Design Specification

Version 1.0

WibTeX

Reference Management System

Sponsor:

Dr. James Palmer

Development Team:

Hayden Aupperle

Jarid Bredemeier

Charles Duso

March 20, 2017

Contents

Background

When publishing research in the field of computer science it is common to be required to use BibTeX and LaTeX to construct documents for publication. LaTeX is a document preparation system designed by Leslie Lamport in 1985 that uses a markup language to structure documents [1]. BibTeX is a reference management system, created in 1985 by Oren Patashnik and Leslie Lamport as a method to construct reference data for documents prepared in LaTeX [2].

When publishing research outside of the computer science field, however, it is commonly required to use Microsoft Word to construct documents for publication. Microsoft Word is a word processor developed by Microsoft for use in constructing documents [3]. Researchers who wish to publish across scientific fields will find themselves a time-consuming endeavor when trying to transition their bibliographic work constructed in BibTeX to a format that is suitable in Microsoft Word. Unfortunately, there does not yet exist a solution for the efficient transfer of bibliographic data constructed in BibTeX to Microsoft Word.

1 Introduction

As a result of the issue described in the previous section, the WibTeX development team has designed the WibTeX Reference Management System to simplify the process of preparing documents for cross-discipline research publications by eliminating the need to reconstruct bibliographic information produced in BibTeX to a format suitable for Microsoft Word. WibTeX will allow the user to construct reference pages and in-text citations sourced from BibTeX bibliographic information from within a Microsoft Word Document, using a LaTeX workflow.

1.1 Purpose

The purpose of this document is to describe the implementation details of the WibTeX Reference Management System. These implementation details will include technologies of choice, architectural design of the system, module and interface descriptions of the architectural components, and a plan for the implementation of the system.

1.2 Scope

This Design Specification is to be used by the WibTeX development team as a definition of the design to allow for the implementation of the WibTeX Reference Management Software.

1.3 Requirements and Constraints

The purpose of this section is to briefly state the requirements and constraints of the system as determined through the software conception phase. This documentation will serve to provide a clear expectation of what is required of the product upon release.

1.3.1 Functional Requirements

This section includes the requirements that specify all fundamental actions of the WibTeX Reference Management System.

Specify Style File:	The user should be able to specify the style file they want to use for their document. The user will either specify the style file by way of path in the command line, or through graphical user-interface via drop-down menu.
Extend Style File:	The user should be able to create additional style files from a generic and well-documented template. *This also implies that the user can modify existing style files.*
Complete Style File:	The style file should be complete in that it provides coverage for all resource types supported by a citation style.
CCSC Style Support:	The system should support the Consortium for Computing Sciences in Colleges reference style.
Specify BibTeX Database:	The user should be able to specify the BibTeX database, containing a list of references the user plans to cite within their document. The user will either specify the database by way of path in the command line, or through graphical user-interface via drop-down menu.

Complete BibTeX Parser:	The system should be able to support and interpret all resources, fields, and special characters that are valid components of the BibTeX syntax (version 0.99d).
Specify Microsoft Document:	The user shall be able to specify the Microsoft Word document they wish to be read from and the Microsoft Word document they wish to write to. *Microsoft Word documents that are not in the .docx format will not be supported.*
LaTeX Syntax Support:	The system should support valid LaTeX syntax used to generate a reference within a Microsoft Word document from BibTeX databases. *The system must not support all valid LaTeX syntax – only those that interface with BibTeX.*
Error Handling Support:	The system should provide error handling tools that correctly locate the source of the error – whether in the BibTeX file, style file, or Microsoft Word document – and accurately communicate the file and location of error to the user.

TABLE 1: FUNCTIONAL REQUIREMENTS

1.3.2 Performance Requirements

Below we will list the performance requirements of the WibTeX Reference Management System. We have constructed performance requirements as thresholds of expectation for components of the systems that we have confidence in achieving with high probability. We will consider the response time and scalability for components of the system.

1.3.2.1 Response Time Requirements

Within this section, we consider the response times of components in the system. Response time requirements envelop all actions by the system that can possibly be conducted by the user. This definition excludes our testing suite and other such functionality that will not be utilized by the primary user. We have constructed a table below that lists the components of the WibTeX system that are of interest for response time.

