DESIGN & IMPLEMENTATION OF A PDF TO EXCEL CONVERSION TOOL (P2X)

By

LATOYIA DEVONNE PENNY

Dual Bachelor of Science in

Mathematics and Computer Science

Langston University

Langston, Oklahoma

1999

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2009

DESIGN AND IMPLEMENTATION OF A PDF TO EXCEL CONVERSION TOOL (P2X)

Thesis Approved:

Dr. K. M. George

Thesis Adviser

Dr. N. Park

Dr. G. Hedrick

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to extend my sincere appreciation to my graduate advisor, Dr. K.M. George for his constructive research guidance, intellectual advice, and support. My appreciation also extends to my other committee members Dr. Hedrick and Dr. Park whose assistance and guidance has been indescribable. I would like to thank Dr. George and the staff of the Department of Computer Science at OSU, Stillwater for providing me with this research opportunity and their generous financial support. Moreover, I wish to thank the OC-ALC/ENET and 552 CSS/SCT and SCZ organizations at Tinker Air Force Base, and the Information Technology Department at OSU/OKC for giving me the opportunity to advance in my career, for the learning experiences, and for financial support.

I would also like to express my sincere gratitude to those who provided suggestions, assistance, and advice while inspiring me throughout my research: Uncle Eddy and Aunt Vi, Grandma Ruth, Aunt Leota (Deceased), Kenneth Simmons, Leander Johnson, Liz Ruckwied, Carol Drink, Major John K. Pringle, James Harned, and my father Ollie.

I would like to give my special appreciation and dedication to my siblings LaTonja, LaTrishia, Justin, David, my babies Jada, Javen, and Jlonna, and my heart, Quenita, for her unbiased suggestions, her strong encouragement in times of difficulty, and support that only she could articulate. Thank you to my loving mother, Donna Marie, for her undying love, support, encouragement, and for being a major influence throughout my life. True love is unconditional.

Lastly, I would like to thank God my father in heaven from whom all blessings flow. Without Him, none of this would have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Nomenclature/Definition of Terms

3PP	three phase process
AAR	Adobe Acrobat Reader®
AFF	American Fact Finder
Batch	a large amount of files
CD	MS DOS change directory command
C.S.	computer science
Delimiters	characters inserted to delimit columns in tables of valid PDF documents
Java	Sun Microsystems® java object-oriented programming language
Mod1	module 1 application
Mod2	module 2 application
MS Excel	Microsoft® Excel software application
Out1	output file 1
PDF	portable document format
PTF	plain text file
RMA	records management application
VB6	Microsoft® Visual Basic 6.0

CHAPTER I

INTRODUCTION

The need to make major advances in the area of data conversion processes has been expressed by various organizations [8]. In this day and time, the use of Adobe Acrobat's© portable document format (PDF) is a popular choice amongst most for the distribution of significant publications and documents. The popularity and ubiquity of PDF, causes the ability to retain significant data quick, relatively inexpensive, and without complexity to become attractive commodities for organizations in reaching greater flexibility and predicting higher standards for data utilization [5].

By the use of document conversion, organizations and ordinary users alike will have the ability to capture the PDF data and transform it to a more suitable format to achieve their ultimate goals. While other document formats are available, and will be discussed, converting the PDF documents has an advantage because it allows the user to modify significant data. However, this process can be considerately tedious and time consuming when done manually. This in-turn has an adverse effect on the amount of money spent to complete the task.

In the following chapter Review of Literature; we will briefly discuss conversion programs that have previously been done on various document formats and processes. In Chapter III, Methodology, the problem that has been researched is described and problems that the study resolves are presented. Finally, the thesis concludes with the results of the research and future works are exposed.

Problem Statement

Currently, researchers express the need to make major advances in methodology and applications in the area of data conversion processes [16]. Organizations produce data for publication and use by other government agencies, individuals, and private sector organizations, alike. Much of the data is then processed and used for the individuals or organizations personal publications. Problems persist for customers when the need for processing large amounts of the published data into manageable formats in an effort to improve records management arises. The financial cost of solving these problems is also a known burden. And because of the diversity of the customer, the less complex the solution the better!

Purpose of Study

Through program implementation, testing experiments and examples the goal of this study is to generate an economical, automated, user-friendly means to process, convert, and record batches of PDF documents into tabular format for improved document conversion. The implementation of the new developments presented in this study, along with an increasing knowledge of the research topic and literature review; has yielded expectations that many will consider to be a novel approach in the promotion of innovative applications systems for data acquisition, processing, conversion, and portability.

Objectives of the Study

The objective of this research is to develop a more novel approach to convert documents in PDF format to editable form.

Scope and Limitations

The scope of this study is limited to focus on an implementation of a conversion tool (P2X); developed to automatically convert large batches of PDF tabular data (PDF tables) to spreadsheet format (MS Excel).

CHAPTER II

REVIEW OF LITERATURE

There are several commercial and non-commercial PDF converters available offering to extract text from PDF and represent the output as either plain text or some other document format such as Excel [2, 3, 9, 12, 14 and 18]. They all either lack the ability to *automatically* handle large amounts of data, distort the original PDF document layout, exhibit the incapability to process large amounts of data in a timely manner, or they advertise functional capability in exchange for capital gain. Timeliness and costliness has equally been an issue in retrieving digital portable document file data and transferring the data into editable documents.

Adobe PDF to HTML Converter

Adobe's PDF to HTML converter is the most popular converter to result when queried through the Google search engine via the internet [9]. It is a simple online text flow utility that attempts to preserve text formatting such as font but disregards all other structural formatting of the converted document. The output file produced by this application does not mimic the original PDF file, and in particular ignores page breaks. Also, this conversion tool lacks the direct ability to convert the resulting files into an Excel spreadsheet because it only produces the HTML document equivalent to the original PDF file.

PDFtoText

An exceptional conversion tool known as the PDFtoText© was created to generate the text equivalent of the original PDF document to be converted. PDFtoText© software program is a command line conversion tool [12]. This tool was incorporated as part of this research to reduce redundancy during the initial programming phase and because it was freeware, reducing costliness. This application is designed to convert PDF documents into ASCII text format with a minimal loss of formatting [12]. Unfortunately, the term minimal used here is dependent on the actual document being converted. While the conversion tool produces the text form of PDF documents, the text documents made with the application does not maintain the exact layout of its original counterpart, either. Also, the tool only generates one output text file for each PDF document submitted. Larger text files take more time to process. To combat the timing problem, it was decided to reproduce the resulting PDFtoText© text file as an MS Word document. Numerous modifications had to be made to the generated text file. These will be further discussed in Chapter III of this research.

Manual Conversion Method

The task of successfully exporting data from Portable Document Format into Microsoft Excel spreadsheet manually has previously been accomplished. Adobe Acrobat Reader freeware has been used to export tables from PDF documents into Excel [3]. They even went a step further and published the procedures via the World Wide Web. Unfortunately, the software application will only make conversions page by page, the Adobe Acrobat Reader freeware has limited editing options, and it lacks the versatility of simultaneously converting batches of documents automatically. The invention of high-end technological software advancements and automation has rendered the manual way as obsolete.

The Influence of Herman Hollerith

Automated innovations have been evident as far back as the late 1800's when automation was first introduced into the world of data collection, manipulation, and collation [21]. Herman Hollerith is credited with inventing and patenting punched cards [21]. He also notably created an electrical counting machine, the Tabulating Machine; to mechanically read, sort, and organize the punched cards [21]. These cards represented accountability of data and as a result allowed organizations the flexibility to automate their fundamental data processes. That was over a century ago. Currently, technology is presumed more advanced with smaller equipment and the added convenience of software to perform enhanced data processes, namely document conversion.

Commercial Software

Many commercial companies exist today, offering portable automated system (PAS) applications and analysis tools that will capture a portable document file and export it into MS Word, MS Excel, and RTF to name a few [2, 3, 9, 12, 14 and 18]. But they all either lack the ability to handle large amounts of data, exhibit the incapability to process large amounts of data in a timely manner, or they offer various functional capabilities at unaffordable prices. Processing speed has equally been an issue in retrieving digital portable document file data and transferring the data into meaningfully editable documents.

3 Heights™ PDF Extract API is an automated extraction component that can extract and retrieve information from PDF documents [14]. "PDF Tools AG offers stand-alone tools and libraries, along with extensions and consulting services to deliver customer-specific solutions" [14]. The 3 Heights™ PDF Extract API also offers several options for extracting data from the digital files with very little emphasis on converting the data into other portable formats. Although, the company that distributes the application, PDF Tools AG, has extended its services by

offering a free trial to validate some of the components' functionalities, they require a client license of $266 to $399, and a server license of $1330 to $1995 to use their software development kits. Consequently, the amount of overhead still exists. The user is left with prohibitive developmental processing time even after purchasing the 3 Heights™ PDF Extract API products.

Convert Doc

Like PDFtoText©, Convert Doc by SoftInterface, Inc. is a simple to use, yet sophisticated document conversion utility. If there is a need to convert thousands of documents with a variety of file types located in many folders in a short period of time, this is the tool [20]. Convert Doc (CD) can be ran from the command line allowing for use in batch files or can be launched from within other programs. PDF, Text, RTF and HTML are among the formats Convert Doc customers convert their files to and from. As previously stated, Convert Doc is not only an executable application, but also an ActiveX component. That is, the file, ConvertDoc.EXE, can be ran as a stand alone application, and, can be referenced as a component within a separate development environment. Though CD possesses many useful attributes, we are interested in the generated output results. CD output presents major loss of formatting when converting PDF documents to text, which is coincidentally another common factor that Convert Doc and PDFtoText© share. P2X resolves this issue.

On-Demand PDF to HTML

Sommerer achieves significant improvements to the quality of on-demand PDF to HTML conversion at insignificant costs in terms of increased file size and processing time [18]. This work was noteworthy in that Sommerer shows a slightly more advanced HTML coding that compensates for file size increases when including line graphics and images. Unfortunately, the visual representation of PDF files in HTML is usually very poor. His study weaned away from simplicity in that it was more focused on graphics and images and less on tabular data. Although Sommerer introduces a section for text extraction, the study offers more essence in the areas of HTML optimization, web services, and search engines. The focus of this research is PDF to editable format conversion representations using an automated conversion tool to implement the conversion, and customer/user satisfaction through the use of a GUI model.

XML and XFDL

Alternatively, the option of outputting paper-based tabular data and forms documents into other commonly used document formats namely the extensible markup language, XML can achieve transaction non-repudiation to formulate electronic records to maintain the data's structure and for use on the Internet. Table 1 highlights some of the more commonly used document formats, the organizations which maintain those document's standards, and their source/format types.

Doc Type	Organization	Open Source	Proprietary
XML	W3 Consortium	✓	
XFDL	W3 Consortium	✓	
TXT		✓	
HTML	W3 Consortium	✓	
RTF	Microsoft Corporation		✓
DOC	Microsoft Corporation		✓
XLS	Microsoft Corporation		✓
PDF	Adobe Systems Incorporated	✓	
PPT	Microsoft Corporation		✓
FPK	Adobe/Accelio/JetForm Corporation	✓	
ZIP	PKWARE Incorporated (Phil Katz)	✓	
ODF	Organization for the Advancement of Structured Information Standards (OASIS)	✓	

Table 1: Commonly used document format types and their organizations.

XML is heralded as a key enabling technology involving the integration of structured data, having standard syntax for creating and exchanging data structures into business-to-business transactions [4, 7, and 15]. The most common use of the XML framework is web-based e-commerce. The emergence of e-commerce for technological development is known to enable transactions of structured data from business-to-customer to business-to-business relationships, effectively.

The Extensible Forms Description Language (XFDL) is an application of XML that allows organizations to move their paper-based forms systems to the Internet while maintaining the original attributes of paper-based transaction records. XFD relates to PDF in that they are both widely recognized and provide an open, extensible structure for data exchange [17]. They are also comparable in that PDF captures a paper-based document while maintaining the documents original format, while XFD provides similar functionality for web-based electronic forms documents [1, 10, 11, 16, 17 and 19]. On the contrary, proprietary or prohibited formats from a business standpoint are costly, inefficient, and risky [6]. Many organizations have strict rules governing the maintenance of transaction records that can also restrict automation [1, 10, 11, 16, 17 and 19].

CHAPTER III

METHODOLOGY

Overview

In this chapter, we describe the methodology of the research. We will begin by introducing the PDF specification standards on table structure. Next we give a scenario example of the problem and a description of the conversion tool (P2X) architecture. Then, specific details of the algorithms and applications used during the PDF to plain text format (PTF) conversion process follows. A brief overview of the reformatting process and a formalization of the table tags that we identified using regular expressions will be introduced. Lastly, a description of the GUI, its images, and functionality will be discussed in the User Interface section.

PDF Specification

The PDF specification published by Adobe® Acrobat®, available via web, provides a description of the PDF file format and is intended for application developers preparing to develop applications that create PDF files directly, as well as read or modify PDF document content [13]. The PDF specification states that a standard PDF document allows for file attachment inclusion. Specifically, a table document can be annotated into a PDF documents code as an object and referenced with particular field types, similar to HTML code. Basic layout modes are considered in the specification to further annotate the ways and locations of where the object can be displayed. Block, top to bottom, and layout modes all references progression direction of the object. Block progression directions is trivial while layout mode considers the before and after edges of a reference area inline progression direction and start to end edges of the reference area. In retrospect, the PDF spec also explains the standard structure types for grouping, block level, paragraph-like, and table elements [13]. The PDF spec also contributes a section to the common existence of strongly structured vs. weakly structured PDF documents [13]; implicating the possibility of lackluster portability from a documents original destination, thru PDF, to another format. It would be exceedingly difficult to compare with limited knowledge of PDF structures and attributes for the varied PDF styles. Comparing the documents code structures any further is beyond the scope of this research. The next section will further demonstrate extenuating measures that were accomplished in continuation of our research.

P2X Architecture

To illustrate the problem we include the following scenario: The application is needed to convert PDF documents which include data about Agriculture in the U.S. The goal of the application is to integrate all information in a computer program spreadsheet that will allow one to easily capture the data for manipulation and to provide a Java interface for ease of use.

Refer to Figure 1. This figure formally exhibits the P2X architecture. Using P2X, the application programmer first locates the documents by importing a Java program which provides validation and an OO view of the portable document format (PDF) data. An imported generic conversion program provides a plain text format (PTF) view of the PDF data. Using a graphical user interface, these programs are combined and customized for this specific application, providing a single *unified* conversion (A) from PDF to PTF. The application design does not expose details of the conversions and concentrates on their integration through some high level representation. Then, a PTF to HTML conversion program (B) is imported and customized in a similar way. If the application requires a different GUI display for any states agriculture or subtopic of agriculture, the program can be further customized to provide different behaviors.

Particularly, it is assumed that the PTF data were materialized. It is also possible for the PTF data to be processed virtually, i.e. the conversions to the final Excel output and from the original PDF objects are composed to yield a one-step conversion program and the existence of a PTF is unknown to the user. Type checking is used by the system to verify the coherence of the conversions and in particular, the coherence of their composition.

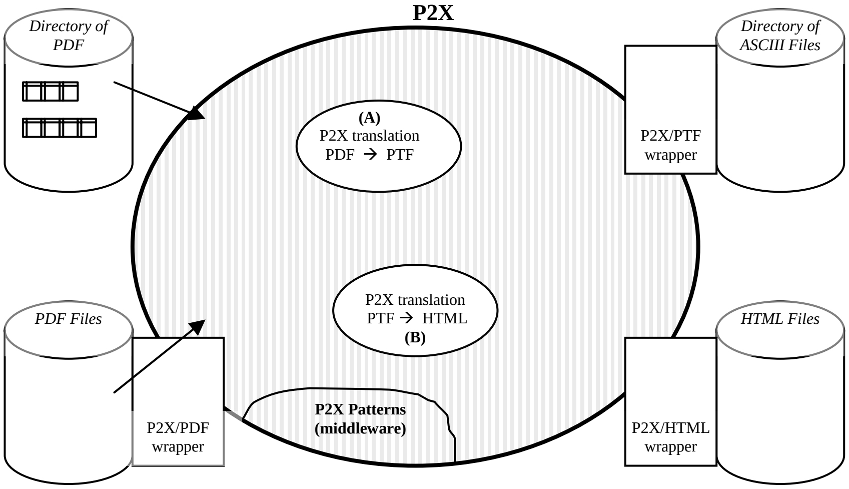


Figure 1: P2X GUI Architecture.

The system, called P2X is designed to be the "offspring" of a mediator/wrapper system. The current version of P2X allows for materializing the pre-converted PTF and the target data (Excel) representations. A complimentary goal of this work would be to query the data without fully materializing all converted steps of the process. However, the focus of this work is on conversions, with emphasis on implementing a more novel approach than what is usually considered. Efficient database storage and querying of the target Excel data representations without fully materializing all conversion steps and the management of updates of both source (PDF) and target (Excel) data will be considered in future works.

Plainly, P2X relies on this data model to allow a rich and structured representation of the significant PDF data.

PDF to PTF Conversion

In continuing with the problems stated, details of the PDF to text conversion will now be introduced. The necessity to specify in depth and further explain the problem of modifying the PDFtoText© output previously discussed in the Related Work section of this study begins with the explicit details of the PDF to PTF conversion process.

PDFtoText Description

To combat the format/layout problems previously mentioned, it was decided to reproduce the text file into another usable format namely PTF. To illustrate this process, Figure 2 shows the usage parameters associated with the PDFtoText© command line entry.

Usage Parameters and Options

```
pdftotext [options] <PDF-file> [<text-file>]
```

Figure 2: Usage parameters entered using Derek B. Noonburg's PDFtoText© app to convert PDF docs to text.

Figure 2 indicates the usage parameters that appear when the user types "pdftotext" on the command line prompt of the directory where the PDFtoText© executable is located. The first entry into PDFtoText© on the command line indicates the applications executable file being called to administer the conversion, as mentioned previously. The [options] tag indicates what options the user wishes to execute during the conversion process. Some options chosen will be processed during conversion and will indicate how the user would like the output file layout. Other options are only related to the actual PDFtoText© tool application data. Table 2 briefly depicts the limited options that can be used at the command prompt in conjunction with the application tool and its usage parameters.

PDFtoText Usage Options	
Options	**Description**
-f <int>	: first page to convert
-l <int>	: last page to convert
-ascii7	: convert to 7-bit ASCII
-latin2	: convert to ISO Latin-2 character set
-latin5	: convert to ISO Latin-5 character set

-eucjp	: convert Japanese text to EUC-JP
-raw	: keep strings in content stream order
-upw <string>	: user password (for encrypted files)
-q	: don't print any messages or error
-v	: print copyright and version information
-h	: print usage information
-help	: print usage information

Table 2: PDFtoText' optional entries and their descriptions.

Next, the PDF file entry enclosed in the greater than and less than symbols indicates the name of the PDF file to be converted. And lastly, the text file indicates the name of the output file to be produced. This name is chosen by the programmer and is associated with the original PDF file for identification purposes.

As previously stated, once the PDF file is converted using the PDFtoText© conversion tool, there is still much work to be done to maintain the original document's layout and produce the final PTF.

PDF Document Layout Issues

The layout of the original PDF documents contains tremendous amounts of information including headers and footers, maps, borders, and other visual data that coexists with the tabular data that is the focus of the research.

For clarity purposes, the output file explained earlier will be referred to as Out1. Out1 contains the tabular data in text format. To help eliminate layout issues, a java function was produced to sift through unwanted data. In other words, the HEADER_DEPLETION and NEW_HEADER_DEPLETION functions "trims" the PDF document down to tabular/relevant data only, i.e. Out1. These functions are further described in Appendix A. For the remainder of this research, relevant data refers to all data represented in tabular format on the original PDF document. To illustrate the process, first P2X creates an instance of Out1. The tool then creates a new directory with the associated file name, in this case, Out1. This introduces a proper naming convention and presents good records management. Once the new directory is created, the new instance is eliminated and the old file is moved to the newly created directory for safe keeping.

Consequently, when Out1 is properly stored, we generate another application, CDTMod1, to produce a unique copy of the Out1 file in HTML format. This application is created in Visual Basic 6.0 programming language to import .doc (MS Word) files and convert them to HTML. It was convenient to include the CDTMod1 application since this application was produced by the author of this research prior to this research; the application was amended to the P2X to reduce production time and redundancy. Once the HTML files are stored, the MS DOS command line function XCOPY command is used within the JAVA runtime executable to convert the HTML file to text. This new .txt file captures the original PDF document's HTML source code; i.e. the PTF. Examples of these processes are shown in the Reformatting Process section of this research.

