Dylan Barnes, Ryan Kilbride, Cameron Kerbaugh

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

DC# Converter PROJECT DEFINITION AND SPECIFICATION

DC# Converter Project Definition and Specification Table of Contents

[Revision History 2](#_Toc447660752)

[1 Introduction 3](#_Toc447660753)

[2 Problem Analysis and Solutions 3](#_Toc447660754)

[2.1 Data Conversion 3](#_Toc447660755)

[2.2 Data File Contents 3](#_Toc447660756)

[2.3 Conversion Options 3](#_Toc447660757)

[3 Requirements List 4](#_Toc447660758)

[3.1 Data Conversion 4](#_Toc447660759)

[3.2 Data View 4](#_Toc447660760)

[3.3 Accessibility and Simplicity 4](#_Toc447660761)

[3.4 Graphical User Interface 4](#_Toc447660762)

[3.5 Command Line Arguments 5](#_Toc447660763)

[4 System Modeling 5](#_Toc447660764)

[4.1 Finite State Automata of Parsing and Output Processes 5](#_Toc447660765)

[4.2 Command-Line Use Case Diagram 6](#_Toc447660766)

[4.3 Graphical User Interface Use Case Diagram 7](#_Toc447660767)

[4.4 Library Use Case Diagram 8](#_Toc447660768)

[5 Prospective Planning 8](#_Toc447660769)

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| VERSION | TEAM MEMBER | DESCRIPTION | | DATE |
| 0.1.0a | Dylan Barnes | Initial Document Layout and Introduction | 01/29/2016 | |
|  |  | Add Sub-Clause 2.1 *Data Conversion*  Add Sub-Clause 2.2 *Data File Contents*  Add Sub-Clause 2.3 *Conversion Options*  Add Sub-Clause 3.1 *Data Conversion*  Add Sub-Clause 3.2 *Data View*  Add Sub-Clause 3.3 *Accessibility and Simplicity*  Add Sub-Clause 3.4 *Graphical User Interface*  Add Sub-Clause 3.5 *Command Line Arguments* |  | |
| 0.2.0a | Dylan Barnes | Add Page Numbers  Add Table of Contents  Correct Ambiguous Grammar | 02/18/2016 | |
| 0.3.0a | Dylan Barnes  Ryan Kilbride | Add Main Clause 4 *System Modeling*  Add Sub-Clause 4.1 *FSA of Parsing and Output Processes*  Add FSA Diagram 4-1 to Main-Clause 4  Update Requirements in Main-Clause 3  Correct Ambiguous Grammar  General Grammar Improvements | 03/07/2016 | |
|  |  |  |  | |
| 0.4.0a | Cameron Kerbaugh | Correct Grammar and Punctuation  Clarify Requirements Details | 03/22/2016 | |
| 0.5.0a | Dylan Barnes | Add UC Diagram 4-2 to Main-Clause 4  Add UC Diagram 4-3 to Main-Clause 4  Add UC Diagram 4-4 to Main-Clause 4 | 04/05/2016 | |
|  |  |  |  | |

# Introduction

DC# Converter is a portable, multi-function conversion tool. It is part of a larger open source software suite, known as Data Control Sharp (DC#). Main features include: simple file conversion using a graphical user interface (GUI), C# classes, or command line arguments; a list view display of the data contained within converted files; and various options to modify converted data. All of these features are explored in further detail in the “Requirements List” section.

This document will explain everyday issues that DC# Converter plans to solve, as well as a list of requirements and features necessary to accomplish aforementioned goals. These requirements will be technically expanded in future documentation; this document simply provides a high-level overview.

# Problem Analysis and Solutions

## Data Conversion

In past experiences, users may have had to use third party packages to parse content from one format into another, such as CSV to XML, for example. However, if they needed to parse the content into yet another format, this required a different package altogether, along with an entirely new set of syntax. Even worse, if the user had to program in a different programming language, they’d need an entirely new set of packages to perform the parsing and conversion.

DC# Converter plans to solve this conundrum. The primary goal of this application is to take in various forms of file types and allow the user to parse or convert the contents into another file type. In order to allow cross-language use, our application will support parsing via command-line arguments. All that needs to be provided is the input file, output file type, and any additional options the user prefers. In addition to this, multiple file types are supported. This means the user will simply need to use one package, DC#, to parse multiple types of input files.

## Data File Contents

Due to the variety of file types and different standards, understanding the content structure within a file can be difficult. If a user wishes to see the contents of a data file but does not understand the syntax, it is typically up to an external tool to solve this.