Component	Metric	Response Time
User Interface	Any event that can be triggered by interface interaction	0.1 to 1.0 seconds until program executes appropriate action related to triggered event
Command Line	Any valid command that can be executed using WibTeX	0.1 to 1.0 seconds until program returns response to user
Document Render	Execution of the render action after sufficient resources have been provided	1.0 to 30.0 seconds until render is complete

TABLE 2: RESPONSE TIME REQUIREMENTS

1.3.2.2 Scalability Requirements

We will now consider the scalability requirements for the system. We have determined that our system should support the response times listed in the previous section up to a worst-case scenario in which the user has a 100-page Word document and 1,000 references to account for. Although we do not expect any user to reach or exceed the worst-case scenario we have derived, we will still aim to achieve our desired response times should the situation occur.

1.3.3 General Constraints

In this section, we will consider the general constraints for potential users who wish to use the reference management system. These constraints derive out of a need to satisfy legal requirements as well as ensuring a quality product that does not require deprecated software packages and technologies to support previous versions of Microsoft Word, BibTeX, LaTeX, or various operating systems.

Operating System Limitations:	The system shall need one of the following operating systems installed; Windows 10, macOS, or Ubuntu 16.04 LTS. *Later versions of the specified operating systems will also be supported.*
Hardware Limitations:	The system shall meet the hardware requirements of the respective operating system installed on the user's computer. Additional hardware or upgrades are not required of the user.
Licensed Microsoft Office or Word:	The user shall own a valid license of Microsoft Office or Microsoft Word and have sufficient means to create, modify, and store Word documents.
*Python 3.5.x Installation:	The system must have Python version 3.5.x or greater. * *This constraint is potentially irrelevant as the end-goal of our product is to create an executable for each platform – thus not requiring Python installation.*

TABLE 3: GENERAL CONSTRAINTS

2 Implementation Overview

In this section, we will document our general approach to the implementation of the WibTeX reference management system, as well as, the tools and technologies that will be the driver for the construction of the system. As a reminder, we intend for our system to construct a reference page and in-text citations (that conform to a specified citation style), within a Microsoft Word document, using a BibTeX database. This functionality requires that we derive solutions to parse BibTeX databases, parse and manipulate Microsoft Word documents, and parse and interpret a file containing citation style data.

2.1 Design Approach

We have determined that the ideal approach to the construction of our system would be to design a monolithic architecture that employs modularity achieved through object-oriented programming. This means that the final product would be a single, cohesive unit with a distribution of tasks assigned to each module, increasing performance by removing the need of inter-process communication. As we do not intend to expand the system to account for new

features, we are not hindered by the tight coupling that can plague software designed with a monolithic architecture.

2.2 Technologies Used

Listed below is the programming languages, and packages that we plan to use to construct the system. We can say with confidence that the following items will remain throughout the implementation of the system, but there may be additional tools used and so this document will be updated accordingly.

2.2.1 Programming Language - Python 3.4.x

We chose Python version 3.4.x as our programming language of choice for several reasons:

- Python contains several pre-existing packages that support BibTeX database parsing, document templating, and XML parsing (necessary for manipulating Word documents)
- Python is a language that all members of our group are comfortable programming with
- Python allows for object-oriented programming - which is necessary for our design architecture
- Programs in Python can be easily prototyped and tested, allowing for a more agile design process

2.2.2 Package - Jinja2

Jinja2 is a package for Python that offers a robust templating engine [4]. A templating engine allows for the generation of human-readable data that can be dynamically substituted - so long as the data being represented maintains a specific form. Jinja2 enables us to generate the reference pages and in-text citations for a Microsoft Word document. We achieve this by constructing the template for the reference pages and in-text citations within our style file, and using that template within the Microsoft Word document we can dynamically fill-in reference data.

2.2.3 Package - BibtexParser

BibtexParser is a package for Python that allows for simple parsing of BibTeX databases [5]. Using the BibtexParser package, we can quickly parse a BibTeX database and store the reference data in a more suitable format. Once we have the reference data, we can then dynamically fill-in

the contents of the structures produced by Jinja2 within the Microsoft Word document we are managing.