The Reformatting Process

During the reformatting phase of the study, another application had to be developed to perform in conjunction with the PDF-to-Text conversion tool to edit "irrelevant" data that resides in the converted text file. For the scope of this thesis, this process will be referred to as "trimming". For the duration of the thesis, the "significant data" will also be referred to as the core. This section includes a description of the trimming process, a depiction of the "core" of the document, and a sample illustration of the P2X process. Figures are also included for visual aid.

By converting the initial PDF document to Unicode first, the goal to capture core data can be achieved. This is largely due to the merging of the Unicode values of the html source code tags contained in each source code file generated. Any modern Word document can be converted to HTML by simply changing the extension of the file name from .doc to an .html document. This automated process was executed with the implementation of a Visual Basic macro executable called CDTMod1. Although it may seem trivial, by doing so, the document does not capture the format of the original text. It does, however, produce an HTML file for the purpose of obtaining the source code or a "tagged snapshot" of the document so that the original data can be extracted. Furthermore, with these HTML attributes, the documents encompass the capability to be re-published online. Nonetheless, online publication deviates from the scope of the research and will be addressed for future works.

The notable tags recognized in this study, were the <p class=MsoPlainText></p> tags. As mentioned earlier, core data is data that is in tabular format. Although it can be accomplished, we could not fully rely on the standard HTML TABLE tags from the HTML source code to identify significant data typically because many PDF documents do not use those tags to clearly identify tabular data; one of the drawbacks of HTML. We could, alternatively, rely on the data contained inside of the <p class=MsoPlainText></p> tags to pinpoint the focal area of the core. Each table was represented by a unique identifier (the merged Unicode values) allowing us to focus on that as the primary key for identifying the initialization of the core data. Each character in a document has a corresponding Unicode value. Once the beginning of the tabular data was identified, the source code data was processed for its Unicode value and based on when the particular open tags <p class=MsoPlainText> and close tags </p> were reached, we could capture the core data that was contained within those tags character by character. We then had to "trim" away the data that was immaterial leaving only the core data for each record in the document to be dumped as the final output. For precision, only one record was processed at a time. This, consequently, verified that the information processed was managed appropriately and maintained with accuracy.

Table Tags

In addition to the reformatting process, it was appropriate to include what major factors exist for us to determine the possibility of capturing tabular data from the PDF documents. Those factors we refer to as the table tags. Once identified, these key factors make the conversion of PDF tabular data into Excel spreadsheet format attainable.

There were several primary identifiers discovered while finding the tabular data in the html source code files once they were converted from PDF. After the documents had become source code, we were able to establish five distinguishing identifiers to direct us to the core data. The combination of the five basic table tags produces a regular expression. Regular expressions and finite automata were used here to formally describe the process of identifying core tabular data. As a result of researching the patterns, terminal, and non-terminals strings, the following regular expression evolved:

REGEX: <div class =.* | <p class=MsoPlainText>Table.*|<o:p>&nbsp;</o:p>.*|<p.*>|</p>|>

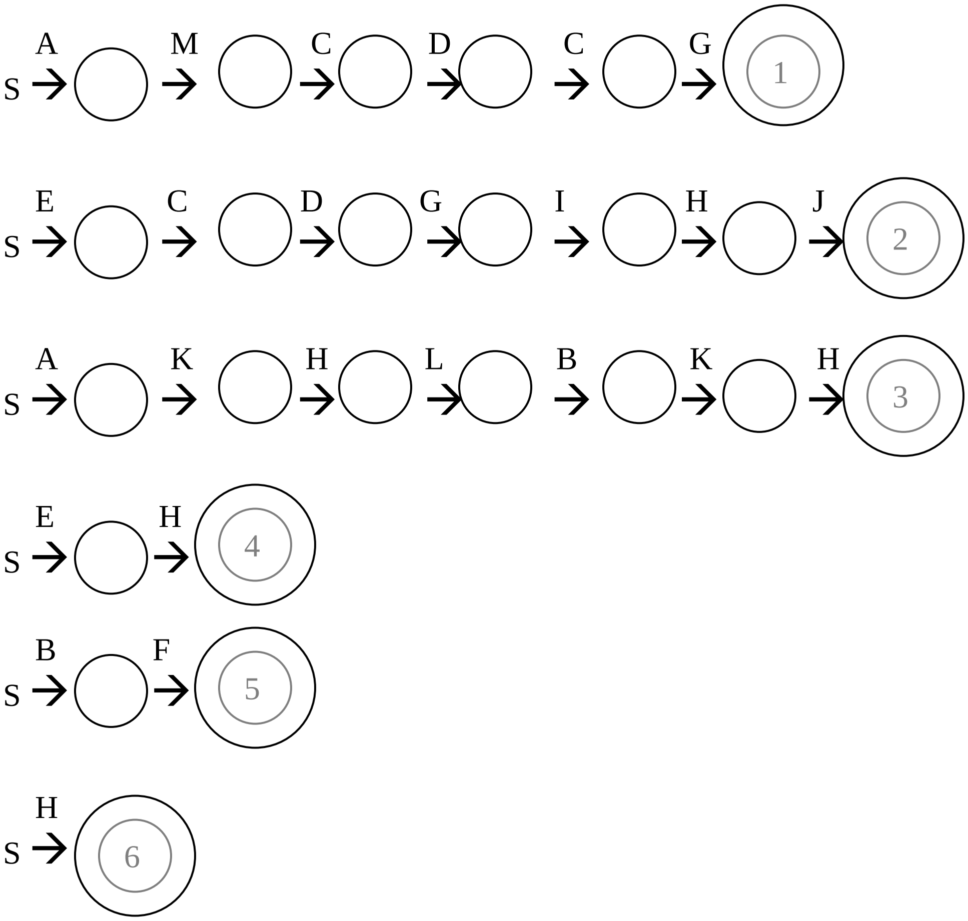
The list of patterns, terminal strings, and non-terminal strings:

A: <
B: </
C: whitespace
D: class
E: <p
F: p>
G: =
H: >
I: MsoPlainText
J: Table
K: o:p
L: &nbsp;
M: div

An equivalent regex using the previous list exposes the following regex:

REGEX: AMCDCG|ECDGIHJ|AKHLBKH|EH|BF|H
(1) (2) (3) (4) (5) (6)

The Finite Automata which corresponds to the previous regex follows:



These are the main criteria of a valid document’s structure that identifies the core data once the PDF document has been converted to HTML. Exception and error handling has been implemented and will continue to be enforced. However, the critical issues that were initially discovered with finding the core data have been resolved.

The significant documents that had to be researched thoroughly in order to find the tags were the PDF document, source code (.txt) document, the output document after final conversion (after trimming) but before conversion to Excel, and the documents directory structure. The P2X program code was re-modified to verify the tag's validity. Random files were chosen and researched/reviewed during the testing and analysis phases in order to come to these conclusions.

The random documents that were chosen for testing were reviewed and compared, smaller documents were tested thoroughly first, then larger files followed. Document management was also implemented in the program code where each document generated individual directories and each page of a PDF document was stored as a separate text file, for storage, review, research, and modulation purposes. There has been findings where extra data was included in certain output files which also helped in further identifying exceptions. Extra data included headers, footers, and some pages that were not tabular data or core data. These extra data helped in alleviating some of the originally thought of identifiers during earlier stages of the research. The non-tabular data will not properly store into Excel spreadsheet cells, and in-turn, does not properly store into the output documents that are used as inputs into Excel. Although not intended as the focus, those non-tabular data will be referenced as "ill formatted data" throughout the span of the research.

Graphical User Interface

The P2X graphical user interface is an advanced JAVA application that allows user interaction to convert PDF tabular data into MS Excel with the click of a button. This section will include a step-by-step process to run the P2X system. The P2X has two interactive buttons that the user will push to maintain P2X functionality; IMPORT and CLOSE. A visual of the GUI is included in Figure 9 of

Appendix D to support the P2X system processes. Ultimately, the P2X algebraic description is exposed for clarity.

Run P2X

This section lists the instructions to run P2X in a step-by-step process. A detailed manual is included in Appendix D.

Step 1: From the DOS command prompt, change the directory to the location of the P2X executable file (Type: cd c:\PennysResearch; refer to Figure 8).

Step 2: Compile the program by typing: javac DirectorySetup.java

Step 3: Run the program by typing: java DirectorySetup

Step 3.1 - P2X Tool FORM1 Press **Import** (Refer to Figure 10).

Step 3.2 - CDTMOD1-5-04 states the pathname of the file to be processed. If the path is correct, user selects ok (Refer to Figure 10).

Step 3.3 - When the document is complete, the Form 1 box will flash and the Import button will disappear. Select the **Close** button (Refer to Figure 11).

Step 4: Repeat Steps 3.1 through 3.3 for every document in the directory tree.

Step 5: Open the PennysResearch directory to view the *Progress Log* and verify the correct documents were processed.

Step 6: Open the USCENSUSBUREAU directory to review all processed documents.

P2X GUI Description

GUI: = STEPS 1+2+3+4 or GUI: = VL +CD+JC+RJ

Figure 3: P2X GUI Description.

Figure 3 shows an algorithmic description of the P2X graphical user interface. In order to achieve the P2X GUI, the user must follow a sequence of commands:

```
1: Validate File Storage (VL)
2: Change Directory to P2X executable location (CD)
3: Compile the P2X application - javac (JC)
4: Run the P2X application - java (RJ)
```

CHAPTER IV

CONCLUSION

Discussion of Conversion

In this thesis, a novel approach to improve the process of converting PDF documents to a more editable form has been presented and evaluated. To reduce issues of conversion formatting, processing time, and costliness, we have implemented a working conversion tool to show the conversion of PDF tabular data to MS Excel spreadsheets can be simple by use of a graphical user interface with user interaction. This system was produced using the high-level programming languages Java and Visual Basic 6.0. These implementations are presented in Appendices A, B, and C. A user's manual has been incorporated to validate the use of the system and reduce user error. The manual is conveniently located in Appendix D. Appendix D includes more visuals of the P2X tool to further assist the user with the problems presented throughout this research.

Although P2X proved to be a successful conversion approach, it was discovered at the end of the final testing phase that the final output of the text data stored in the Excel spreadsheet file will need minimal manual editing by the user to dispose of unwanted non-breaking space and to suit the individual user’s storage preferences. These preferences are expected to vary on a case-by-case basis.

Future Work

Future works has been presented to further demonstrate the usefulness of this thesis towards future technological advancements in the data conversion discipline.

Documents do not have to be snapshots of a point in time. By pulling together technologies such as XML-based authoring and publishing tools and connecting to live data, documents can become dynamic and interactive — alive in a sense. Rather than creating and then updating the same document with different versions, a document can really begin to work by pulling live data so that it always stays up-to-date.

Structured authoring already provides the benefit of writing once and publishing multiple times for a deliverable like software help manuals. By combining this with the power of live data, more complex deliverables can be provided like standard operating procedures and complex technical manuals for capital equipment and regulated processes where information must be disseminated with the context and the persistence of a document – but where the cost of static information is unacceptably high. As stated, these documents are subject to ongoing change, as complex arrays of data within sources of record are updated. So with live, interactive documents, the risk of rework or redesign costs, launch delays, non-compliance, or most importantly, putting inaccurate or out-of-date information in the hands of the end user can be avoided. As stated earlier, storing data into a relational database such as MS Access can further enhance the P2X tool by adding the user capability of data querying. Updating Visual Basic 6.0 to Visual Basic.NET, database modeling and SaaS (Software as a Service) deployment model are also future motivations.

BIBLIOGRAPHIC REFERENCES

1. Chapman, N. and Chapman, J. *Digital multimedia, second edition*. John Wiley & Sons, Ltd., San Francisco, 2004.

2. Chellapilla, K., Simard, P. and Radoslav, N. Fast optical character recognition through glyph hashing for document conversion. in *8th International Conference on Document Analysis and Recognition*, (Seoul, S. Korea, 2005), IEEE Computer Society, 829-834.

3. Cohene, T. and Khouri, A. How to export a table from a PDF file into an Excel spreadsheet. Electronic Data Resources Services, Feb 2002. Retrieved Dec 16, 2002, from McGill University: http://www.mcgill.ca/edrs/services/information/help-tools/howto/pdf_to_other/.

4. Harold, E.R. *XML bible, second edition*. John Wiley & Sons, Ltd., San Francisco, 2001.

5. Irwin, K. and Swenson, K. D. Workflow technology: tradeoffs for business process reengineering. in *proceedings of the Conference on Organizational Computing Systems* (Milpitas, CA 1995), 22-29.

6. Khare, R. and Rifkin, A. Capturing the state of distributed systems with XML. *World Wide Web Journal*, 2 (4). 207-217. Retrieved April 6, 2007, from O'Reilly Media, Incorporated: http://www.xml.com/pub/a/w3j/s3.khare.html.

7. Knox, R. *XML: What is it and Why Should Users Care?* Harvard School of Business Press, Boston, 2006.

8. Kobayashi, M. and Takeda K. Information retrieval on the web. *ACM Computing Surveys*, *32* (2). 144-173.

9. Margulies, B. Big dots, little dots, and circled dots: How Unicode can help (and hurt) the process of converting documents to information. Retrieved November 21, 2006, from Basis Technology Corporation, Cambridge, Massachusetts:

http://www.basistech.com/knowledge-center/unicode/big_dots_little_dots.pdf.

10. Microsoft Corporation. *Microsoft computer dictionary, fifth edition*. Microsoft Press, Redmond, 2002.

11. Microsoft Corporation. *Microsoft internet & networking dictionary*. Microsoft Press, Redmond, 2003.

12. Noonburg, D.B. PDF-to-Text, Sanface, Nov 1998. Retrieved August 4, 2003, from AimNet.com: http://www.aimnet.com/~derekn/xpdf/.

13. PDF Reference, Sixth Edition, 1993. Retrieved November 9, 2006, from Adobe Systems Inc.: http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

14. PDF Tools AG: Company Background, 2001. from Premium PDF Technology http://www.pdf-tools.com/asp/about-us.asp?lang=en.

15. Phillips, L.A. *Special edition using XML*. Que Publishing, Indianapolis, 2000.

16. Research Opportunities at the U.S. Census Bureau, 2007. Retrieved January 8, 2008, from U.S. Census Bureau, The ASA/NSF/Census Bureau Research Fellow Program: http://www.census.gov/srd/research.pdf.

17. Sethi, I.K., Khosla, R., and Damiani, E. *Intelligent multimedia multi-agent systems: A human-centered approach*. Kluwer Academic Publishers, Norwell, 2000.

18. Sommerer, R. Presentable document format: Improved on-demand PDF to HTML conversion, *Microsoft Research Technical Reports 2004*, 2004. Retrieved March 14, 2007, from Microsoft Research Publications: http://research.microsoft.com/research/pubs/view.aspx?tr_id=824.

19. Tozer, G.V. *The role of metadata on the internet metadata management for information control and business success*. Artech House Publications, Boston, 1999.

20. User's Guide for Convert Doc vers. 3.X, 2006. Retrieved August 13, 2006, from SoftInterface, Inc.:

http://www.convert-files.com/SII/Convert-DOC/English/PDF/ConvertDoc.pdf.

21. Utley, B. The digital revolution - The impact of Herman Hollerith, *Technology Evangelist Digest*, *12*, Dec 2005. Retrieved May 24, 2006: http://www.technologyevangelist.com/2005/12/the_digital_revoluti.html.