DC# Converter intends to include this functionality whenever a user parses content. Obviously, the data contained within a data file would be best displayed in the GUI, but there will be command-line and class support as well. With this feature, a list view of the contents of the file will be organized and displayed, allowing the user to easily see the content within the file they are converting.

## Conversion Options

In standard conversion systems, if you wish to modify parsed content, you typically parse the content first and then manually modify the elements before writing to a new format.

DC# Converter plans to merge this functionality into the conversion process. Options will be provided that allow users to select ways to modify the file’s content—such as removing white space—before conversion, adapting the output to meet their unique needs.

# Requirements List

The following requirements list will describe the features and requirements that will be in the initial release of DC# Converter. For the sake of readability, the requirements will be grouped based on their similarities.

## Data Conversion

* The system shall accept various forms of input for conversion, such as XML, JSON, and CSV.
* Input files may also be pulled directly from a web URL.
* Functions shall be called via a GUI, in-language function call, or command-line arguments.
* The system shall contain various options for modifying the output:
  + Remove white space
  + Specify root node
  + Append metadata
* The system shall perform internal data parsing.
  + If using DC# Converter in C#, you can use exclusive methods that allow you to obtain the parsed data of a file as an internal data type.

## Data View

* The system shall contain a GUI (Graphical User Interface).
  + After converting the data, the parsed content will be displayed in an additional window as a list view object. Details are provided in the Graphical User Interface section.
* The system shall contain command-line functionality.
  + The content will be displayed as command line standard output.
* The system shall be usable through functions as part of a C# Library.
  + The content may be parsed directly into native C# data types.

## Accessibility and Simplicity

* The system shall have 3 forms of accessibility:
  + A custom-built GUI.
  + A command line interface, callable from other programs and languages.
  + Function use directly from another C# class.
* The system shall contain simple function calls:
  + One function call to convert to a specific output format.
  + Auto-detect input file type and parse accordingly.

## Graphical User Interface

* A GUI shall be included with DC# Converter.
* The user interface shall contain a simple design that:
  + Allows users to easily manipulate the GUI.
  + Includes all core functionality:
    - Convert to and from all supported data types.
    - View data structure and contents.
* The user shall “browse” to select an input file.
* Once an input file is selected, the user shall select the output type via a drop-down menu.
* Additional options in the form of check boxes and drop-down menus may be selected:
  + Remove white space
  + Specify root node
  + Append metadata
* A “Convert” button shall be pressed once everything has been selected, effectively converting the file. The file shall be placed into a user-selected directory.
* A list view of the data shall be displayed in an additional window, which the user may close with an “OK” button.
* The system shall be designed with the C# programming language.

## Command Line Arguments

* The command line system shall allow the DC# functionality to be used outside of C# and the GUI.
* The command line system shall contain all the same functionality the GUI does.
* The command line system shall follow the methodology of the function calls – simplicity.
  + This simplicity involves a single call to convert any supported input type into a specified output type.
  + Additional options such as removal of blank space may be added here as well.
* The command line system shall not include data parsing capabilities. This is only available when using the C# library.

# System Modeling

## Finite State Automata of Parsing and Output Processes

Figure 4-1 contains the finite state automate that describes the parsing and output processes, which constitutes the majority of our functionality.



Figure 4-1: Finite State Automata diagram for the parsing and output processes. States and state changes are color coded to reduce redundancy. It can be assumed that all yellow processes denote CSV actions, green denotes JSON actions, and orange denotes XML actions.

## Command-Line Use Case Diagram

Figure 4-2 depicts the UML use case diagram for interacting with DC# Converter via a command-line.