2.2.4 Package - TkInter

TkInter is a package for Python that allows for the construction of lightweight graphical user interfaces [6]. With TkInter we will construct the user interface that will act as the gateway to execution of the WibTeX Reference Management System.

2.2.5 Package - PythonXml

XML package is a custom Office Open XML implementation written in Python that allows WibTeX to decompress, parse, extract information, save, and compress Microsoft Word files [7].

2.2.6 Package - PyInstaller

PyInstaller is a Python package that is designed to generate executables from Python programs, containing third-party packages [8]. This enables us to freely use external Python packages such as Jinja2 and BibtexParser without having to accommodate for said packages to produce a functional executable.

3 Architectural Overview

Within this section, we will discuss the high-level architecture of the WibTeX Reference Management System. This section will include an architectural diagram to summarize the overall function of the system, and a textual description of the architectural diagram to clearly define the purpose and functionality of each component that constitutes the system.

3.1 Architectural Diagram

Listed below is the architectural diagram for the WibTeX Reference Management System.

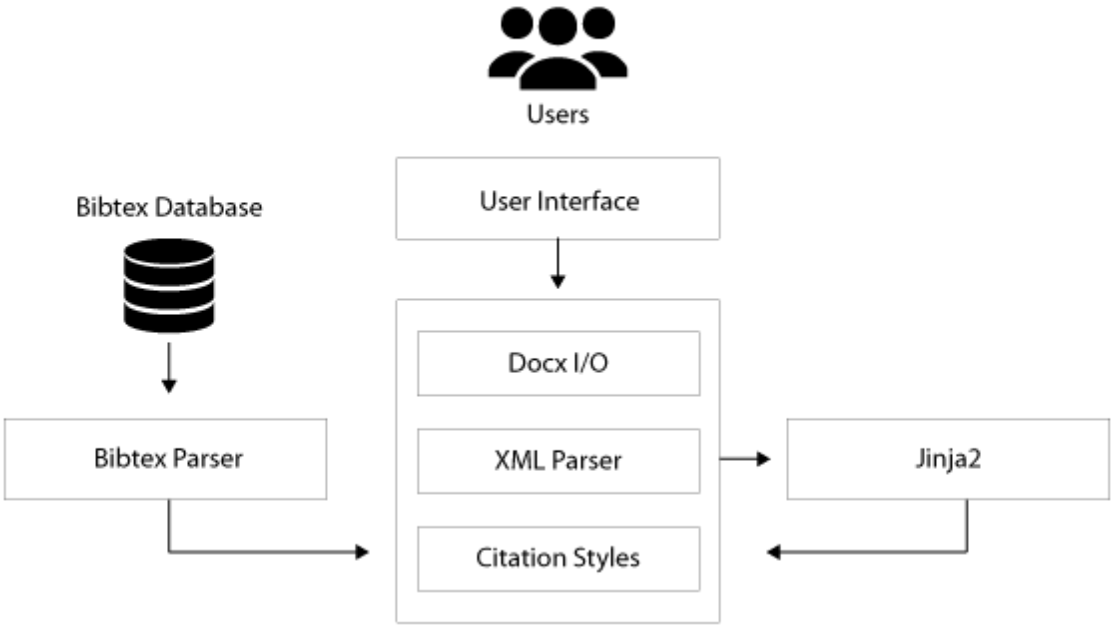

FIGURE 1: SYSTEM ARCHITECTURE

3.2 Component-level Description

In this section, we will describe the components of the system architecture.

- User interface is the junction between the user and the WibTeX Reference Manag ement System
    - It allows the user to target a BibTeX database and docx document for processing
- A BibTeX database is read into the system by the BibTeX parser
- Docx I/O decompresses and opens a docx document and then extracts the XML content while parsing Latex markup into a data structure
- XML Parser applies styles to the BibTeX data and updates the information inside the data structure
- Jinja2 framework is used to replace the Latex markup inside a docx XML with the markup processed by the XML parser
    - Docx I/O then writes a copy of the original docx file with the applied changes, compresses and saves the files into a new docx document

4 Module and Interface Descriptions

We will now consider the system at the module level with a class diagram, modeling the whole system and descriptions of each class that constitute the system.