CHAPTER VI

APPENDICES

Appendix A

P2X Program Code

```
import java.io.*;
import java.awt.*;
import javax.swing.*;
import java.lang.String;
import java.lang.Object;
import java.util.Scanner;
import java.util.Vector;
/**********************************************************************
 Program Author: LaToyia DeVonne Penny
 Date of Completion: June 17, 2008
 Program Description:
      This program is an application formed in conjunction with the
thesis created in partial completion of the Master of Science at the
Oklahoma State University in Stillwater, Oklahoma.  The thesis is
entitled "Implementation & Analogy of the DESIGN & IMPLEMENTATION OF A
PDF TO EXCEL CONVERSION TOOL (P2X) By LaToyia DeVonne Penny.  The
program produces a file management system for the United States Census
Bureau PDF files and formally converts them to Microsoft Excel
Applications in Spreadsheet format.  The program was formulated in
several modules that were merged to create the P2X tool.  Module 1
consists of the Directory Setup class and functions.  Module 2 entails
the parsing process.  This section of the program takes the PDF files
from the generated directory from Module 1 using the VBSETUP and
GetUnival classes, their functions, and the CDTMOD1 Visual Basic
application to parse the files based on specific characters in the HTML
source code of each document.  These characters identify the location
of the "significant" data that is the focus of this study.  The parsed
documents are then transformed into output text files that maintain the
original PDF document format.  Module 3 consists of the
ExcelConvert2008 Visual Basic application the implementation process of
taking the converted documents as input into the Microsoft Excel
application to finish processing the parsed text files producing the MS
Excel final products.

Program Function List:
      main
      RUN
```

```
Dir_List
      PARSE_OP
      HEADER_DEPLETION
      NEW_PG_HEADER_DEPLETION
      N_CRITICAL_OR_CRITICAL

Program Class List:
      P2X
      VBSETUP
      GetUniVal

**********************************************************************/

class P2X extends JFrame{

      PrintWriter pro_tracker;
      PrintWriter current;
      File [] hold;
      boolean success, exists;
      String made_dir="";
      VBSETUP vbsu;
      GetUniVal guv = new GetUniVal();


/*Create a GUI with a Windows Environment look and feel*/
P2X() throws IOException {

try {


//SET THE LOOK AND FEEL OF THE FRAME TO THE SYSTEMS SETTINGS

UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
SwingUtilities.updateComponentTreeUI(this);

} catch(Exception e) {

 System.out.println("ERROR 1****");
 System.out.println("This doesn't work due to: " +e.getMessage()+"!!");
 e.printStackTrace();

}//end try catch

}//end P2X constructor


/*Main method to call the function to initialize the P2X process*/
public static void main (String[] args) throws IOException
{
     P2X frame = new P2X();
     Process p;

     //Size the frame.
     frame.pack();

     //Centers frame onscreen
     frame.setLocationRelativeTo(null);
     //Show it.
     frame.setVisible(false);

     frame.RUN();                     //Run the P2X program


     frame.dispose();

     try{

      /**Started Excel Module June 14, 2008***Created ExcelConvert2008 VB6
      App to convert final txt to Excel**********/


      /*Run VB*/
      p = Runtime.getRuntime().exec("c:\\Program Files\\Microsoft
          Visual Studio\\VB98\\VB6.exe /run
          c:\\PennysResearch\\ExcelConvert2008\\ExcelConvert2008.exe");

      p.waitFor();


      } catch (Exception e) {

            System.out.println(e.getMessage());
            e.printStackTrace();

      }//end try catch

}//end main method


/*Function to initialize a directory and file listing of the current
  system*/
public void RUN() throws IOException
{
     File dir = new File("c://PennysResearch/USCENSUSBUREAU");
     File [] dir_list = dir.listFiles();
     int num_files;


     pro_tracker = new PrintWriter( new BufferedWriter(new
                   FileWriter("ProgressLog.dat")), true);

     current = new PrintWriter( new BufferedWriter(new
                   FileWriter("c://PennysResearch/currentDir.dat")),
                   true);

     if(dir_list == null)
         System.out.println("No Files Exist in "+ dir +"!");
     else
     {

         if(dir_list != null)
            hold = Dir_List(dir_list);   //Produce a directory listing
     }//end if else




}//end RUN method


/*Function that stores the list of files and directories*/
/*and sends them to VBSETUP for further processing*/
public File[] Dir_List( File[] d_list) throws IOException
{
     int i=0, k;           //Initialize the element identifier of the
                           //directory list
     String filename="";
     String dir_index ="";


     try{

     if( d_list == null ) {
         System.out.println("ERROR2****: Cannot get a list of files.");
             } else {

                  //Traverse through d_list's files and directories
                  for( i=0; i<d_list.length; i++ )                        {

                     if((d_list[i].isDirectory()))
                     {
                        File dir1 = new File(d_list[i].toString());
                        File [] d_list_new = dir1.listFiles();
                        hold = Dir_List(d_list_new);
                     }
                     else
                     {
                        if((d_list[i].isFile()) &&
                      ((d_list[i].getAbsolutePath()).endsWith(".pdf")))
                      {
                        // Initiate directory and exists variables;
                        //all ancestor directories must exist
                        made_dir = (d_list[i].getAbsolutePath()).substring(0,
                        ((d_list[i].getAbsolutePath()).length() - 4));

                        exists = (new File(made_dir)).exists();

                         //If the directory does not exist, then create
                         //one and process it
                         if (!exists)
                         {
                              //send absolute path name to output file
                              current.println(made_dir);
                        success = (new File(made_dir)).mkdir();
                              vbsu = new VBSETUP(made_dir);

                              /****************************************
                              * GETUNIVAL.JAVA BEGINS...WILL BEGIN    *
                              * PROCESS TO ACQUIRE THE UNICODE        *
                              * VALUE OF EACH CHARACTER GENERATED FROM*
                              * EACH HTML SOURCE CODE DOCUMENT        *
                               * PRODUCED BY LOADING THE DIRECTORY    *
                               * SETUP MODULE. THE PROGRAM WILL THEN  *
                               * GENERATE A TEXT FILE THAT WILL BE AN *
                               * EXACT SNAPSHOT OF THE ORIGINAL       *
                               * PORTABLE DOCUMENT FORMAT (PDF) FILE  *
                               * AND CONCLUDE BY GENERATING THE EXCEL *
                               * DOCUMENT OF THAT ORIGINAL PDF FILE.  *
                               *    JULY 12, 2006 by LaToyia D. Penny *
                               ***************************************/


                              k= made_dir.lastIndexOf("\\");
                              filename = made_dir.substring(k+1);

                              guv.PARSE_OP(made_dir, filename);


                              /*****GETUNIVAL PROCESS COMPLETE*****/

                         }//end if-else statement

                      }//end if-else statement


                   }//end if-else statement



                  //Process progress information to the progress log
                  //output file
                   if(d_list[i] != null)
                       pro_tracker.println(d_list[i] +" Process
                                                       Completed...");
             else
                        JOptionPane.showMessageDialog(null, d_list[i]+"
                                                          is empty!");

                  }//end for loop

           }//end else

           } catch (Exception e) {

              System.out.println("ERROR 2****");
              System.out.println("This doesn't work due to: "
              +e.getMessage()+"!!");
              e.printStackTrace();

         }//end try catch

         return hold;   //return last visited directory

      }//end method Dir_List

}//end class P2X :)
```

```
/*Class to begin conversion processes and calls the Visual Basic Application
to convert files into various file formats to maintain original document
layout*/
class VBSETUP extends JFrame
{
     File file, dir;
     boolean success;
     Process p;


     /*VBSETUP constructor - utilizes the current directory from the
       directory list*/
    VBSETUP (String curr_dir) throws IOException{

    file = new File(curr_dir+".txt");
    success = file.createNewFile();

    /*Begins process to open the Visual Basic application*/
    String [] commandLine = {"pdftotext", "-layout", curr_dir+".pdf",
                              curr_dir+".doc"};

    try {

    p = Runtime.getRuntime().exec(commandLine);  //Run PDFTOTEXT
    p.waitFor();


    /************MS WORD DOCUMENT MANAGEMENT***********************/

            // Create an instance of the MS Word file
            file = new File(curr_dir+".doc");

            // New destination directory
            dir = new File(curr_dir);

            // Move the word file to the new directory
            success = file.renameTo(new File(dir, file.getName()));

            // Delete the old MS Word file
            success = (new File(curr_dir+".doc")).delete();

    /*************************************************************/


    /*Run VB*/
    p = Runtime.getRuntime().exec("c:\\PennysResearch\\CDTMOD1-5-04.exe
                                                        /run");
    p.waitFor();


    String [] commandLine2 = {"xcopy", curr_dir+".htm",
                                             curr_dir+".txt", "/y"};

    p = Runtime.getRuntime().exec(commandLine2);   //Create Source File
    p.waitFor();


/************HTML DOCUMENT MANAGEMENT***************************/

// Create an instance of the hypertext modeling language file
file = new File(curr_dir+".htm");

// New destination directory
dir = new File(curr_dir);

// Move the HTM file to the new directory
success = file.renameTo(new File(dir, file.getName()));

// Delete the old HTM file
success = (new File(curr_dir+".htm")).delete();


/***************************************************************/



/************HTML SOURCE CODE TEXT DOCUMENT MANAGEMENT**********/

// Create an instance of the TEXT file
file = new File(curr_dir+".txt");

// New destination directory
dir = new File(curr_dir);

// Move the TEXT file to the new directory
success = file.renameTo(new File(dir, file.getName()));

// Delete the old TEXT file
success = (new File(curr_dir+".txt")).delete();

/***************************************************************/


UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
SwingUtilities.updateComponentTreeUI(this);


} catch(Exception e) {

   System.out.println("ERROR 4****");
   System.out.println("This doesn't work due to: "
                            +e.getMessage()+"!!");
   e.printStackTrace();

}//end try catch

} //end VBSETUP constructor

}//end class VBSETUP :)
```

```
/*Class that completes the conversion process by getting the */
/*unicode value of each character to determine "significant*/
/*data" locations and maintain original document formatting*/
class GetUniVal //extends JFrame
{

    /*Function to perform parse operations of each record of each
      document*/
    /*One record is equivalent to one row of significant data*/
    public void PARSE_OP(String curr_dir, String filename) throws
                          IOException
    {

        char upperCaseChar, c;
        int i, count, dummy_int, unicodeValue, pg=1, new_page=0;
        String dummy_str="", final_record;
        String store_tag="";            /* Tag storage bin initiated */
        Integer ob_unicodeValue;

        /* Unicode storage bin initiated */
        /*Object allows anything to be stored in the list*/
        Vector<Object> v_uniVal = new Vector<Object>();

        // Create BufferedReader class instance
        BufferedReader FileInput = new BufferedReader(new
                          FileReader(curr_dir+"\\"+filename+".txt"));

        BufferedReader dummy = new BufferedReader(new
                          FileReader(curr_dir+"\\"+filename+".txt"));

        // Create BufferedWriter class instance
        PrintWriter output = new PrintWriter(new BufferedWriter(new
                 FileWriter(curr_dir+"\\"+"o"+filename+".txt")), true);

        dummy_str = dummy.readLine();


        try {


        while((dummy_str != null) && (dummy_str.indexOf("<div class=")
               == -1))
        {
            dummy_str = dummy.readLine();
            FileInput.readLine();

        }//end while loop


        dummy_str = HEADER_DEPLETION(FileInput, dummy, dummy_str);
        unicodeValue = FileInput.read();   //Reads single character
                                           //from file and returns an
                                           //integer value
        dummy_int = dummy.read();
        c = (char) unicodeValue;           //Casts the integer value
```

```
                                     //into a character
store_tag += c;
ob_unicodeValue = new Integer(unicodeValue);
v_uniVal.add(ob_unicodeValue);


count=1;                 /* Character Counter initialized to 1 */

while(dummy_int != -1)
{
     /* Scan HTML Document and extract useful data while
        generating a storage bin */
     while((unicodeValue != -1) && (!store_tag.endsWith("</p>")))
     {


     /* Decrement counter if the character read is a line feed */
      if((unicodeValue == 10) && (count == 1))
      {
          count--;
          store_tag = store_tag.substring(0, (store_tag.length()
                                                      - 1));
          v_uniVal.remove((v_uniVal.size() - 1));

      }//end if

        unicodeValue = FileInput.read();  //Reads single
                                          //character from file
                                          //and returns an
                                          //integer value
        dummy_int = dummy.read();
        c = (char) unicodeValue;    //Casts the integer value
                                    //into a character
      store_tag += c;               //Character Storage

      /*Unicodevalue Object*/
      ob_unicodeValue = new Integer(unicodeValue);
      v_uniVal.add(ob_unicodeValue);

      /* Continue reading if reached the end of line (eol) but
         not the end of paragraph (eop) */
      if(unicodeValue == 13)
      {
          store_tag = store_tag.substring(0, (store_tag.length()
                                                    - 1));
          v_uniVal.remove((v_uniVal.size() - 1));


          unicodeValue = FileInput.read();
          dummy_int = dummy.read();
          c = (char) unicodeValue;
          store_tag += c;
          ob_unicodeValue = new Integer(unicodeValue);
          v_uniVal.add(ob_unicodeValue);


      }//end if
```

```
            /* Increment the counter unless the unicode is a carriage
               return or line feed */
            if((unicodeValue != 13) && (unicodeValue != 10))
                  count++;                     // Tracking Device
            else
            {
                store_tag = store_tag.substring(0, (store_tag.length()
                                                         - 1));
                v_uniVal.remove((v_uniVal.size() - 1));

            }//end if-else statement

           }//end while


/**********************************************************************

     *FUNCTION N_CRITICAL_OR_CRITICAL PASSES OR GRABS LOADED DATA*
      **********************************************************************/

final_record = N_CRITICAL_OR_CRITICAL(v_uniVal, store_tag);


/* Create a new output file for each new page encountered */
if((unicodeValue != -1) && (final_record != null) &&
((final_record.indexOf("CORE") > -1) || (final_record.indexOf("COREBUSINESS")
> -1) || (final_record.indexOf("U.S. Census Bureau") > -1) ||
(final_record.indexOf("CENSUS ") > -1) || (final_record.indexOf("Census
Bureau") > -1)))
{

     pg++;   //Increment the page counter

     /* Create a new instance of the output writer (Create a New Output
        File) */
     output = new PrintWriter(new BufferedWriter(new
                  Filewriter(curr_dir+"\\"+"o"+pg+filename+".txt")),
                  true);

     FileInput.readLine();
     dummy_str = dummy.readLine();


     if(dummy_str != null)
     {


      /* A NEW PAGE of the Document has been reached */
      NEW_PG_HEADER_DEPLETION(FileInput, dummy, dummy_str);

      final_record = "";       //Re-Inititalize the final_record
                               //storage bin

      new_page = 1;            //Flag: Tells the application
                               //there is a new page encountered
      }//end inner if statement
```

```
}//end if statement



if((unicodeValue != -1) && (dummy_str != null) && (final_record != null) &&
((final_record.indexOf("CORE") == -1) ||
(final_record.indexOf("COREBUSINESS") == -1) || (final_record.indexOf("U.S.
Census Bureau") > -1) || (final_record.indexOf("CENSUS") > -1) ||
(final_record.indexOf("Census Bureau") > -1)))
{

     if(final_record.startsWith("&nbsp;"))
     {

        final_record = "";                  //&nbsp: Non-Breaking Space
        output.println(final_record);       //Print Critical Data to
                                            //Output File

     }
     else
      if((new_page != 1) && (!final_record.startsWith("&nbsp;")))
         output.println(final_record);


}//end if statement

new_page=0;                                  //Reset new page flag
count=1;                                     //Reset counter
store_tag="";                                //Reset tag storage
v_uniVal.removeAllElements();                //Reset uniValue storage

unicodeValue = FileInput.read();             //Input
dummy_int = dummy.read();
c = (char) unicodeValue;
store_tag += c;
ob_unicodeValue = new Integer(unicodeValue);
v_uniVal.add(ob_unicodeValue);


}//end outer while loop

}catch (Exception e) {
     System.out.println(e.getMessage());
     e.printStackTrace();

}//end try catch statement


}//end method PARSE_OP





/*Function that removes header and footer data from the first page of
  each document*/
```

```
public String HEADER_DEPLETION(BufferedReader FileInput, BufferedReader
dummy, String dummy_str) throws IOException
{

    /* Cut the Header Data */
    while((dummy_str != null) && (dummy_str.indexOf("<p
          class=MsoPlainText>Table ") == -1))
    {
         dummy_str = dummy.readLine();

         if((dummy_str != null) && (dummy_str.indexOf("<p
             class=MsoPlainText>Table ") >= 0))

              break;
         else
              FileInput.readLine();

    }//end while

    FileInput.readLine();

    /* Search File until significant data found! */
    while((dummy_str != null) &&
          (dummy_str.indexOf("<o:p>&nbsp;</o:p>") == -1))
    {
          dummy_str = dummy.readLine();
          if((dummy_str != null) &&
             (dummy_str.indexOf("<o:p>&nbsp;</o:p>") >= 0))
                 break;
          else
                 FileInput.readLine();

    }//end while


    FileInput.readLine();

    do /* BEGINNING OF DATA */
    {
         dummy_str = dummy.readLine();

         if((dummy_str != null) && (dummy_str.startsWith("<p")))
                  break;
            else
                  FileInput.readLine();

    }while((dummy_str != null) && (!dummy_str.startsWith("<p")));

    FileInput.readLine();  /* MY FINAL BUG FIX!!!!!!!!! */

    return dummy_str;

}//end HEADER_DEPLETION method


/*Function that removes header and footer data from additional pages of
  each document*/
```

```
public void NEW_PG_HEADER_DEPLETION(BufferedReader FileInput,
                                     BufferedReader dummy, String
                                     dummy_str) throws IOException
{

    while((dummy_str != null) && (dummy_str.indexOf("page-break") == -
          1))
    {
          dummy_str = dummy.readLine();

          if((dummy_str != null) && (dummy_str.indexOf("page-break") >=
             0))
                  break;
            else
                  FileInput.readLine();

    }//end while loop

    FileInput.readLine();

    dummy_str = HEADER_DEPLETION(FileInput, dummy, dummy_str);

}//end NEW_PG_HEADER_DEPLETION method


/*Function Identifies critical and non-critical data*/
public String N_CRITICAL_OR_CRITICAL(Vector v_uni, String s_tag)
{

    int i=0, v_uni_intVal;      //Declares and initializes variables
    String v_u="", fin_s_tag="";


    if((i <= v_uni.size()-1) && (i >= 0))
    {
        v_u = (v_uni.elementAt(i)).toString();
        v_uni_intVal = Integer.parseInt(v_u);


      while(i < v_uni.size())
      {
           switch(v_uni_intVal)
           {

                              /****PASS: NON-CRITICAL DATA****/
             case 60: while((v_uni_intVal != 62) && (i < v_uni.size()))
                          {
                        i++;
                                if(i <= v_uni.size()-1)
                        {
                            v_u = (v_uni.elementAt(i)).toString();
                                  v_uni_intVal = Integer.parseInt(v_u);

                        }//end if

                      }//end while loop
                      break;
```

```
                              /****GRAB: CRITICAL DATA****/
                 case 62: i++;
                      if(i >= v_uni.size())
                        v_uni_intVal = 999999;
                    else
                    {
                        v_u = (v_uni.elementAt(i)).toString();
                              v_uni_intVal = Integer.parseInt(v_u);

                    }//end else


                      while((v_uni_intVal != 60) && (i < v_uni.size()))
                      {
                         if(i+1 < s_tag.length())
                            fin_s_tag += s_tag.substring(i, i+1);

                         else
                            fin_s_tag += s_tag.substring(i);

                         i++;

                           if(i >= v_uni.size())
                             v_uni_intVal = 999999;
                         else
                         {
                             v_u = (v_uni.elementAt(i)).toString();
                             v_uni_intVal = Integer.parseInt(v_u);

                         }//end else


                     }//end while


/******DELIMETER INSERTION - July 11, 2006 -- Revised June 16, 2008***

       if((!fin_s_tag.equals("")) && (!fin_s_tag.endsWith("|")))
            fin_s_tag = fin_s_tag + "|";

***********************************************/

                    break;


                 default:

                     if((i >= 0) && (i < v_uni.size()))
                     {
                        i++;

                        if(i < v_uni.size())
                        {
                            v_u = (v_uni.elementAt(i)).toString();
                                  v_uni_intVal = Integer.parseInt(v_u);
                        }//end inner if statement
                    }//end outer if statement
                    break;
           }//end switch
       }//end while
     }//end outer if
     return fin_s_tag;
    }//end method N_CRITICAL_OR_CRITICAL

}//end class GetUniVal  :)
```

Appendix B

CDTMOD1-5-04 (VB6.0)

```
Option Explicit

Dim b_Excel_Closed As Boolean

Private Sub cmd_Close_Click()

    Unload Me

Exit_cmd_Close_Click:
    Exit Sub

    Resume Exit_cmd_Close_Click

End Sub


'PROCEDURE:     cmd_Import_Click()
'DESCRIPTION:   This method is used as the driver for
importing Word
'   files into HTML and formatting according to
requirements.
'PARAMETERS:    None
'RETURNS:       None
Private Sub cmd_Import_Click()
On Error GoTo Err_cmd_Import_Click

  'declare all variables
  Dim objWord
  Dim oDoc

  Dim fsoStream As TextStream
  Dim objFso
  Dim TextLine$, Filename$
  Dim FileHandle As Integer
  Dim colFiles
  Dim curFile
  Dim curFileName
  Dim folderToScan As String
  Dim folderToSave As String
  Dim folderToScanExists
  Dim folderToSaveExists
  Dim objFolderToScan

  ''''''''''''''''''
'Location to find the next file to convert to HTML
Filename$ = "C:\PennysResearch\currentDir.dat"

  ' Test if the file exists
  If Dir(Filename$) = "" Then Exit Sub

  FileHandle = FreeFile ' This is safer than assigning a
number

  Open Filename$ For Input As #FileHandle

  Do While Not EOF(FileHandle)         ' Loop until end of
file
   Line Input #FileHandle, TextLine$   ' Read line into
variable
  Loop

  MsgBox TextLine$
  Close #FileHandle

  'set some of the variables
  folderToScanExists = False
  folderToSaveExists = False
  Const wdSaveFormat = 8                  'for Filtered
HTML output

  '********************************************
  'change the following to fit your system
  folderToScan = TextLine$
  folderToSave = TextLine$
  '********************************************

  'Use FSO to see if the folders to read from
  'and write to both exist.
  'If they do, then set both flags to TRUE,
  'and proceed with the function
  'Otherwise, send out an error message
  Set objFso = CreateObject("Scripting.FileSystemObject")
  If objFso.FolderExists(folderToScan) Then
    folderToScanExists = True
  Else
    MsgBox "Folder to scan from does not exist!", 48, "File
System Error"
  End If
  If objFso.FolderExists(folderToSave) Then
    folderToSaveExists = True
  Else
    MsgBox "Folder to copy to does not exist!", 48, "File
System Error"
  End If
  If (folderToScanExists And folderToSaveExists) Then
    'get your folder to scan
    Set objFolderToScan = objFso.GetFolder(folderToScan)
    'put al the files under it in a collection
    Set colFiles = objFolderToScan.Files
    'create an instance of Word
  Set objWord = CreateObject("Word.Application")

  'Check to see if the Word file to read from exitsts
  'If it does exist and is a Word document, proceed with
the function
  'Otherwise, send out an error message

    If objWord Is Nothing Then
      MsgBox "Couldn't start Word.", 48, "Application Start
Error"
    Else
      'for each file
      For Each curFile In colFiles
        'only if the file is of type DOC
        If (objFso.GetExtensionName(curFile) = "doc") Then
          'get the filename without extension
          curFileName = curFile.Name
          curFileName = Mid(curFileName, 1,
InStrRev(curFileName, ".") - 1)
          'open the file inside Word
          objWord.Documents.Open
objFso.GetAbsolutePathName(curFile)
          'do all this in the background
          objWord.Visible = False

          'create a new document and save it as Filtered
HTML
          Set oDoc = objWord.ActiveDocument
          'oDoc.SaveAs folderToSave & curFileName & ".htm",
wdSaveFormat
          oDoc.SaveAs folderToSave & ".htm", wdSaveFormat
          oDoc.Close
          Set oDoc = Nothing
        End If
      Next
    End If
    'close Word
    objWord.Quit
    'set all objects and collections to nothing
    Set objWord = Nothing
    Set colFiles = Nothing
  Set objFolderToScan = Nothing
  End If

  Set objFso = Nothing

'Do the Import in the background
    cmd_Import.Visible = False
    cmd_Close.SetFocus
    Screen.MousePointer = vbDefault

Exit_cmd_Import_Click:
    Exit Sub

End Sub
```

