event based system with the main system’s functionality. *Form1* and the auto generated classes are used to display the GUI. *Form1* reacts to user events and calls functions on the back end.



Figure : Data conversion process overview.

## 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 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 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 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 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 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 (CLI)

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 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. The user is notified of the error via a message box.

It is possible that a user may specify a file for conversion that is not valid JSON, XML, or CSV syntactically. DC# Converter does not validate these files before attempting is parsed. However in the event this does occur, the exception(s) generated shall be handled gracefully by the Converter.

# 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

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. Performance will be obtained by increasing conversion times on the back-end, which may be done by manually modifying existing code or incorporating a new open-source parsing library.

Performance is being kept in mind throughout the development of DC# Converter. Improved performance is typically achieved through minimal use of looping, as well as minimal use of print statements within said loops. Current parsing times are sufficient for that of a conversion software, but further optimization may be investigated following the initial release.

## Critical Systems

The most critical functionality of DC# Converter is accurate data parsing. If a data file is not parsed correctly, the output will be incorrect as well. Ensuring the integrity of the original data is critical for user acceptance.

Throughout the development of DC# Converter, the accuracy of the parsing and corresponding output is consistently tested and optimized. The accuracy of the parsing is done manually as opposed to programmatically. The accuracy is typically done by comparing the input file to the output file, and ensuring that not only the syntax is correct but that no data was lost in the conversion process.

There are future plans to create programmatic unit tests. However, due to the accelerated schedule of the initial release, manual testing has been preferred, as programming unit tests would require an extensive amount of additional work time.

## Test Cases

Table 6-4 lists all currently planned test cases. Test cases are assumed to be performed on the graphic user interface, unless otherwise specified.

|  |  |
| --- | --- |
| FEATURE | CASES |
| *Input/Output File Selection* | User shall be able to select an input file through a file explorer. |
|  | User shall be able to select an output file through a file explorer. |
|  | Input and output file types shall be restricted to the file types available for input and output. |
| *Output File Type* | User shall be able to select an output file type. |
|  | The output file type restricts the output file type selection in the file explorer. |
|  | The user shall not have to specify an output file type if the command line is used, the output file type will be interpreted. |
| *Output File Options* | Additional options shall be provided to the user for the output file. |
|  | The provided options shall be dependent upon the selected output file type. Thus, the options differ between file types. |
| *Data Grid View* | The parsed data shall be displayed after conversion is performed. |
|  | The data will be displayed in a data grid, allowing users to easily view the content that will be output to a new file. |
| *Command-Line* | Users shall be able to access core functionality through the command line. |
| *C# Library* | Users shall be able to utilize all functionality through the use of a .dll library. |
| *Data Integrity* | The data integrity shall be verified throughout development. |
|  | Data integrity shall be validated manually. |
| *Syntax Validation* | The syntax out the output files shall be verified throughout development. |
|  | Syntax shall be validated manually. |
| *Performance* | Time all actions to ensure command-line conversion runs at a consistent, acceptable speed. |
|  | Time all actions to ensure the GUI conversion process runs at a consistent, acceptable speed. |

Table 6-4: Test Cases

# Glossary

The glossary in Table 7 shall contain definitions for terms and abbreviations pertinent to this project and its goals.

|  |  |
| --- | --- |
| TERM | DEFINITION |
| DLL | Dynamic Link Library |
| CLI | Command-Line Interface |
| CSV | Comma-Separated Values |
| GUI | Graphical User Interface |
| JSON | JavaScript Object Notation |
| UND | University of North Dakota |
| XML | eXtensible Markup Language |

Table 7: Glossary