4.1 Class Diagram

Listed below is the class diagram for the WibTeX Reference Management System. The system is relatively condensed; this is true for several reasons: third-party packages absolve our need to account for several portions of the system that would otherwise involve significant design, and the task itself is not a complicated process. It is for these reasons that we are able to produce a system that is lightweight, but effective.

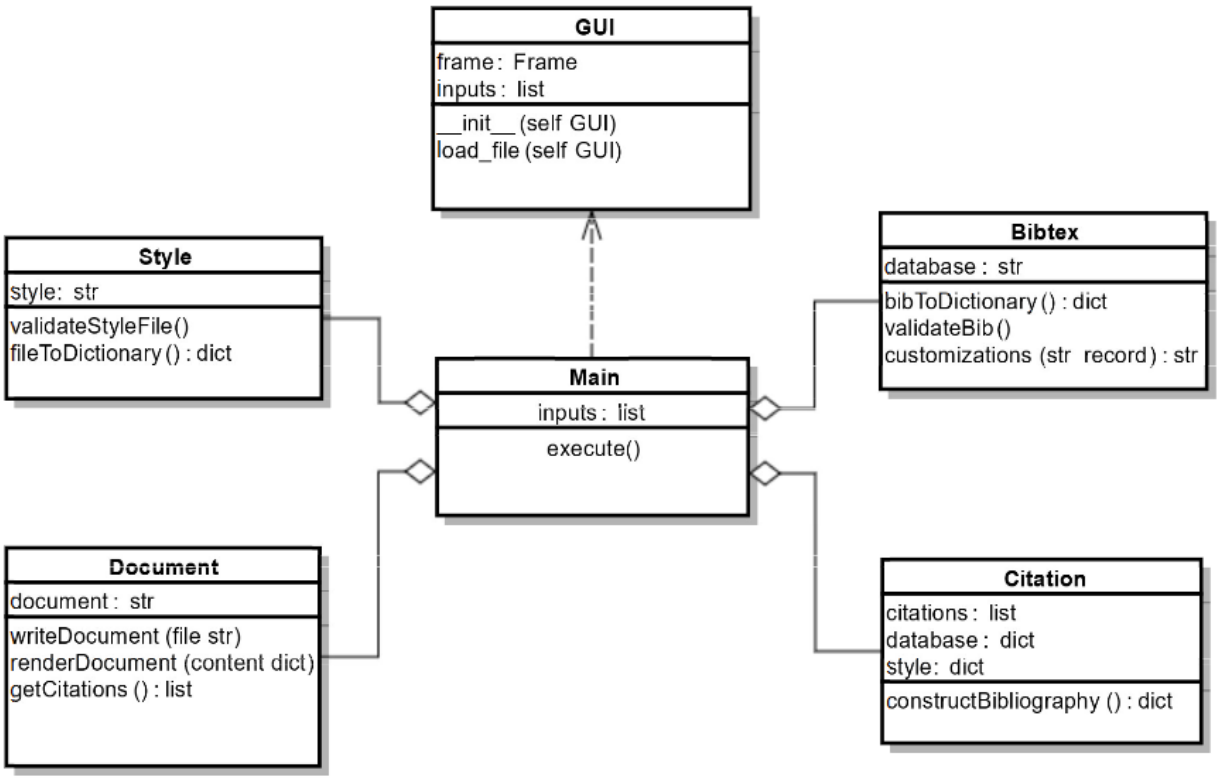

FIGURE 2: SYSTEM-WIDE CLASS DIAGRAM

4.2 Bibtex Class – Bibtex.py

The Bibtex class provides the system with the ability to parse BibTeX databases and extract necessary information to a Python dictionary. The Bibtex class uses the BibtexParser package to

complete the primary parsing task of the class, but other methods have been designed by the WibTeX team to account for special characters, invalid syntax, and multiple authors.