Appendix C

ExcelConvert (VB6.0)

```
Option Explicit

Dim b_Excel_Closed As Boolean
Dim obj_Excel As Excel.Application
Dim xl_Workbook As Excel.Workbook


Private Sub cmd_Close_Click()


    Unload Me

Exit_cmd_Close_Click:
    Exit Sub

    Resume Exit_cmd_Close_Click

End Sub


'PROCEDURE:     cmd_Import_Click()
'DESCRIPTION:   This method is used as the driver for importing comma
delimited text
'   files into Excel and formatting according to requirements.
'PARAMETERS:    None
'RETURNS:       None
Private Sub cmd_Import_Click()
On Error GoTo Err_cmd_Import_Click

'  ********************************************
'    EXCEL
'  ********************************************
    Dim i As Integer
    Dim Y As Integer
    Dim Z As Integer
    Dim FileNames$()
    Dim s_Path As String
    Dim s_File_Name As String
    Dim List1 As ListBox


    Const OFN_ALLOWMULTISELECT = &H200&         'To allow selection of
more than one file in the Common Dialog Control
    Set obj_Excel = CreateObject("Excel.Application")
    b_Excel_Closed = False
    Set xl_Workbook = obj_Excel.Workbooks.Add
    xl_Workbook.Activate

    CommonDialog1.Filename = ""                 'Clear the file name
    CommonDialog1.DefaultExt = "txt"
```

```
  'Limit selection to Text files
  CommonDialog1.Filter = "Text Files (*.txt)|*.txt"

  'Allow selection of more than one file
  CommonDialog1.flags = OFN_ALLOWMULTISELECT
  CommonDialog1.MaxFileSize = 2048   'Set buffer to maximum string size
    CommonDialog1.Action = 1       'Standard Open file dialog
CommonDialog1.ShowSave

    If CommonDialog1.Filename <> "" Then

        'Store the file names selected by the user in the Filename
property as one long string
        'Each file name is separated by a space (32) character
         CommonDialog1.Filename = CommonDialog1.Filename & Chr(32)

         Screen.MousePointer = vbHourglass

        'Index the individual file names into FileNames$()
         Z = 1
         For i = 1 To Len(CommonDialog1.Filename)
            i = InStr(Z, CommonDialog1.Filename, Chr(32))
            If i = 0 Then Exit For
            ReDim Preserve FileNames(Y)
            FileNames(Y) = Mid(CommonDialog1.Filename, Z, i - Z)
            Z = i + 1
            Y = Y + 1
         Next


        'If there are more than 3 files chosen by the user, we must add
more worksheets to the workbook
        If Y > 4 Then
            For i = 4 To Y - 1
            xl_Workbook.Worksheets.Add.Move
After:=Worksheets(Worksheets.Count)
            Next i
        Else
        End If


        'Extract out the Path which is only stored in FileNames(0)
        s_Path = Left$(FileNames(0), InStrRev(FileNames(0), "\"))


        'Import the file(s) and then add the file name to the list
        'If only one file is chosen, the Path is not passed to the
import method since the path exists
        'in Filenames(0)
        'If multiple files are chosen, the Path must be passed to the
import method
        If Y = 1 Then
            Import_File "", FileNames(0), xl_Workbook, Y
            List1.AddItem Mid(FileNames(0), (InStrRev(FileNames(0),
"\") + 1))
        Else
```

```
            For i = 1 To Y - 1
                Import_File s_Path, FileNames(i), xl_Workbook, i
'Local method
                List1.AddItem FileNames(i)
                s_File_Name = FileNames(i)
            Next
        End If
        List1.AddItem ".....Import Complete"
        Label1.Caption = "Files Imported into Excel . . . .Complete"
        Me.Refresh

    Else
    End If

    xl_Workbook.Application.Visible = True
    xl_Workbook.Application.WindowState = xlMinimized

    'If Excel is still open, close it
    If b_Excel_Closed = False Then
        Set obj_Excel = Nothing
        Set xl_Workbook = Nothing
        b_Excel_Closed = True
    Else
    End If

    cmd_Import.Visible = False
    cmd_Close.SetFocus
    Screen.MousePointer = vbDefault

Exit_cmd_Import_Click:
    Exit Sub

Err_cmd_Import_Click:
'    List1.AddItem ".....Error in Import." '
'    List1.AddItem ".....See the error log C:\Text_Import_Errors.txt" '
    cmd_Import.Visible = False
    Screen.MousePointer = vbDefault
    cmd_Close.SetFocus
   If b_Excel_Closed = False Then
        obj_Excel.Quit
        Set obj_Excel = Nothing
        Set xl_Workbook = Nothing
      b_Excel_Closed = True
   Else
   End If

   Resume Exit_cmd_Import_Click

    'Set the focus on cell A1
    Range("A1").Select


End Sub
'PROCEDURE:     Import_File()
'DESCRIPTION:   This method imports comma delimited text files into
Excel.  Each individual file
```

```
'               is imported to an individual worksheet within the
workbook and formatted accordingly.
'PARAMETERS:      1)  Path - the location of the files
'                 2)  FileName - The name of the file to be imported
'                 3)  xl_Workbook - Reference to the Excel Workbook
where the file is being imported into
'                     4)  SheetCount - The sheet number in the
workbook, to import the information to
'RETURNS:       None
Private Sub Import_File(ByRef Path As String, ByRef Filename As String,
ByRef xl_Workbook As Workbook, ByRef SheetCount As Integer)

    Dim s_File_Name As String
    Dim s_Path As String
    Dim s_Title As String
    Dim s_Last_Row As String

    s_Path = Path
    s_File_Name = Filename

    'Select the sheet corresponding to the SheetCount
    Sheets("Sheet" & SheetCount).Select
    Sheets("Sheet" & SheetCount).Name = "Sheet" & SheetCount

    'Set sheet properties to import a comma delimited text file
    With ActiveSheet.QueryTables.Add(Connection:= _
        "TEXT;" & s_Path & s_File_Name & " ", Destination:=Range("A1"))
        .Name = s_File_Name
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .RefreshPeriod = 0
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = xlWindows
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited
        .TextFileTextQualifier = xlTextQualifierDoubleQuote
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimiter = False
        .TextFileSemicolonDelimiter = False
        .TextFileCommaDelimiter = False
        .TextFileSpaceDelimiter = False
        .TextFileOtherDelimiter = "|"
        .TextFileColumnDataTypes = Array(2, 2, 2, 2, 2, 2)
        .Refresh BackgroundQuery:=False
    End With

    Cells.Select
    Selection.Columns.AutoFit
    With Selection
        .HorizontalAlignment = xlRight
        .VerticalAlignment = xlBottom
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .ShrinkToFit = False
        .MergeCells = False
    End With

    Selection.Copy

    'Rename the sheet to the file name excluding the prefix "Output_"
and the extension ".txt"
    Sheets("Sheet" & SheetCount).Select
    Sheets("Sheet" & SheetCount).Name = Mid(s_File_Name, 1,
(Len(s_File_Name) - 4))

    'Set the focus on cell A1
    Range("A1").Select

Exit_Import_File:
    Exit Sub


    Resume Exit_Import_File

End Sub



Private Sub Form_Unload(Cancel As Integer)
    'If Excel is still open, close it
    If b_Excel_Closed = False Then
        Set obj_Excel = Nothing
        b_Excel_Closed = True
    Else
    End If
End Sub
```

Appendix D

P2X User’s Manual

1. Using the P2X Tool

P2X is an information management system that converts PDF publications into Microsoft Excel applications. Before using the P2X software tool, it is important to understand what it does. Reassuring that the user meets the minimum system requirements will help eliminate errors. While reviewing the system process flow chart, the user will hopefully gain a clearer understanding of how the product works. Also, making sure the proper documents are being used and following this manual accurately, will help verify that the final product meets the user's expectations. The P2X user's manual will guide the user on the product in a step-by-step process and consequently, minimize or eliminate the possibility of confusion and error.

1.1 Minimum Requirements

System Requirements:

- Microsoft® Windows XP, Windows 2000 SP2, Windows NT 4.0 SP6, Windows ME, Windows 98 SE
- 500 MHz Intel® Business class computer
- 128 Megabytes RAM (256 MB or more Recommended)
- 8 Gigabyte hard disk drive or greater for file storage
- Microsoft compatible Mouse and keyboard
- Broadband Internet Connection (for PDF file access)

Network Requirements:

- Windows 2000 (or above), Windows XP; Product was not tested on Windows Vista
- TCP/IP Network Protocol

Software Requirements:

- Any text editor
- Microsoft Excel 2003 or 2007
- Derek Noonberg's PDF-to-Text Version 3.0
- Visual Basic 6.0
- LaToyia Penny's CDTMOD1-5-04 VB Application Version 3.0
- LaToyia Penny’s ExcelConvert2008 VB Application Version 1.0
- Java (freeware) - Sun Microsystems Java™ Runtime Environment (JRE) 1.5.0 or better and Java Virtual Machine (JVM)

1.2 Process Flow Chart

The P2X Process Flow Chart shows a precise breakdown of the overall P2X process.

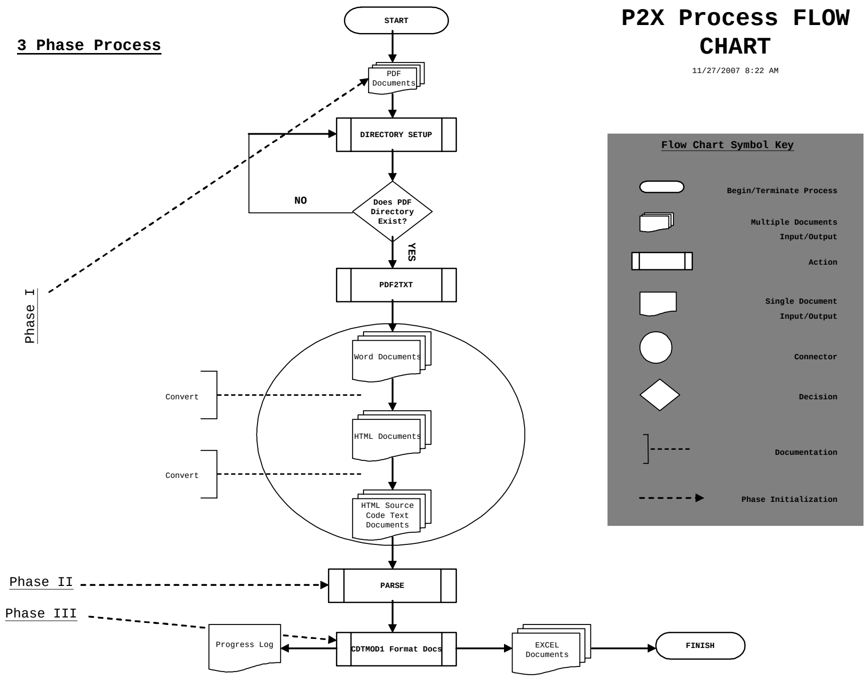


Figure 4: P2X Process Flow Chart

1.3 Getting Started

Please check the following before using the P2X software:

- Check that all minimum requirements are met.
- Create a new folder on your c:\ drive named **PennysResearch**.
- Make sure the P2X *executable* file (P2X.exe) and all other software required are stored on the c:\ drive in the **PennysResearch** folder.
- Create a new folder inside of the PennysResearch folder and name it **USCENSUSBUREAU**.

1.4 Portable Document Format (PDF)

A **Portable Document Format (PDF)** file is a self-contained cross-platform document. In simpler terms, it is a file that will look the same on the screen and in print, regardless of what kind of computer or printer someone is using and regardless of what software package was originally used to create it, or platform independent.

Although they contain the complete formatting of the original document, including fonts and images, PDF files are highly compressed, allowing complex information to be downloaded efficiently.

2. Downloading PDF Documents

2.1. Official Websites

Since PDF format allows the reliable reproduction of published material on many different platforms and only requires the use of free reader software accessible online at the official Adobe website: http://www.adobe.com, it is a way to conveniently and quickly disseminates the tables of "significant data" referred to throughout this research. All original testing publications involved in this thesis are in PDF format and they are available online at the official U.S. Census Bureau website: http://www.census.gov/prod/www/titles.html.

2.2. Valid Documents for P2X

Valid PDF documents for this research are defined as "the publications that [serve as] a resource guide to the programs and services of the Census Bureau." [26] More specifically, the documents that serve as the testing set for this research are composed from the following areas and their perspective sub-areas:

1. Agriculture
2. Business - Trade and Services
3. Construction and Housing
4. Foreign Trade

Note: These test sets were chosen merely based on the diversity of their subjects. The actual topics themselves bare no real significance for the scope of this thesis. The focus of these materials were their raw data and the formats they entail. Refer to Section 2.1 of this document for more detail on each publication.

2.3. File Storage

The following figures (Figures 6 and 7) reproduce the exact locations of where valid documents were stored during the verification process of the quality assurance phase of building the P2X tool. Use these figures to verify the original PDF files are properly stored for easy accessibility ***before*** running P2X.

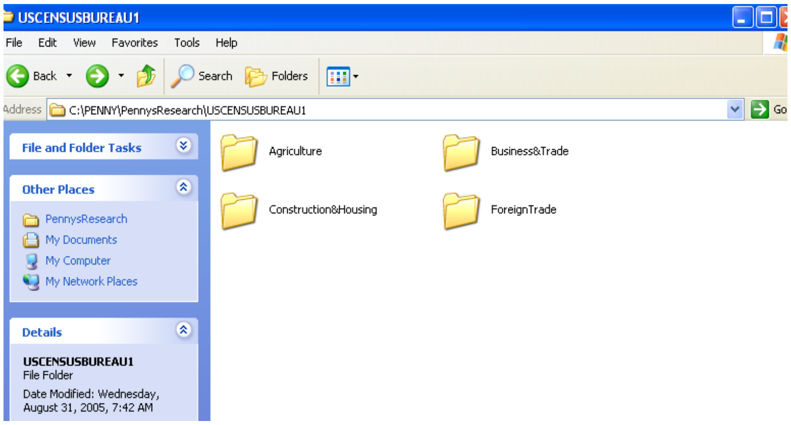


Figure 5: Parent directories of stored PDF files.

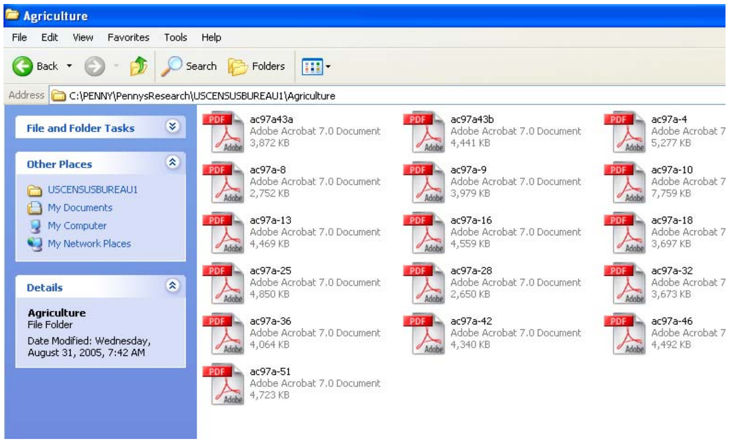
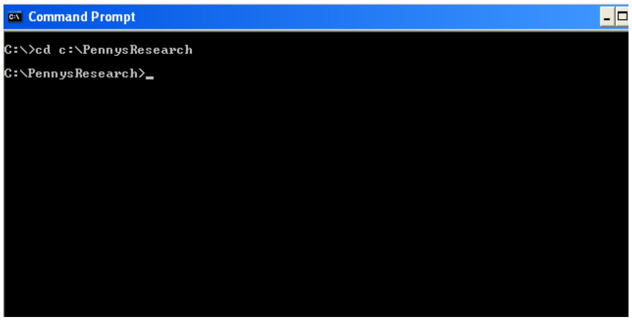

Agriculture
File Edit View Favorites Tools Help
Back Search Folders
Address C:\PENNY\PennysResearch\USCENSUSBUREAU1\Agriculture
File and Folder Tasks
Other Places
USCENSUSBUREAU1
My Documents
My Computer
My Network Places
Details
Agriculture
File Folder
Date Modified: Wednesday, August 31, 2005, 7:42 AM
ac97a43a Adobe Acrobat 7.0 Document 3,872 KB
ac97a43b Adobe Acrobat 7.0 Document 4,441 KB
ac97a-4 Adobe Acrobat 7 5,277 KB
ac97a-8 Adobe Acrobat 7.0 Document 2,752 KB
ac97a-9 Adobe Acrobat 7.0 Document 3,979 KB
ac97a-10 Adobe Acrobat 7 7,759 KB
ac97a-13 Adobe Acrobat 7.0 Document 4,469 KB
ac97a-16 Adobe Acrobat 7.0 Document 4,559 KB
ac97a-18 Adobe Acrobat 7 3,697 KB
ac97a-25 Adobe Acrobat 7.0 Document 4,850 KB
ac97a-28 Adobe Acrobat 7.0 Document 2,650 KB
ac97a-32 Adobe Acrobat 7 3,673 KB
ac97a-36 Adobe Acrobat 7.0 Document 4,064 KB
ac97a-42 Adobe Acrobat 7.0 Document 4,340 KB
ac97a-46 Adobe Acrobat 7 4,492 KB
ac97a-51 Adobe Acrobat 7.0 Document 4,723 KB

Figure 6: Example of initially stored Agriculture PDF files before running P2X.

3. Running P2X

Step 1: From the DOS command prompt, change the directory to the location of the P2X executable file (i.e. cd c:\PennysResearch; refer to Figure 8).



```
C:\>cd c:\PennysResearch

C:\PennysResearch>_
```

Figure 7: Change the directory to the location of the P2X program.

Step 2: Compile the program by typing: **javac DirectorySetup.java**

Step 3: Run the program by typing: **java DirectorySetup**

Note: A progress bar will appear on the top left hand corner of the user's computer screen along with two user-friendly pop-up screens for verification of document processing (Refer to Figure 9).

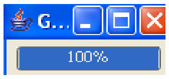
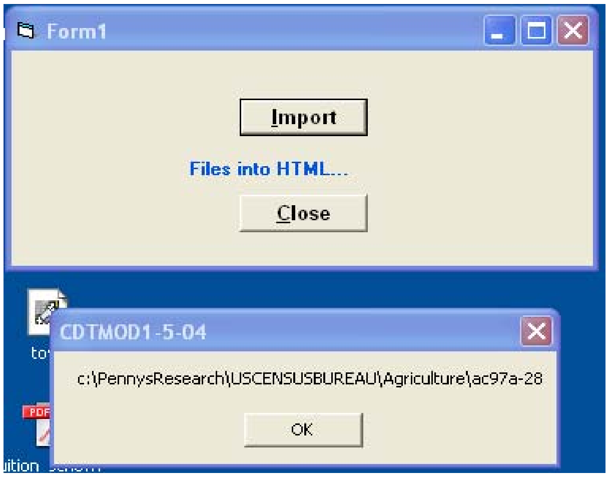


Figure 8: Progress Bar



c:\PennysResearch\USCENSUSBUREAU\Agriculture\ac97a-28

Figure 9: User friendly pop-up screens to assure the user of document progress.

Step 3.1 - P2X Tool FORM1 Press **Import** (Refer to Figure 10).
Step 3.2 - CDTMOD1-5-04 states the pathname of the file to be processed. If correct, user selects **ok** (Refer to Figure 10).

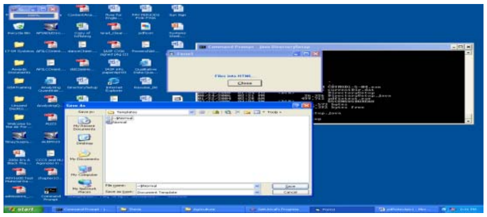


Figure 10: Form 1

Step 3.3 - When the document is complete, the Form 1 box will flash and the Import button will disappear. Select the close button (Refer to Figure 11).

Note: Select the X to close the *Progress Bar* (Figure 9) for that document (Note: if the *Progress Bar* is not closed it will not effect the next document, however, the *Progress Bars* will accumulate for each document and will eventually have to be closed after the program is finished processing completely).

Step 4: Repeat Steps 3.1 through 3.3 for every document in the directory tree.

Note: Be sure that Normal.dot in the Word Templates folder exists and is not being used by another user! In a network environment, the file is in the C:\Documents and Settings\user's name\Application Data\Microsoft\Word directory.

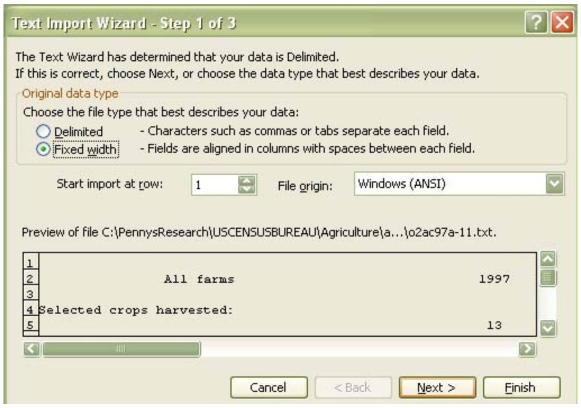
Step 5: Open the PennysResearch directory to view the *Progress Log* and verify the correct documents were processed.

Step 6: Open the USCENSUSBUREAU directory to review all processed documents.

4. Arranging the Final Excel Documents

The final Excel spreadsheet files generated by P2X will require minor adjustments to the imported data to eliminate non-breaking spaces and for each user's personal preference. The guidelines and visual samples below will help support the user in developing readable documents in Microsoft Excel:

Step 1: Open the final Excel document. Open the Text Import Wizard by pointing the mouse to the menu Toolbar and selecting Data -> Import Data. The following window will appear:



Text Import Wizard - Step 1 of 3

The Text Wizard has determined that your data is Delimited.
If this is correct, choose Next, or choose the data type that best describes your data.