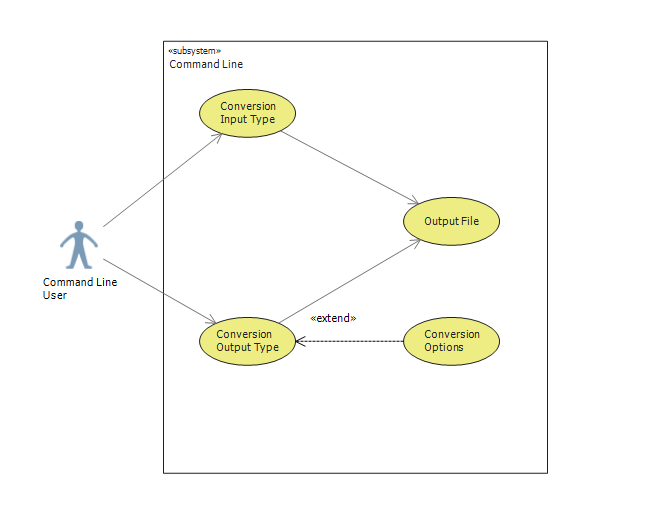


Figure 4-2: Use Case Name: Command Line. Use Case Purpose: The purpose of this use case is to display how the authorized actors (i.e., users) would use DC# Converter functionality to convert data files using the command-line. Precondition: User knows how to use a command-line. Post conditions: Varies depending on the action performed. While a successful action always results in a new file, the input and output is dependent on what the user selects. Constraints: Data view functionality is not included in the command-line. Assumptions: Supported conversion types are CSV, JSON, and XML.

## Graphical User Interface Use Case Diagram

Figure 4-3 depicts the UML use case diagram for interacting with DC# Converter via the packaged graphical user interface.

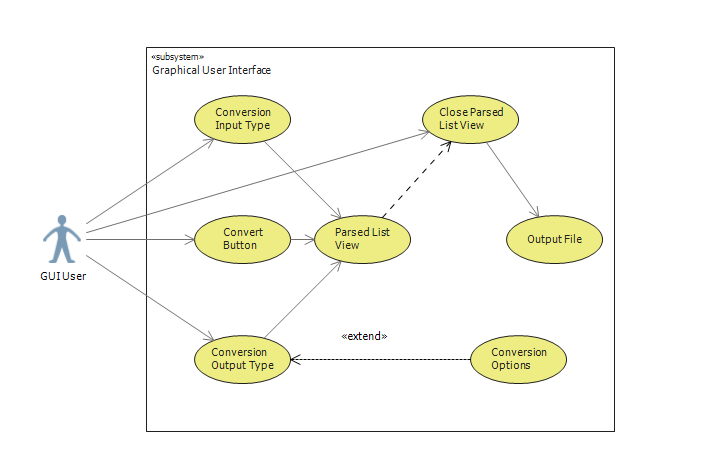


Figure 4-3: Use Case Name: Graphical User Interface. Use Case Purpose: The purpose of this use case is to display how the authorized actors (i.e., users) would use DC# Converter functionality to convert data files using the GUI. Precondition: User is launching the GUI via an executable file. Post conditions: Varies depending on the action performed. While a successful action always results in a new file, the input and output is dependent on what the user selects. Constraints: None. Assumptions: Supported conversion types are CSV, JSON, and XML.

## Library Use Case Diagram

Figure 4-4 depicts the UML use case diagram for interacting with DC# Converter via the .dll file, also known as a C# library.

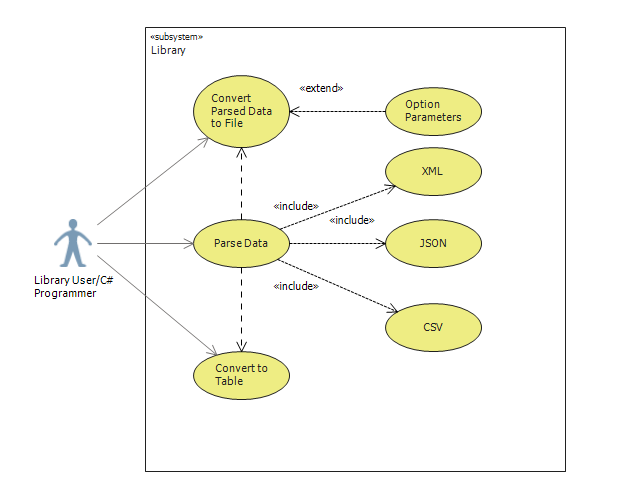


Figure 4-4: Use Case Name: Library. Use Case Purpose: The purpose of this use case is to display how the authorized actors (i.e., users) would use DC# Converter functionality to parse and convert data files. Precondition: User knows how to program in C# and use a DLL file. Post conditions: Varies depending on the action performed. With the library, a user may parse content without converting it to a new file. Constraints: C# Language. Assumptions: None.

# Prospective Planning

The requirements list above describes the features that are planned for release by the end of the Spring 2016 semester. Once completed, additional features will be added to the DC# suite that allow multiple applications within the suite to take advantage of each other. For example, DC# Manager will be able to convert numerous file types by using the methods provided by DC# Converter. After we have added additional features and ensured stability, the DC# software suite will be released as open source software.