4.2.1 Function – bibToDictionary

Purpose:	Primary function of the Bibtex class that extracts valid BibTeX data for use in the WibTeX Reference Management System
Inputs:	None
Outputs:	Python dictionary containing BibTeX reference data organized by entry in the supplied BibTeX database
Called by:	The Main class
Calls:	Bibtex.validateBib()
Algorithm:	• Ensure that the file provided when the class was initialized is a valid BibTeX database • Supply the custom callback function to the BibTeX parser from the BibtexParser package • Parse BibTeX file and retrieve relevant data • Return Python dictionary

4.2.2 Function – validateBib

Purpose:	Validates a BibTeX database, ensuring proper fields for each entry based on its resource
Inputs:	None
Outputs:	None
Called by:	Bibtex.bibToDictionary()
Calls:	None

Algorithm:
- Parse supplied BibTeX database to ensure that it conforms to the BibTeX syntax and standards
  - Respond appropriately if database is not valid
- Ensure that special characters are properly escaped

4.2.3 Function – customizations

Purpose: Acts as a callback function to handle multiple authors in a BibTeX database entry after the parser has retrieved a valid entry

Inputs: record, a Python string containing the entry data from a listed field in the entry

Outputs: record, a Python string containing the entry data from a listed field in the entry

Called by: BibtexParser.load()

Calls: None

Algorithm:
- On callback, retrieve the record stored in the current field of the current entry
- Store this record in a list

4.3 Style Class – Style.py

The Style class provides the system with a means of extracting style data that the user has specified. After having extracted the style data, the system is able to use said data to manipulate references to conform with the standards set by the specified style.

4.3.1 Function – validateStyleFile

Purpose: Validates a style file to ensure that it conforms with the standards that the WibTeX team has deemed necessary

Inputs: None

Outputs: None

Called by: Style.fileToDictionary()

Calls: None

Algorithm:
- Parse supplied style file and verify that file complies with standards
    - Inform the user if the file does not conform with the standards

4.3.2 Function – fileToDictionary

Purpose: Converts the data contained in a style file into a Python dictionary

Inputs: None

Outputs: Python dictionary

Called by: Style.fileToDictionary()

Calls: None

Algorithm:
- Parse style file as a typical JSON file, and retrieve and store information in a Python dictionary

4.4 Citation Class – Citation.py

The Citation class enables the system to dynamically construct a reference page template that will later be processed by the Jinja2 engine and converted to a valid reference page for a Microsoft Word document, conforming to the standards set by the user's specified style file.

4.4.1 Function – constructBibliography

Purpose: Construct a reference page template suitable for use with the Jinja2 templating engine

Inputs: None

Outputs: Python dictionary

Called by: Style.fileToDictionary()

Calls: None

Algorithm:
- For each resource that was cited in the Microsoft Word document
    - Construct a valid citation template and append appropriate

variables for the resource that was cited to later be filled in by the Jinja2 templating engine

- Return the constructed dictionary

4.5 Document Class – Document.py

The Document class acts as the primary communicator between the user's specified Microsoft Word document and the WibTeX Reference Management System. The Document class is capable of reading from documents, writing to documents, and creating new documents. The Document class uses the Jinja2 templating engine to render templates embedded within a document.

4.5.1 Function – writeDocument

Purpose:	Write a Microsoft Word document with valid compression
Inputs:	file, a Python string representing the file name of the document to write
Outputs:	None
Called by:	The Main class
Calls:	None
Algorithm:	• Save data to file using PythonXml package with appropriate compression level for a Microsoft Word document

4.5.2 Function – renderDocument

Purpose:	Renders template data using Jinja2
Inputs:	content, a Python dictionary containing data to be substituted for expressions embedded
Outputs:	None
Called by:	The Main class
Calls:	• Jinja2.render() • Document.writeDocument()

Algorithm:
- Initialize Jinja2 engine class
- Supply Python dictionary
- Render document

4.5.3 Function – getCitations

Purpose: Determines the number of unique citations and their entry tokens from the Microsoft Word document that the user specified as input

Inputs: None

Outputs: A Python list containing unique citations from the document in the order that they appeared

Called by: The Main class

Calls: None

Algorithm:
- Parse the document
    - For every citation contained within the BibTeX token that signifies a citation: *'/cite{…}'*
        - Store the citation in a list if it has not yet been referenced

4.6 GUI Class – GUI.py

A lightweight GUI that initializes the primary execution of the WibTeX Reference Management System, providing necessary input data.