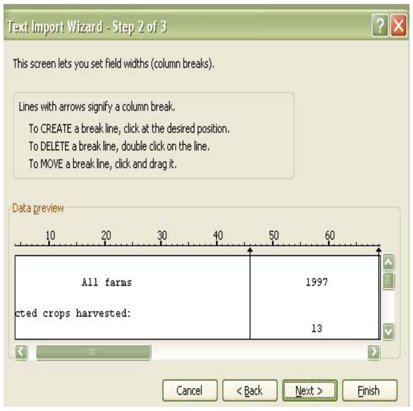
Original data type
Choose the file type that best describes your data:
○ Delimited - Characters such as commas or tabs separate each field.
● Fixed width - Fields are aligned in columns with spaces between each field.

Start import at row: 1 File origin: Windows (ANSI)

Preview of file C:\PennysResearch\USCENSUSBUREAU\Agriculture\a...\o2ac97a-11.txt.

```
1
2              All farms                             1997
3
4 Selected crops harvested:
5                                                    13
```

Step 2: Choose Fixed width as the Original data type. Verify all other options are selected as shown above. Click Next >.



Text Import Wizard - Step 2 of 3

This screen lets you set field widths (column breaks).

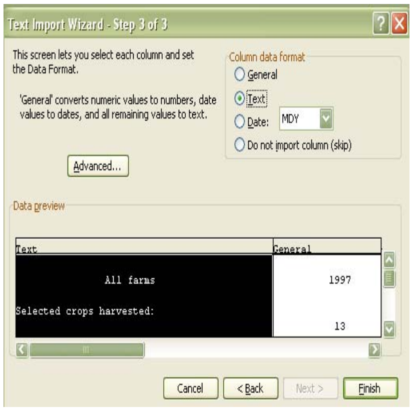
Lines with arrows signify a column break.
To CREATE a break line, click at the desired position.
To DELETE a break line, double click on the line.
To MOVE a break line, click and drag it.

Data preview

```
         10        20        30        40        50        60
          All farms                        1997

cted crops harvested:
                                           13
```

Step 3: Set fields widths to the user's satisfaction. Be sure to scroll down to be sure to capture all data into the correct column areas. Click Next >.



Text Import Wizard - Step 3 of 3

This screen lets you select each column and set the Data Format.

'General' converts numeric values to numbers, date values to dates, and all remaining values to text.

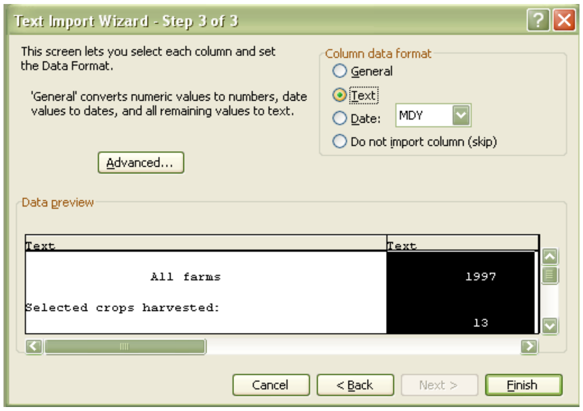
Column data format

○ General
● Text
○ Date: MDY
○ Do not import column (skip)

Data preview

Text	General
All farms	1997
Selected crops harvested:	13

Step 4: Click on each column and choose Text as the Column data format as shown above and below.



Text Import Wizard - Step 3 of 3

This screen lets you select each column and set the Data Format.

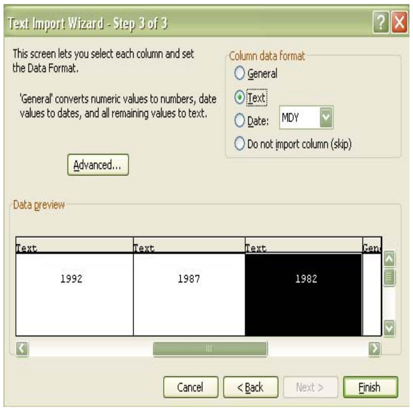
'General' converts numeric values to numbers, date values to dates, and all remaining values to text.

Column data format

○ General
● Text
○ Date: MDY
○ Do not import column (skip)

Data preview

Text	Text
All farms	1997
Selected crops harvested:	13



Text Import Wizard - Step 3 of 3

This screen lets you select each column and set the Data Format.

'General' converts numeric values to numbers, date values to dates, and all remaining values to text.

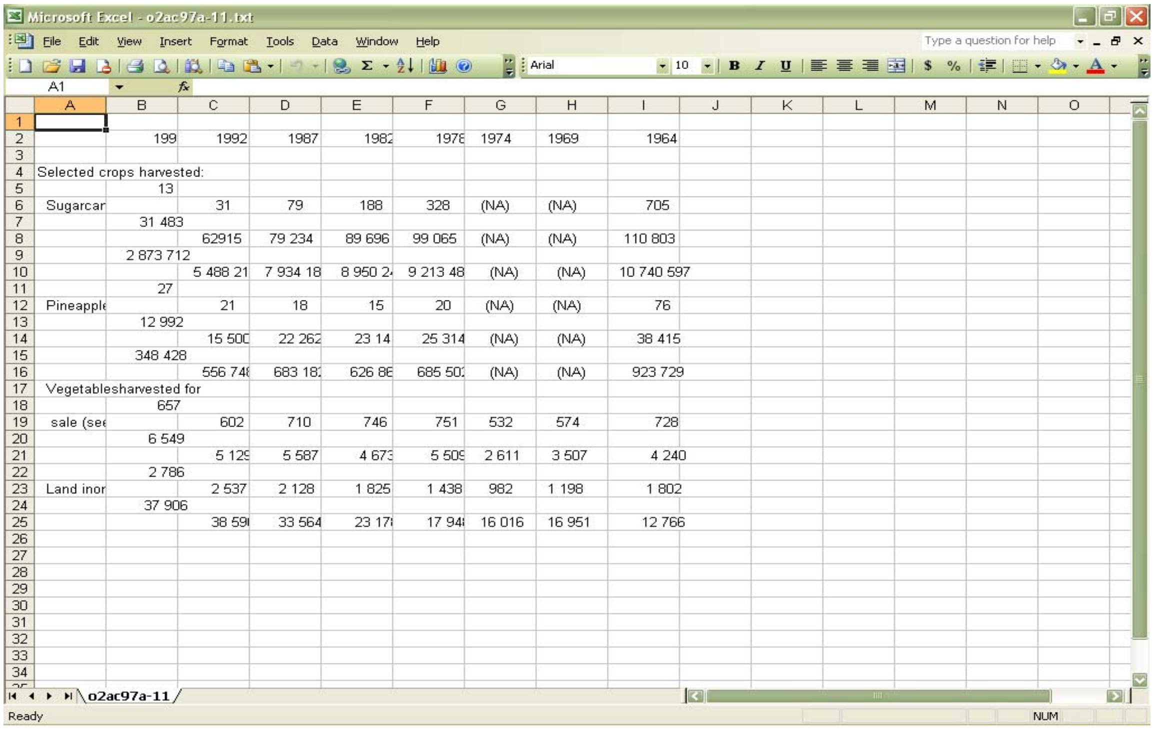
Column data format

○ General
● Text
○ Date: MDY
○ Do not import column (skip)

Data preview

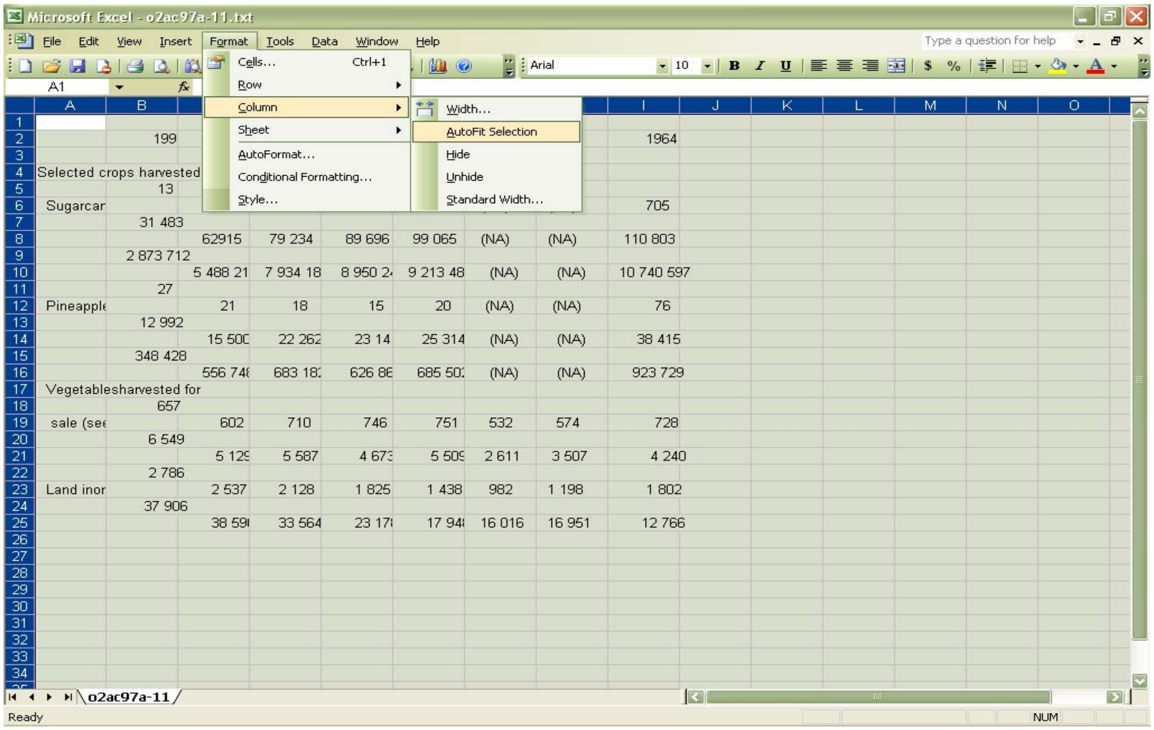
Text	Text	Text	Gen
1992	1987	1982	

Click Finish.


Microsoft Excel - o2ac97a-11.txt
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
10
A1
o2ac97a-11
Ready
NUM

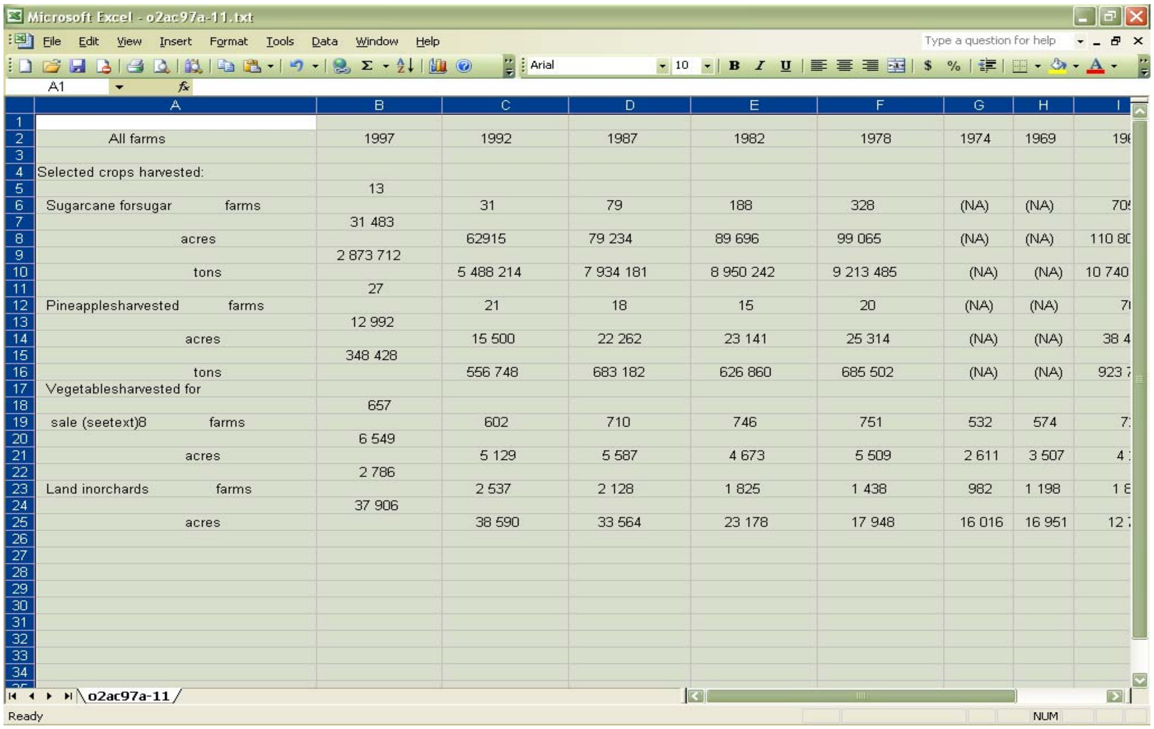
	A	B	C	D	E	F	G	H	I
1									
2		199	1992	1987	1982	1978	1974	1969	1964
3									
4	Selected crops harvested:								
5		13							
6	Sugarcar		31	79	188	328	(NA)	(NA)	705
7		31 483							
8			62915	79 234	89 696	99 065	(NA)	(NA)	110 803
9		2 873 712							
10			5 488 21	7 934 18	8 950 2	9 213 48	(NA)	(NA)	10 740 597
11		27							
12	Pineapple		21	18	15	20	(NA)	(NA)	76
13		12 992							
14			15 500	22 262	23 14	25 314	(NA)	(NA)	38 415
15		348 428							
16			556 74	683 18	626 86	685 50	(NA)	(NA)	923 729
17	Vegetablesharvested for								
18		657							
19	sale (see		602	710	746	751	532	574	728
20		6 549							
21			5 129	5 587	4 673	5 509	2 611	3 507	4 240
22		2 786							
23	Land inor		2 537	2 128	1 825	1 438	982	1 198	1 802
24		37 906							
25			38 59	33 564	23 17	17 94	16 016	16 951	12 766

The document will look similar to the one shown above.



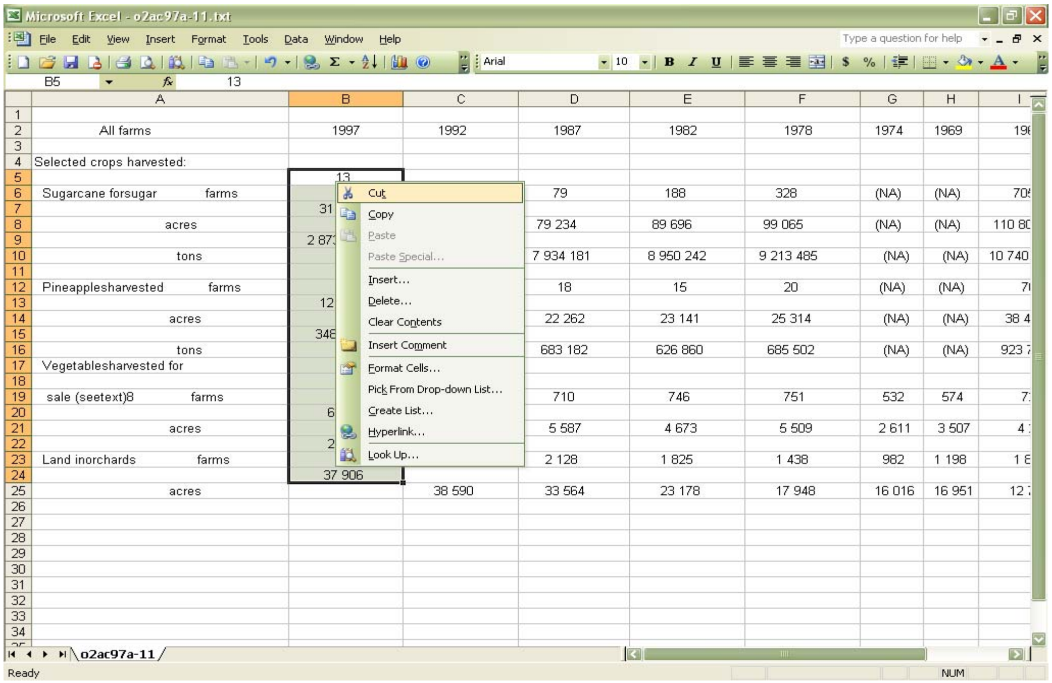
	A	B	C	D	E	F	G	H	I
1									
2		199							1964
3									
4	Selected crops harvested								
5		13							
6	Sugarcar								705
7		31 483							
8			62915	79 234	89 696	99 065	(NA)	(NA)	110 803
9		2 873 712							
10			5 488 21	7 934 18	8 950 2	9 213 48	(NA)	(NA)	10 740 597
11		27							
12	Pineapple		21	18	15	20	(NA)	(NA)	76
13		12 992							
14			15 500	22 262	23 14	25 314	(NA)	(NA)	38 415
15		348 428							
16			556 74	683 18	626 86	685 50	(NA)	(NA)	923 729
17	Vegetablesharvested for								
18		657							
19	sale (see		602	710	746	751	532	574	728
20		6 549							
21			5 129	5 587	4 673	5 509	2 611	3 507	4 240
22		2 786							
23	Land inor		2 537	2 128	1 825	1 438	982	1 198	1 802
24		37 906							
25			38 59	33 564	23 17	17 94	16 016	16 951	12 766

Step 5: Select all data and choose from the menu Format -> Column -> Auto Fit Selection to fit each column to the length of the longest data entry.


Microsoft Excel - o2ac97a-11.txt
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
10
A1
o2ac97a-11
Ready
NUM

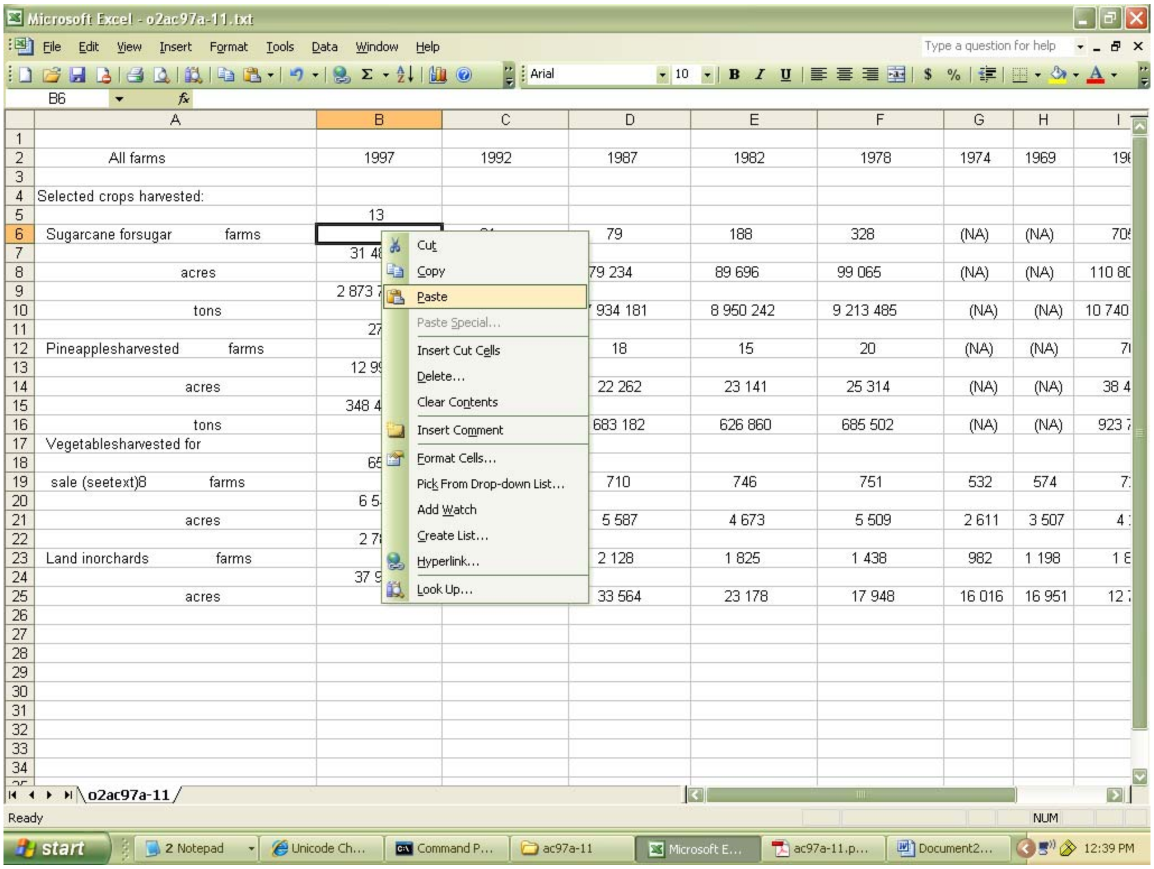
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	19
3									
4	Selected crops harvested:								
5		13							
6	Sugarcane forsugar farms		31	79	188	328	(NA)	(NA)	70
7		31 483							
8	acres		62915	79 234	89 696	99 065	(NA)	(NA)	110 80
9		2 873 712							
10	tons		5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA)	10 740
11		27							
12	Pineappleshavested farms		21	18	15	20	(NA)	(NA)	7
13		12 992							
14	acres		15 500	22 262	23 141	25 314	(NA)	(NA)	38 4
15		348 428							
16	tons		556 748	683 182	626 860	685 502	(NA)	(NA)	923 7
17	Vegetablesharvested for								
18		657							
19	sale (seetext)8 farms		602	710	746	751	532	574	7
20		6 549							
21	acres		5 129	5 587	4 673	5 509	2 611	3 507	4
22		2 786							
23	Land inorchards farms		2 537	2 128	1 825	1 438	982	1 198	1 8
24		37 906							
25	acres		38 590	33 564	23 178	17 948	16 016	16 951	12 7
26									
27									
28									
29									
30									
31									
32									
33									
34									

The document will look similar to the one shown above.