4.6.1 Function – load_file

Purpose: Allows the user to specify input files for the WibTeX Reference Management System to execute on

Inputs: self, the class's instance of itself

Outputs: None

Called by: None

Calls: None

Algorithm:
- As user supplies input data, modify input list so that system has valid information

4.7 Main Class – Main.py

The Main class acts as the control of the system, creating all other classes and calling their respective functions.

4.7.1 Function – execute

Purpose: Execute the system

Inputs: self, the class's instance of itself

Outputs: None

Called by: None

Calls:
- Bibtex.bibToDictionary()
- Style.fileToDictionary()
- Document.renderDocument()
- Document.getCitations()
- Document.writeDocument()
- Citation.constructBibliography()

5 Implementation Plan

This section serves to detail our plans of implementation for the WibTeX Reference Management System. This detailing includes a Gantt chart listing the intended dates of completion for each assignment and the distribution of work amongst each team member. We will also explain our approach to the schedule we have derived, an approach modeled from the major development phases that have been designated.

5.1 Implementation Schedule

Listed below is the Gantt chart we have created and intend to follow. We must note, however, that the schedule is subject to change and any updates will be noted within this document in future revisions.

FIGURE 3: IMPLEMENTATION SCHEDULE

5.2 Scheduling Discussion

Our schedule starts on January 24th, 2017 when we were first assigned the project. We started construction of the design document the first week of February 2017, but our primary focus had been implementation details. We had also started construction of the design review presentation in the first week of February as both the design document and presentation are similar. In-between these two documents, we wrote a short abstract to detail the project for when we present at the Undergraduate Symposium held at Northern Arizona University. Following the completion of the abstract, we then set out to complete the initial design document draft and the design presentation. This leads us to the point in time where we are currently situated as of the date for this document.

Moving forward, we aim to complete the final draft of the design document within a week of our submission of the draft: March 7, 2017. As for the system itself, we have not set a finalized date for the completed project, but we intend to have a functionally complete demo by March 17, 2017. Following this date, assuming there was no hindrance in our plans, we will then enter a

testing and finalization phase that will lead to the completion of the system at some point in April 2017 – before the Undergraduate Symposium.

6 Conclusion

The WibTeX Reference Management System simplifies the process of publishing documents across different scientific fields. Researchers - especially those in the field of computer science - often create written publications through LaTeX and BibTeX. If a researcher who works in the LaTeX and BibTeX environment wishes to publish across different scientific fields, they may be required to create documents in Microsoft Word. The transition from LaTeX and BibTeX can be a time-consuming endeavor as the reference material created in BibTeX does not easily transition to Microsoft Word. Fortunately, WibTeX aims to remediate this problem by allowing the user to retain their BibTeX bibliographic information in its original format and use that information as they would in a LaTeX environment; but, for Microsoft Word documents. This task will be accomplished through a monolithic architecture built in Python, using several third-party packages. We are optimistic for our design and the release dates we have prescribed.

Bibliography

[1] LaTeX – A document preparation system. (2017). Retrieved March 5, 2017 from https://www.latex-project.org/

[2] Alexander Feder. 2006. BibTeX. (2006). Retrieved March 5, 2017 from http://www.bibtex.org/

[3] Word. (2017). Retrieved March 6, 2017 from https://products.office.com/en-us/word

[4] Armin Ronacher. 2008. Welcome to Jinja2. (2008). Retrieved March 6, 2017 from http://jinja.pocoo.org/docs/2.9/

[5] François Boulogne. 2014. Welcome to BibtexParser's documentation!. (2014). Retrieved March 6, 2017 from http://bibtexparser.readthedocs.io/en/v0.6.2/

[6] Tkinter Wiki. (March 2014). Retrieved March 6, 2017 from http://tkinter.unpythonic.net/wiki/

[7] PythonXml. (January 2012). Retrieved March 7, 2017 from https://wiki.python.org/moin/PythonXml

[8] Welcome to PyInstaller official website. (February 2013). Retrieved March 7, 2017 from http://www.pyinstaller.org/