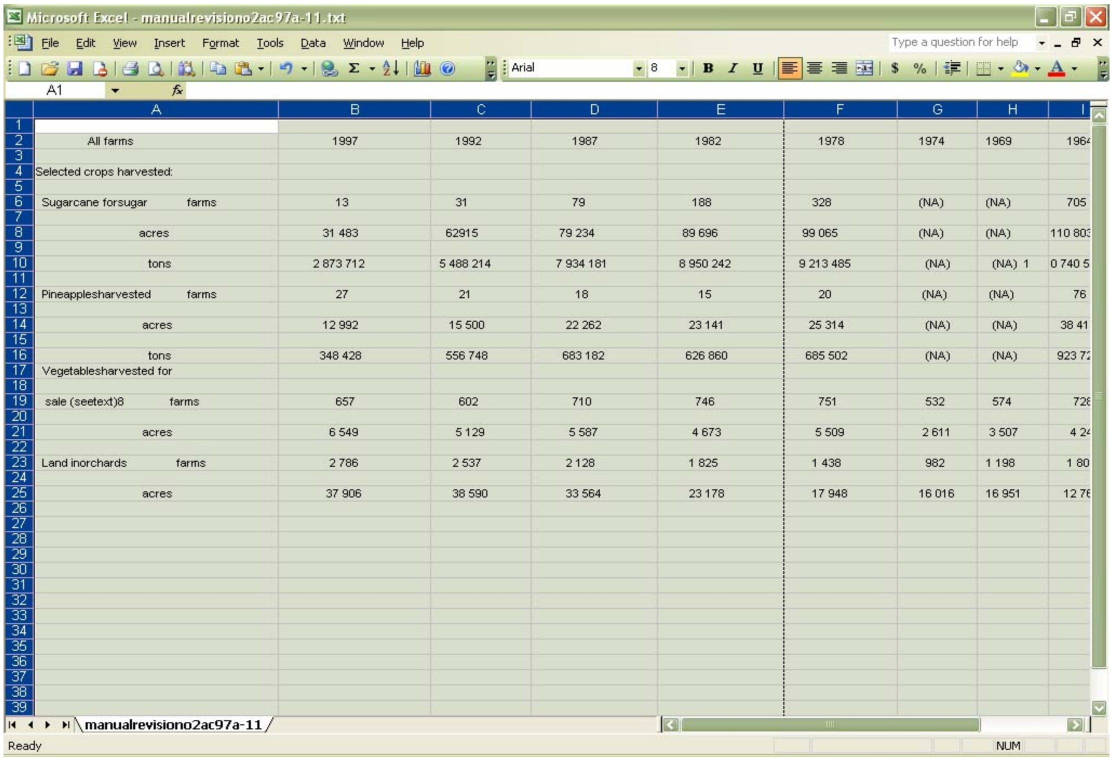
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	19
3									
4	Selected crops harvested:								
5		13							
6	Sugarcane forsugar farms			79	188	328	(NA)	(NA)	70
7		31							
8	acres			79 234	89 696	99 065	(NA)	(NA)	110 80
9		2 87							
10	tons			7 934 181	8 950 242	9 213 485	(NA)	(NA)	10 740
11									
12	Pineappleshavested farms			18	15	20	(NA)	(NA)	7
13		12							
14	acres			22 262	23 141	25 314	(NA)	(NA)	38 4
15		348							
16	tons			683 182	626 860	685 502	(NA)	(NA)	923 7
17	Vegetablesharvested for								
18									
19	sale (seetext)8 farms			710	746	751	532	574	7
20		6							
21	acres			5 587	4 673	5 509	2 611	3 507	4
22		2							
23	Land inorchards farms			2 128	1 825	1 438	982	1 198	1 8
24		37 906							
25	acres		38 590	33 564	23 178	17 948	16 016	16 951	12 7
26									
27									
28									
29									
30									
31									
32									
33									
34									

Step 6: In reviewing the sample above, it is evident that a column of data was inexplicably placed into the spreadsheet during the transfer from a text file to Excel. Cut all data from B6 to B24.



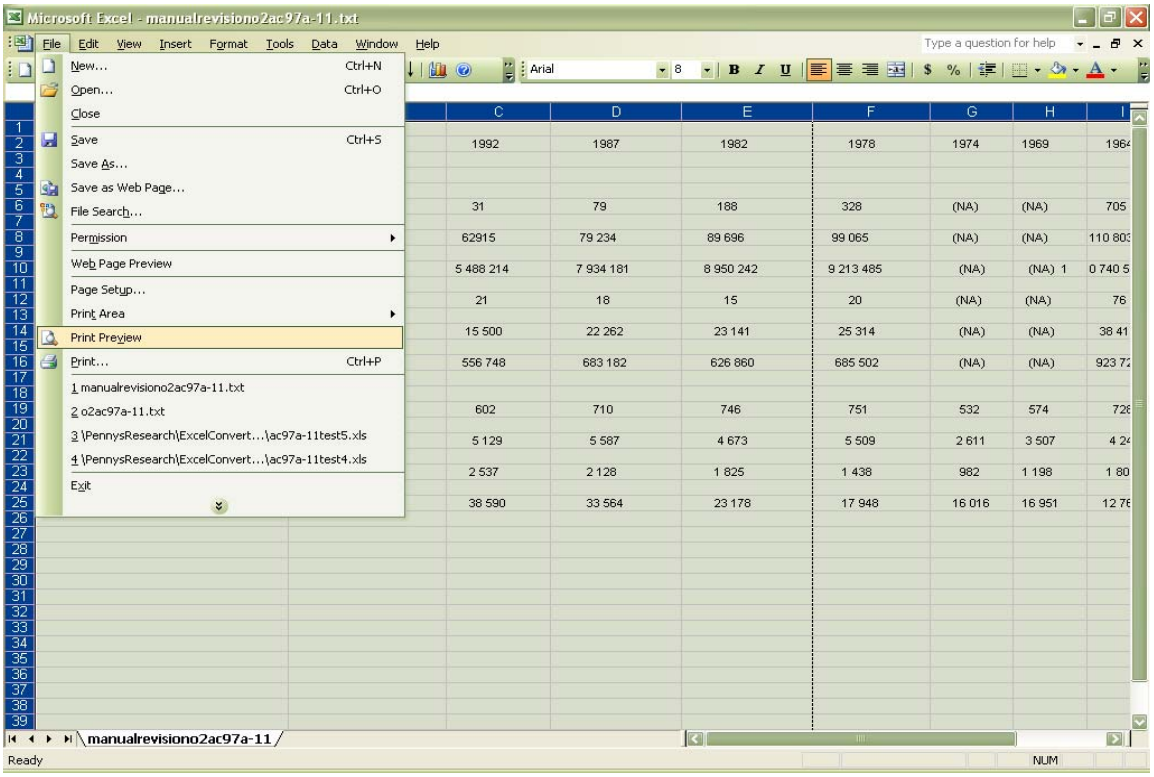
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	19
3									
4	Selected crops harvested:								
5		13							
6	Sugarcane forsugar farms			79	188	328	(NA)	(NA)	70
7		31 4							
8	acres			79 234	89 696	99 065	(NA)	(NA)	110 80
9		2 873 7							
10	tons			934 181	8 950 242	9 213 485	(NA)	(NA)	10 740
11		27							
12	Pineapplesharvested farms			18	15	20	(NA)	(NA)	7
13		12 9							
14	acres			22 262	23 141	25 314	(NA)	(NA)	38 4
15		348 4							
16	tons			683 182	626 860	685 502	(NA)	(NA)	923
17	Vegetablesharvested for								
18		65							
19	sale (seetext)8 farms			710	746	751	532	574	7
20		6 5							
21	acres			5 587	4 673	5 509	2 611	3 507	4
22		2 7							
23	Land inorchards farms			2 128	1 825	1 438	982	1 198	1
24		37 9							
25	acres			33 564	23 178	17 948	16 016	16 951	12

Step 7: Paste all data from B6 to B24 into cell B7 and all data in column B will move down by one cell


Microsoft Excel - manualrevisiono2ac97a-11.txt
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial 8
A1
manualrevisiono2ac97a-11
Ready
NUM

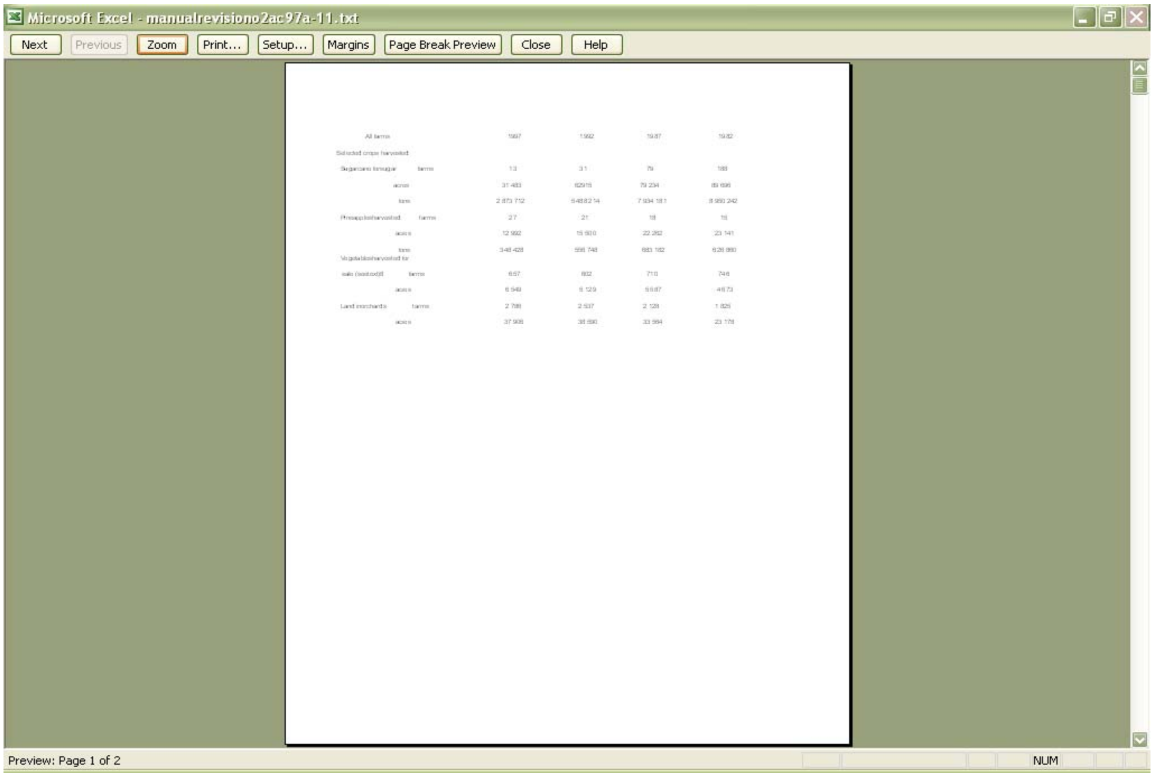
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	196
3									
4	Selected crops harvested:								
5									
6	Sugarcane forsugar farms	13	31	79	188	328	(NA)	(NA)	705
7									
8	acres	31 483	62915	79 234	89 696	99 065	(NA)	(NA)	110 80
9									
10	tons	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
11									
12	Pineapplesharvested farms	27	21	18	15	20	(NA)	(NA)	76
13									
14	acres	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
15									
16	tons	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 7
17	Vegetablesharvested for								
18									
19	sale (seetext)8 farms	657	602	710	746	751	532	574	72
20									
21	acres	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 2
22									
23	Land inorchards farms	2 786	2 537	2 128	1 825	1 438	982	1 198	1 80
24									
25	acres	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 7

The document should now look similar to the one shown above.

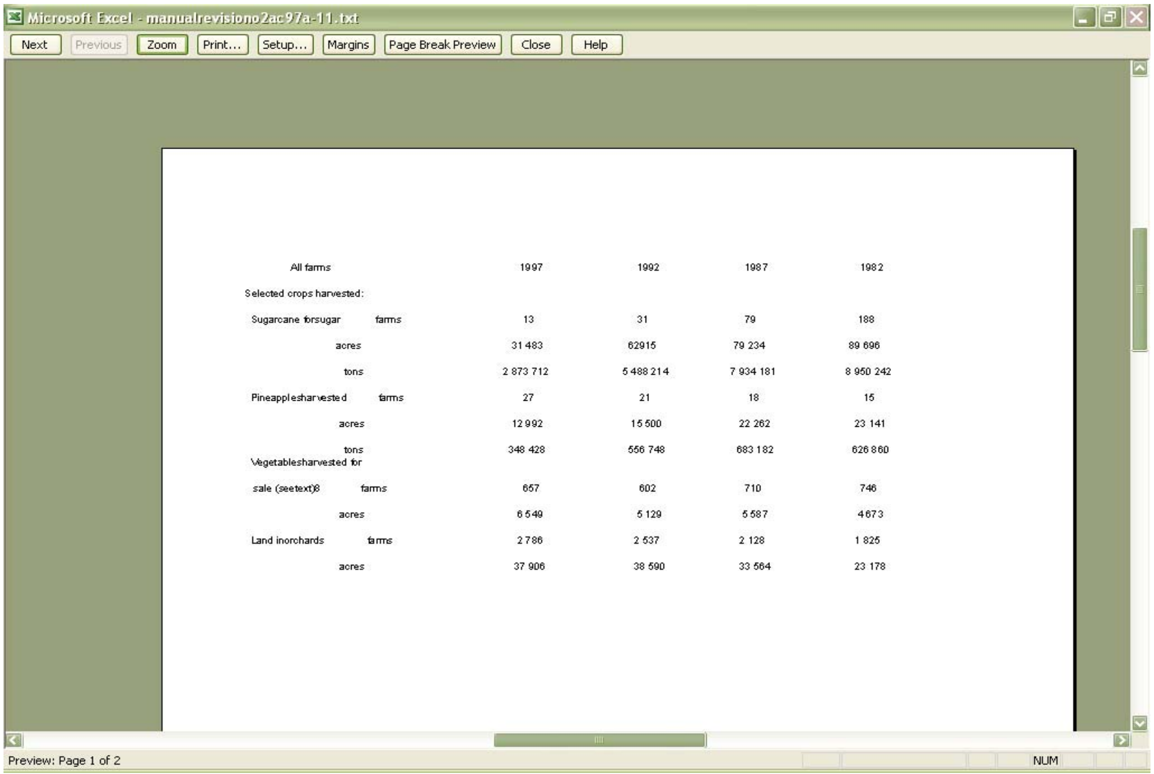


	C	D	E	F	G	H	I
2	1992	1987	1982	1978	1974	1969	1964
6	31	79	188	328	(NA)	(NA)	705
8	62915	79 234	89 696	99 065	(NA)	(NA)	110 803
10	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
12	21	18	15	20	(NA)	(NA)	76
14	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
16	556 748	683 182	626 860	685 502	(NA)	(NA)	923 72
19	602	710	746	751	532	574	728
21	5 129	5 587	4 673	5 509	2 611	3 507	4 24
23	2 537	2 128	1 825	1 438	982	1 198	1 80
25	38 590	33 564	23 178	17 948	16 016	16 951	12 78

Step 8: From the File Menu, choose Print Preview.

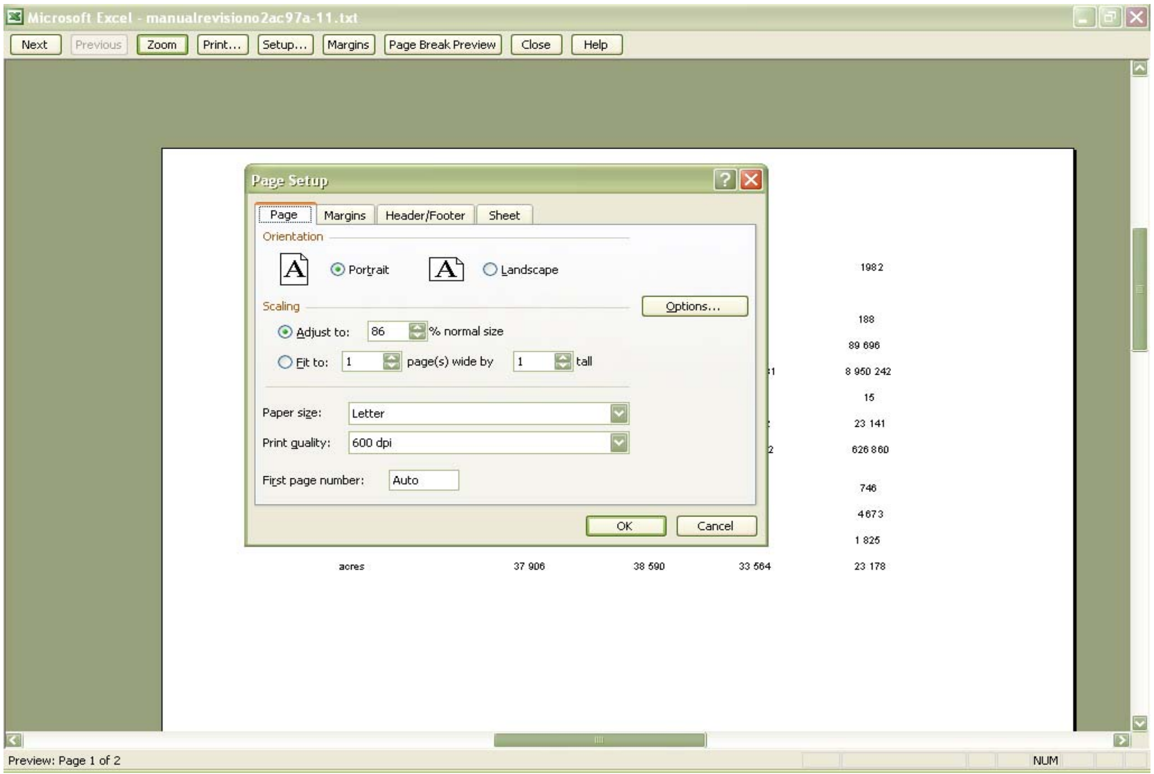


The document will look similar to the picture shown above in Print Preview view.



All farms		1997	1992	1987	1982
Selected crops harvested:					
Sugarcane forsugar	farms	13	31	79	188
	acres	31 483	62915	79 234	89 696
	tons	2 873 712	5 488 214	7 934 181	8 950 242
Pineapplesharvested	farms	27	21	18	15
	acres	12 992	15 500	22 262	23 141
	tons	348 428	556 748	683 182	626 860
Vegetablesharvested for					
sale (seetext)8	farms	657	602	710	746
	acres	6 549	5 129	5 587	4673
Land inorchards	farms	2 786	2 537	2 128	1 825
	acres	37 906	38 590	33 564	23 178

Step 9: Zoom into the document by pushing the Zoom button with the mouse to get a better view of the data.



Orientation

● Portrait ○ Landscape

Scaling

● Adjust to: 86 % normal size

○ Fit to: 1 page(s) wide by 1 tall

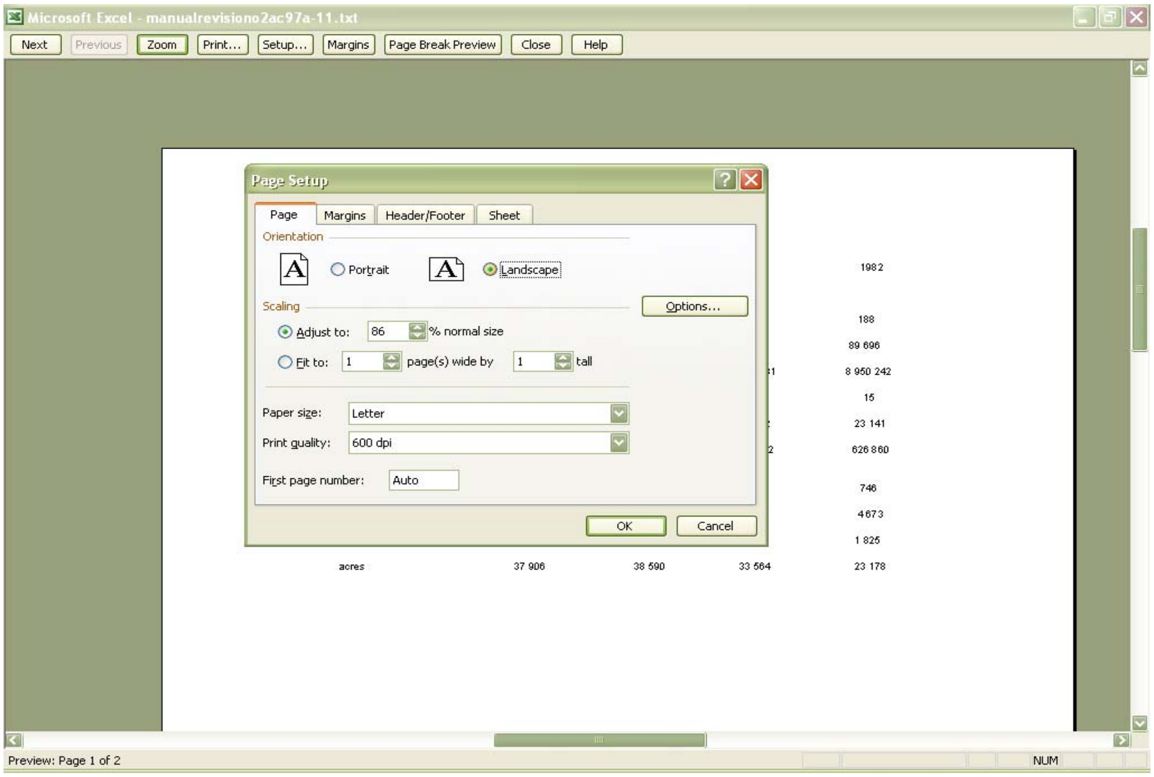
Paper size: Letter

Print quality: 600 dpi

First page number: Auto

				1982
				188
				89 696
				8 950 242
				15
				23 141
				626 860
				746
				4673
				1 825
acres	37 906	38 590	33 564	23 178

Step 10: Select the Setup button. The Page Setup window will appear as shown above.



Orientation

○ Portrait ● Landscape

Scaling

● Adjust to: 86 % normal size

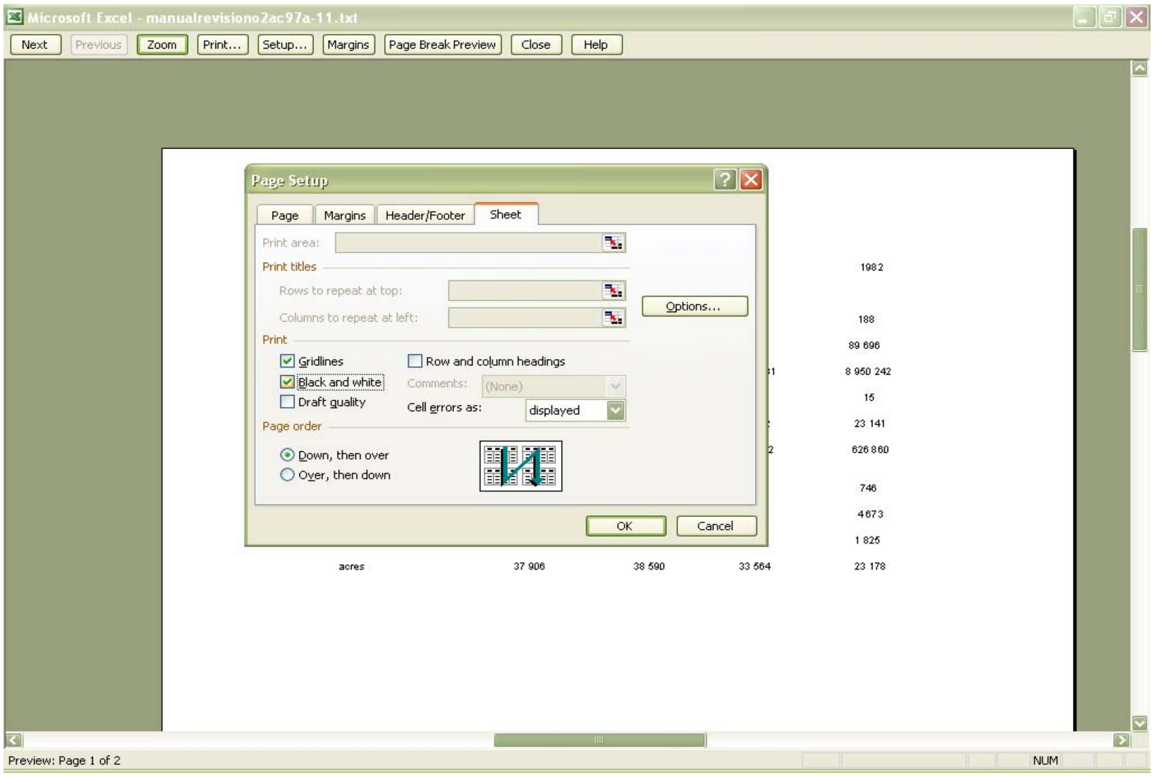
○ Fit to: 1 page(s) wide by 1 tall

Paper size: Letter

Print quality: 600 dpi

First page number: Auto

Step 11: Choose Landscape Orientation leaving all other options as shown above.



Print area:

Print titles

Rows to repeat at top:

Columns to repeat at left:

Print

☑ Gridlines

☑ Black and white

☐ Draft quality

☐ Row and column headings

Comments: (None)

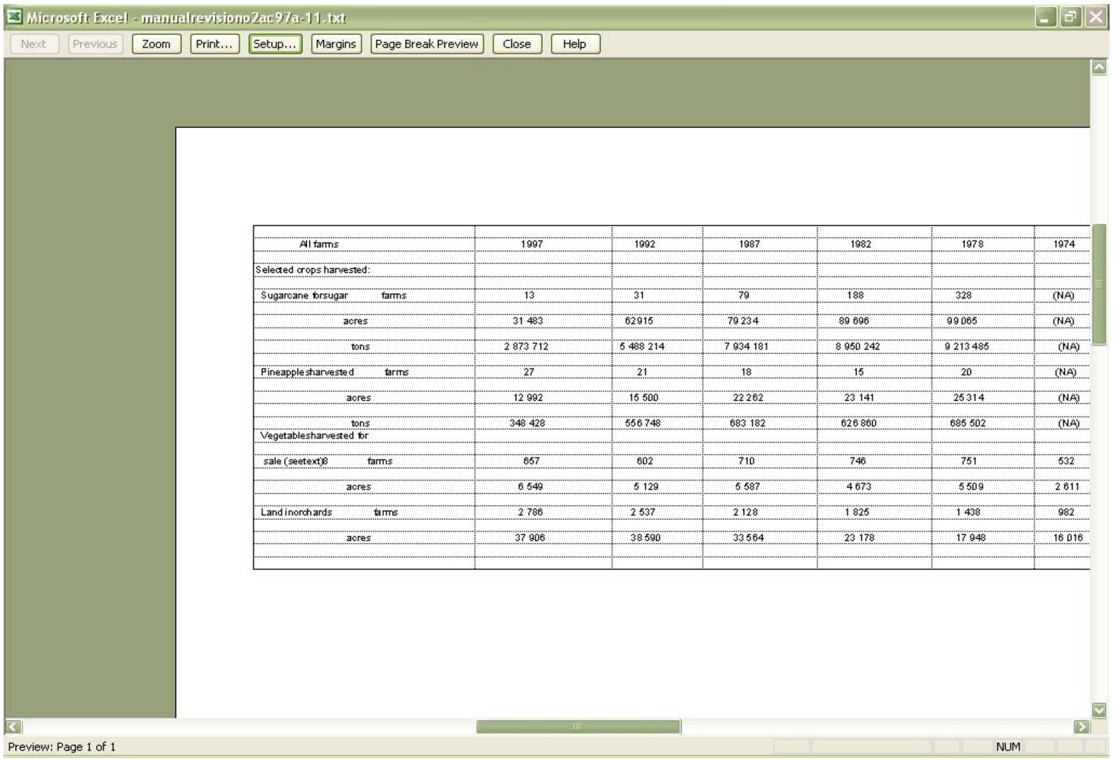
Cell errors as: displayed

Page order

● Down, then over

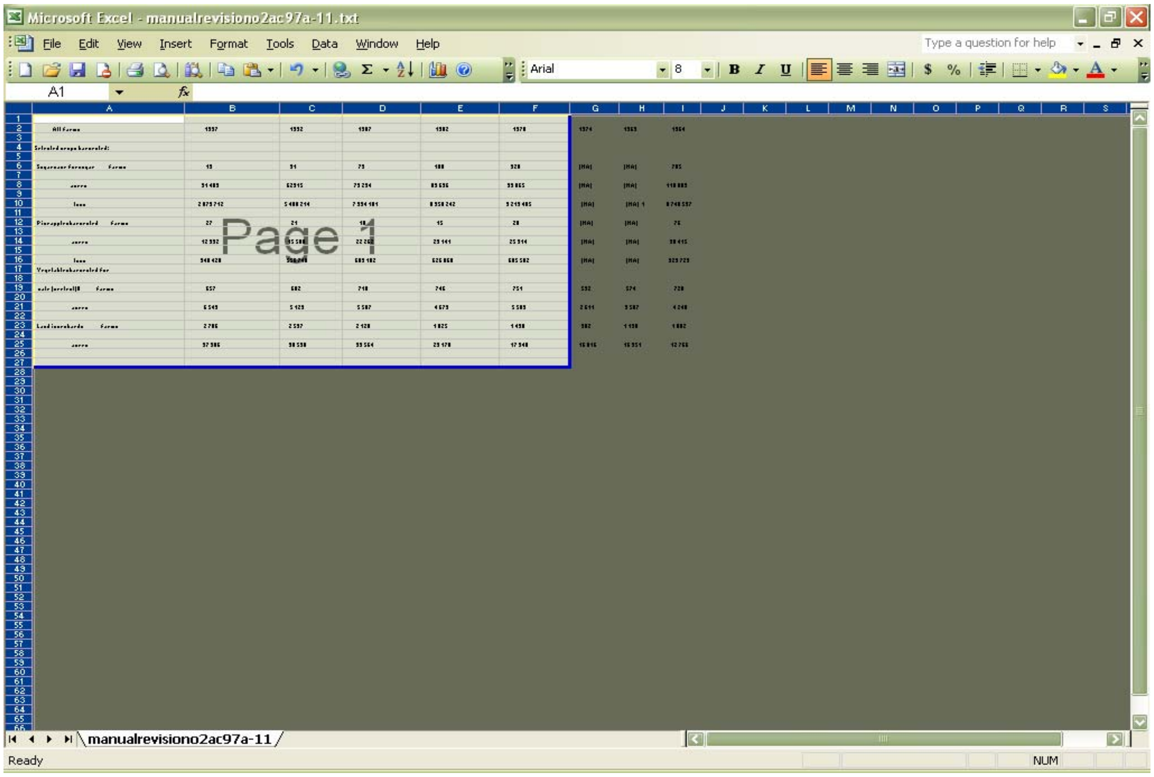
○ Over, then down

Step 12: Select the Sheet tab in the Page Setup window and select as Print options Gridlines and Black and white. Then press OK. The figure above illustrates.

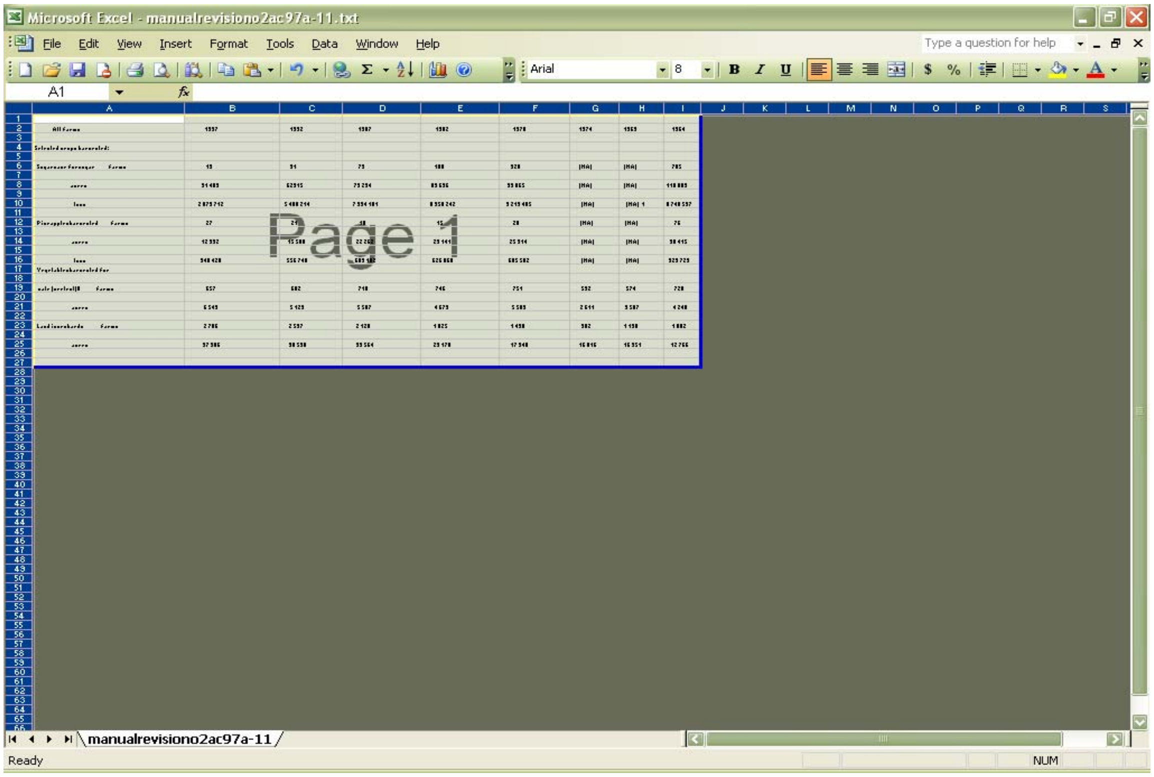


All farms	1997	1992	1987	1982	1978	1974
Selected crops harvested:						
Sugarcane forsugar farms	13	31	79	188	328	(NA)
acres	31 483	62915	79 234	89 696	99 065	(NA)
tons	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)
Pineapplesharvested farms	27	21	18	15	20	(NA)
acres	12 992	15 500	22 262	23 141	25 314	(NA)
tons	348 428	556 748	683 182	626 860	685 502	(NA)
Vegetablesharvested for						
sale (seetext)8 farms	657	602	710	746	751	532
acres	6 549	5 129	5 587	4 673	5 509	2 611
Land inorchards farms	2 786	2 537	2 128	1 825	1 438	982
acres	37 906	38 590	33 564	23 178	17 948	16 016

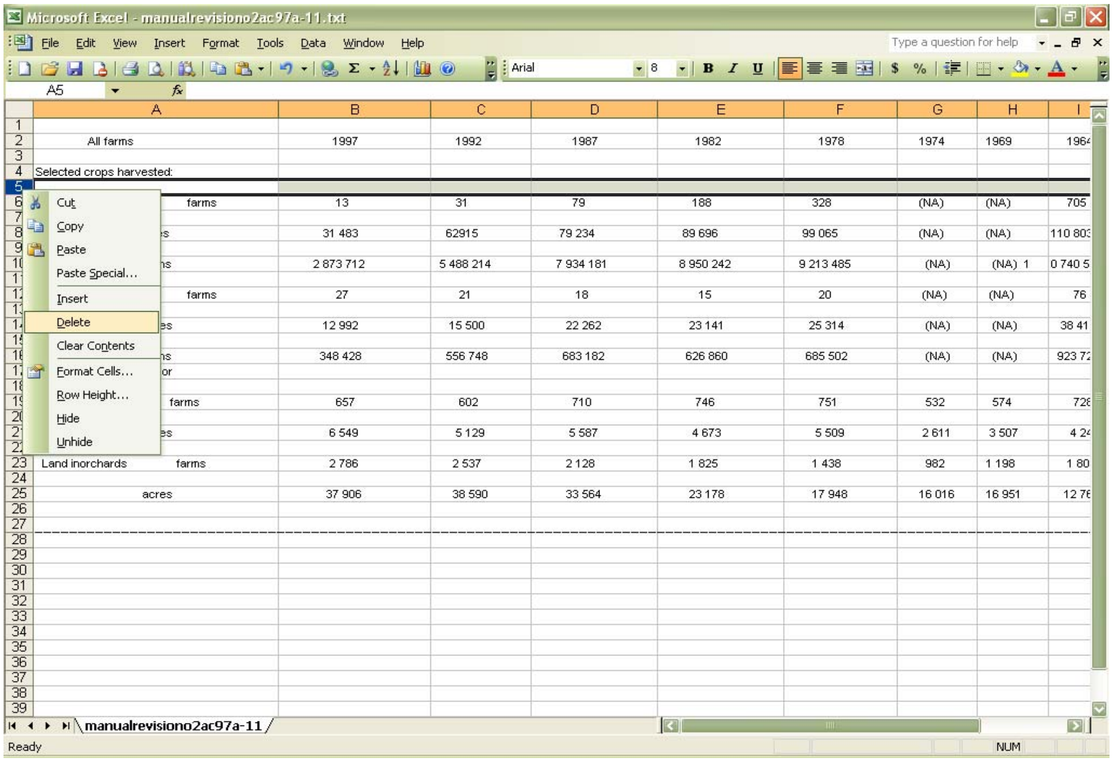
The document will now look similar to the picture shown above in Print Preview view.



Step 13: Select the Page Break Preview button. The view will change to the one shown above.

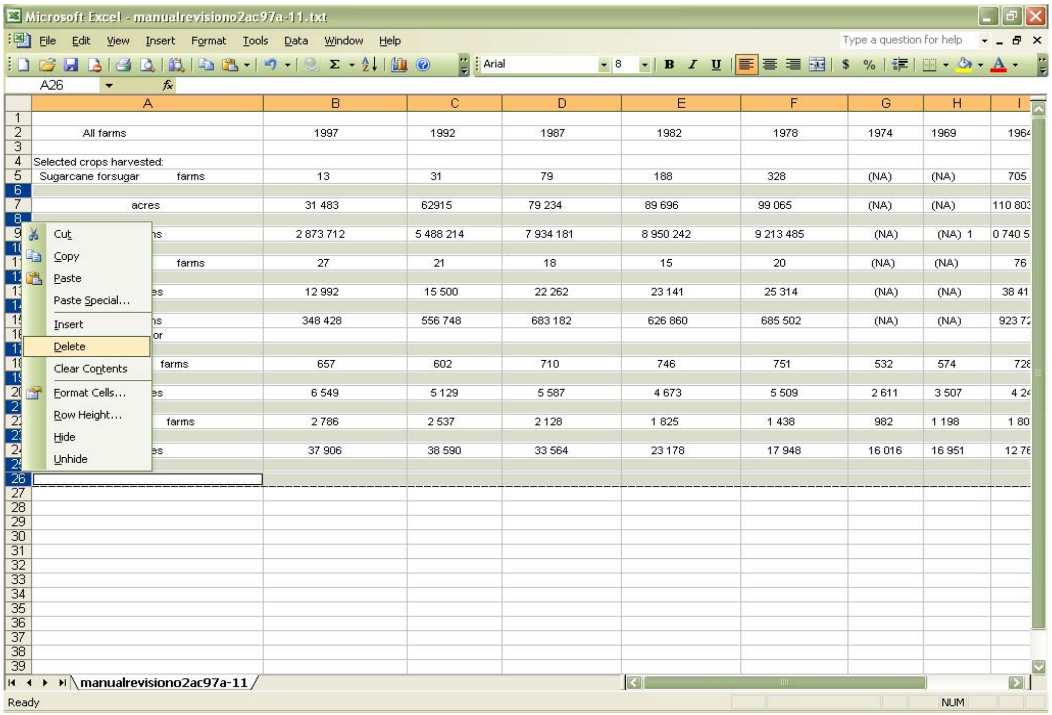

Microsoft Excel - manualrevisiono2ac97a-11.txt
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial
8
A1
Page 1
manualrevisiono2ac97a-11
Ready
NUM

Step 14: Place your mouse pointer directly onto the blue lining at the far right and drag it over until all the data is enclosed into the page break view as shown above.



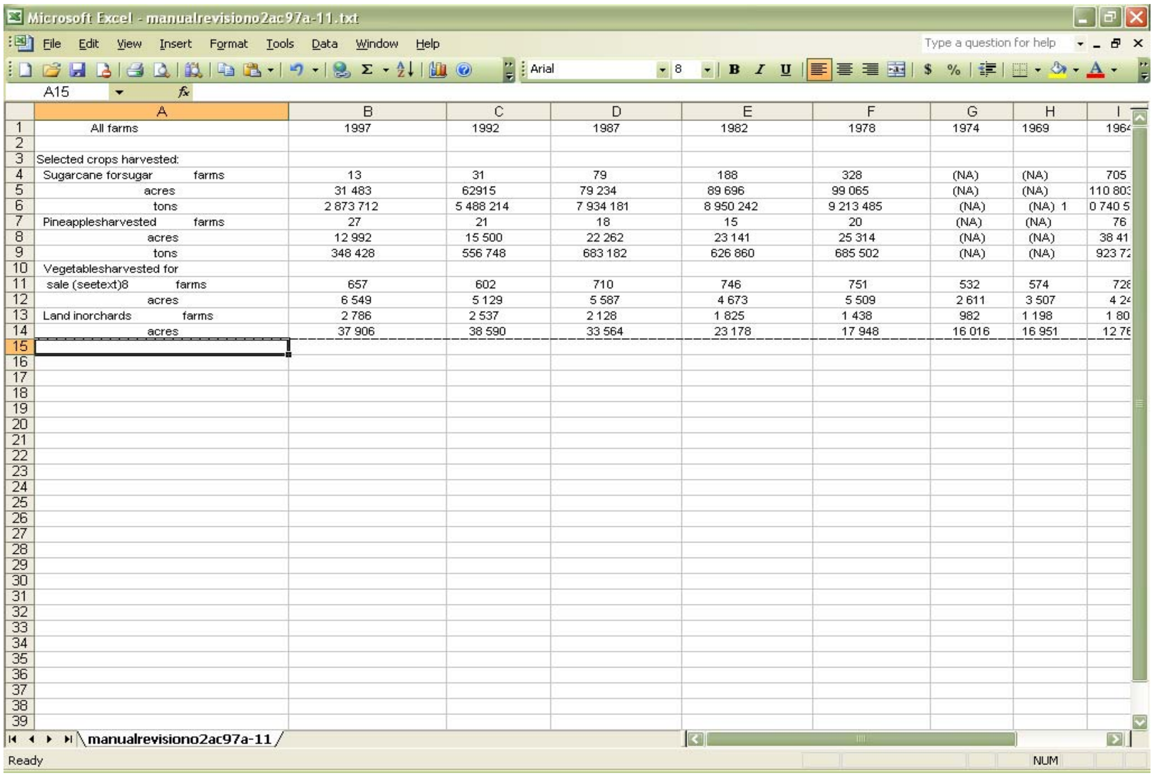
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	1964
3									
4	Selected crops harvested:								
5									
6	farms	13	31	79	188	328	(NA)	(NA)	705
8	s	31 483	62915	79 234	89 696	99 065	(NA)	(NA)	110 803
10	ns	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
12	farms	27	21	18	15	20	(NA)	(NA)	76
14	es	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
16	ns	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 72
17	or								
19	farms	657	602	710	746	751	532	574	728
21	es	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 24
23	Land inorchards farms	2 786	2 537	2 128	1 825	1 438	982	1 198	1 80
24									
25	acres	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 76

Step 15: Go to the toolbar and change the view back to Normal View by selecting View -> Normal. All empty rows may be deleted, if preferred by selecting a row by clicking on one of the numbers on the left side of the document. The row will be highlighted. Right click and select Delete.



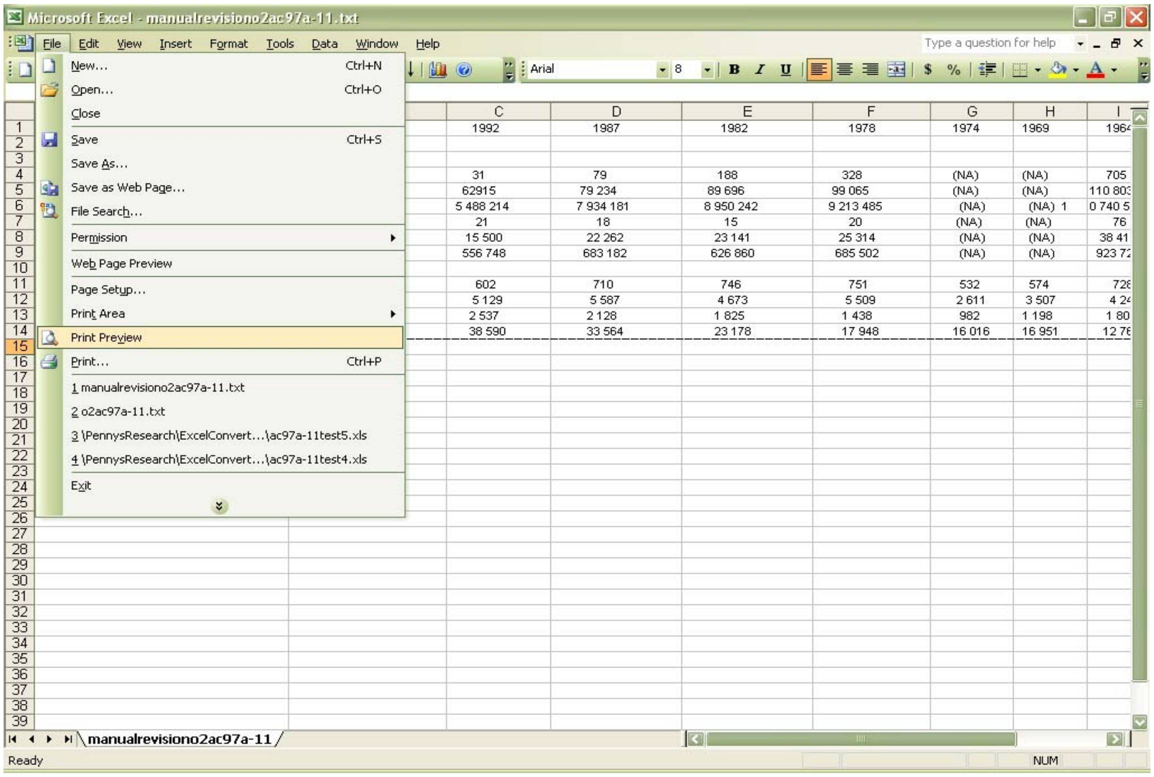
	A	B	C	D	E	F	G	H	I
1									
2	All farms	1997	1992	1987	1982	1978	1974	1969	1964
3									
4	Selected crops harvested:								
5	Sugarcane forsugar farms	13	31	79	188	328	(NA)	(NA)	705
6									
7	acres	31 483	62915	79 234	89 696	99 065	(NA)	(NA)	110 803
8									
9	ns	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
10									
11	farms	27	21	18	15	20	(NA)	(NA)	76
12									
13	es	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
14									
15	ns	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 72
16	or								
17									
18	farms	657	602	710	746	751	532	574	728
19									
20	es	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 24
21									
22	farms	2 786	2 537	2 128	1 825	1 438	982	1 198	1 80
23									
24	es	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 76

Multiple rows may be deleted at one time by holding down the CTRL key and selecting each row then right click over one of the numbers highlighted in blue while continuing to hold down the CTRL key then select Delete.



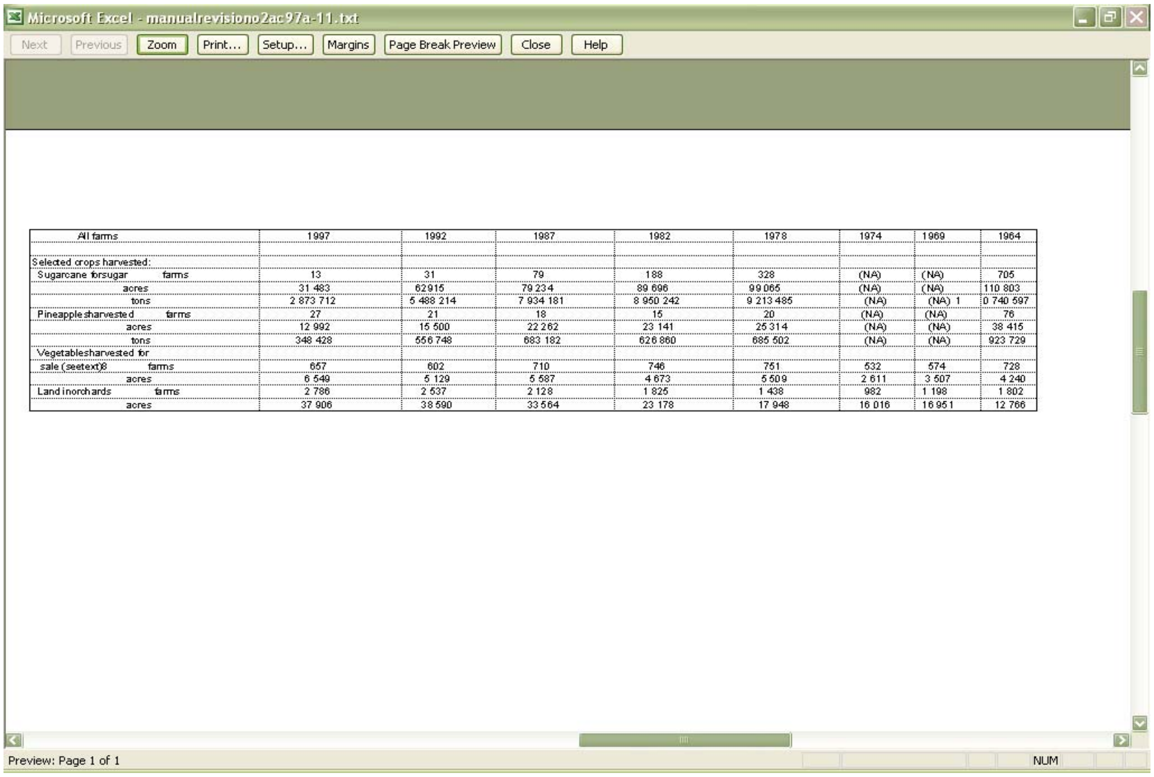
	A	B	C	D	E	F	G	H	I
1	All farms	1997	1992	1987	1982	1978	1974	1969	1964
2									
3	Selected crops harvested:								
4	Sugarcane forsugar farms	13	31	79	188	328	(NA)	(NA)	705
5	acres	31 483	62915	79 234	89 696	99 065	(NA)	(NA)	110 803
6	tons	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
7	Pineapplesharvested farms	27	21	18	15	20	(NA)	(NA)	76
8	acres	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
9	tons	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 72
10	Vegetablesharvested for								
11	sale (seetext)8 farms	657	602	710	746	751	532	574	728
12	acres	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 24
13	Land inorchards farms	2 786	2 537	2 128	1 825	1 438	982	1 198	1 80
14	acres	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 76

The document will now look similar to the picture shown above in Normal view.



	C	D	E	F	G	H	I
1	1992	1987	1982	1978	1974	1969	196
2							
3							
4	31	79	188	328	(NA)	(NA)	705
5	62915	79 234	89 696	99 065	(NA)	(NA)	110 80
6	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 5
7	21	18	15	20	(NA)	(NA)	76
8	15 500	22 262	23 141	25 314	(NA)	(NA)	38 41
9	556 748	683 182	626 860	685 502	(NA)	(NA)	923 72
10							
11	602	710	746	751	532	574	72
12	5 129	5 587	4 673	5 509	2 611	3 507	4 24
13	2 537	2 128	1 825	1 438	982	1 198	1 80
14	38 590	33 564	23 178	17 948	16 016	16 951	12 7

Step 16: For verification purposes, go to Print Preview to view how the document and data will look if printed.



All farms	1997	1992	1987	1982	1978	1974	1969	1964
Selected crops harvested:								
Sugarcane forsugar farms	13	31	79	188	328	(NA)	(NA)	705
acres	31 483	62915	79 234	89 696	99 065	(NA)	(NA)	110 803
tons	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA) 1	0 740 597
Pineapplesharvested farms	27	21	18	15	20	(NA)	(NA)	76
acres	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 415
tons	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 729
Vegetablesharvested for								
sale (seetext)8 farms	657	602	710	746	751	532	574	728
acres	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 240
Land inorchards farms	2 786	2 537	2 128	1 825	1 438	982	1 198	1 802
acres	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 766

The final Excel document in Print Preview view.

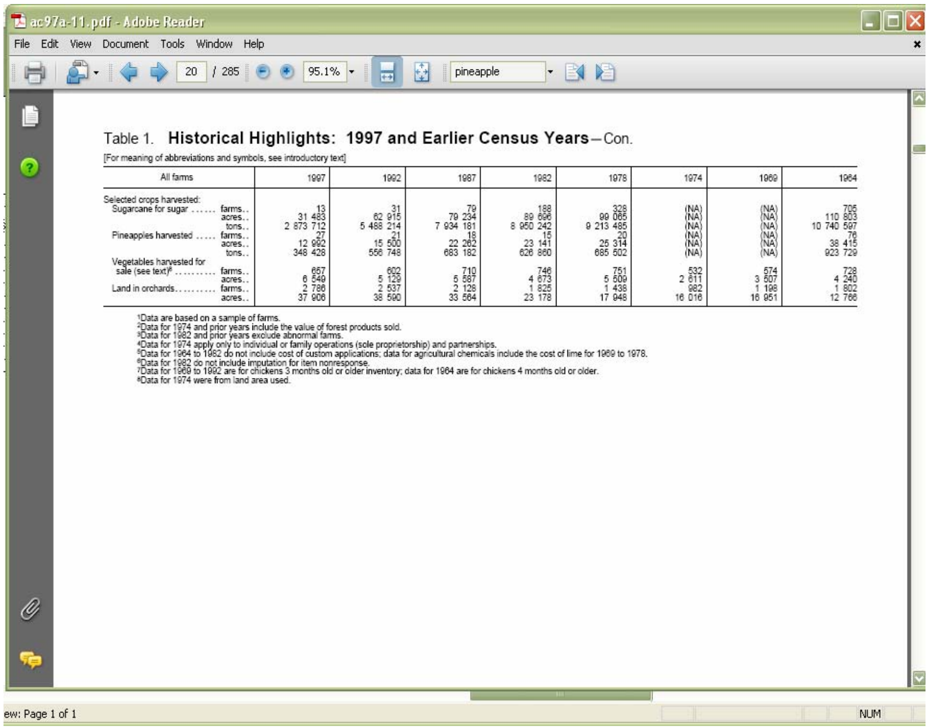

ac97a-11.pdf - Adobe Reader
File Edit View Document Tools Window Help
20 / 285
95.1%
pineapple
ew: Page 1 of 1
NUM

Table 1. **Historical Highlights: 1997 and Earlier Census Years**—Con.

[For meaning of abbreviations and symbols, see introductory text]

All farms		1997	1992	1987	1982	1978	1974	1969	1964
Selected crops harvested:									
Sugarcane for sugar	farms..	13	31	79	188	328	(NA)	(NA)	705
	acres..	31 483	62 915	79 234	89 696	99 065	(NA)	(NA)	110 803
	tons..	2 873 712	5 488 214	7 934 181	8 950 242	9 213 485	(NA)	(NA)	10 740 597
Pineapples harvested	farms..	27	21	18	15	20	(NA)	(NA)	76
	acres..	12 992	15 500	22 262	23 141	25 314	(NA)	(NA)	38 415
	tons..	348 428	556 748	683 182	626 860	685 502	(NA)	(NA)	923 729
Vegetables harvested for sale (see text)[8]	farms..	657	602	710	746	751	532	574	728
	acres..	6 549	5 129	5 587	4 673	5 509	2 611	3 507	4 240
Land in orchards..........	farms..	2 786	2 537	2 128	1 825	1 438	982	1 198	1 802
	acres..	37 906	38 590	33 564	23 178	17 948	16 016	16 951	12 766

[1] Data are based on a sample of farms.
[2] Data for 1974 and prior years include the value of forest products sold.
[3] Data for 1982 and prior years exclude abnormal farms.
[4] Data for 1974 apply only to individual or family operations (sole proprietorship) and partnerships.
[5] Data for 1964 to 1982 do not include cost of custom applications; data for agricultural chemicals include the cost of lime for 1969 to 1978.
[6] Data for 1982 do not include imputation for item nonresponse.
[7] Data for 1969 to 1992 are for chickens 3 months old or older inventory; data for 1964 are for chickens 4 months old or older.
[8] Data for 1974 were from land area used.

The original PDF document viewed with Adobe Acrobat Reader®.

Note: The user should compare the significant data captured in both formats for further conversion verification.

5. Error Messages

5.1. Error 1

System Error Message - Problem with the directory setup: Check to verify that the directory tree is setup correctly with the valid pathnames and PDF documents.

5.2. Error 2

System Error Message - Cannot get a list of files: Check to see if the directory exists.

5.3. Error 3

System Error Message - This error will give the specific error message pertaining to the problem that exists: Check to see if the document is valid and that there were no interruptions in processing. User may need to recompile and re-run P2X.

5.4. Error 4

Program Error Message - Problem with VBSetup in the P2X program. Program uses the directory list to identify which document is being processed: Check document formatting at time of error. Confirm that the user is using valid PDF documents.

6. Further Assistance

For help or further assistance, email: mailto:lpenny_2002@yahoo.com

VITA

LaToyia DeVonne Penny

Candidate for the Degree of

Master of Science

Thesis: DESIGN & IMPLEMENTATION OF A PDF TO EXCEL CONVERSION TOOL (P2X)

Major Field: Computer Science

Biographical:

Personal Data:
Born in Tulsa, Oklahoma, On August 19, 1976, the daughter of Donna and Ollie Penny.

Education:
Bachelor of Science degrees in Computer Science and Mathematics from Langston University, Langston, Oklahoma in May 1999. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2008.

Experience:
Raised as an Air Force brat in Minnesota, Colorado, California, Virginia, Oklahoma City, Oklahoma, and Tulsa, Oklahoma; employed on the production line at BAMA Pies in Tulsa, Oklahoma during summers; employed by Langston University, Department of Mathematics and Department of Business as an undergraduate teacher's assistant; employed by United States Department of Energy during the summer of 1997 and 1999; employed by Oklahoma State University, Department of Computer Science 1999 to 2004 as a graduate RA, TA, and Instructor; employed by U.S. Department of Defense, 552 ACNS/SCOD Tinker Air Force Base, OKC, Oklahoma since 2004 as an I.T. Specialist.

Professional Memberships:
Delta Sigma Theta Sorority, Incorporated.

Name: LaToyia DeVonne Penny Date of Degree: July, 2009

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: DESIGN AND IMPLEMENTATION OF A PDF TO EXCEL CONVERSION TOOL (P2X)

Pages in Study: 78 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

The scope of this study is limited to focus on an implementation of a conversion tool (P2X); developed to automatically convert large batches of PDF tabular data (PDF tables) to spreadsheet format (MS Excel). We begin by introducing the PDF specification standards on table structure. A scenario example of the problem and a description of the conversion tool (P2X) architecture. Specific details of the algorithms and applications used during the PDF to plain text format (PTF) conversion process follows. A brief overview of the reformatting process and a formalization of the table tags that we identified using regular expressions will be introduced. Lastly, a description of the GUI, its images, and functionality will be discussed in the User Interface section.

Findings and Conclusions:

We have implemented a working conversion tool to show the conversion of PDF tabular data to MS Excel spreadsheets can be simple by use of a graphical user interface with user interaction. This system was produced using the high-level programming languages Java and Visual Basic 6.0. These implementations are presented. A user's manual has been incorporated to validate the use of the system and reduce user error. More visuals of the P2X tool to further assist the user with the problems presented throughout this research. Although P2X proved to be a successful conversion approach, it was discovered at the end of the final testing phase that the final output of the text data stored in the Excel spreadsheet file will need minimal manual editing by the user to dispose of unwanted non-breaking space and to suit the individual user’s storage preferences. These preferences are expected to vary on a case-by-case basis.

ADVISER’S APPROVAL: Dr. K. M. George