

Endicia Label Editor

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirement for the

Degree of Bachelor of Science

by

Eric Baicker-McKee

Binh Pham

Jing Zhang

4/23/2012

Sponsored by:

DYMO Endicia

Approved by:

Professor David Finkel, Advisor

Professor Emmanuel Agu, Advisor



Endicia

Abstract

Endicia's Label Server's core functionality is to generate shipping labels. The LabelServer's rendering engine takes a predefined layout and generates labels in different formats. We added visualization of the layouts that was previously code-defined. We researched layout editors in terms of price, feature, and popularity and chose Microsoft Visio. We wrote a program that translates the layouts generated by the editor into the data structure used by the Label Server's rendering engine. We also wrote code to translate the current code-defined layouts into the layout editor's format. This project helped accelerate the new layout design process and made the process available to a wider client base.

Executive Summary

Despite the convenience of sharing information through the Internet and phone service, the demand for shipping and mailing service is still significant. Thousands of packages and documents travel every day around the globe. Founded in 1982, Endicia is a company that strives to create software solutions to make the postal service easy for its customers. Our project involved one of Endicia's products, the Label Server. Endicia's Label Server allows customers to integrate United States Postal Service into their own application. The main functionality of Label Server is to produce shipping labels.

Our goal in this project was to implement a solution to enhance the efficiency of label design. It reduces the amount of time Endicia employees spend on designing shipping labels. Our solution had to be able to work with the Label Server. Before our project, shipping labels design within the Label Server consisted of many lines of code, and it was extremely complicated to create or edit label designs. Our solution to the problem includes a module that allows designers to create/edit shipping labels in a graphical editor program and produces a label layout format that can be handled by the Label Server.

At the beginning of our project, we researched a variety of graphic editing programs such as Microsoft Visio, Microsoft Expression Blend, and Inkscape. We picked Microsoft Visio as the graphic editor based on our criteria as well as the user's criteria. The criteria included user friendliness and customizability.

After our choice of Microsoft Visio was approved by our mentors at Endicia, we proceeded to customize Microsoft Visio. We were able to achieve this by developing a Visio Add-in, which is a way for programmers to add custom functionality to Microsoft Visio. The functionality of our Visio Add-in is to translate label layouts designed in Microsoft Visio to Label Server's data structure and vice versa, along with several tools to enhance the label design process. We also added a new feature to the Label Server which allows Endicia's employees to convert already existing label designs into Microsoft Visio format.

The challenge that we faced while implementing the Visio Add-in was to come up with a consistent way of representing binding fields. Binding fields are components of shipping labels that update upon user's request; one example of a binding field is the destination address. We used the Visio Shape Data to keep track of information about the binding fields. The Shape Data

is a feature of Microsoft Visio that allows users to define their own data. Our Visio Add-in helps Endicia's employees manipulated binding fields' data easily.

Our solution introduced a new tool for Endicia's employees. For this reason, we created a user's manual for Microsoft Visio and our Visio Add-in. This manual will help future users to use our solution with ease. We also create a developer's manual that describes the structure of our implementation and instruction on how to maintain and update our program.

Our solution served as internal software that makes the process of label editing and designing much easier. We successfully designed a new shipping label called Global Express Guaranteed using our program. This label was released to Endicia customers on February 28th, 2012.

Table of Contents

Abstract.....	2
Executive Summary	3
Table of Contents.....	5
Chapter 1 Introduction.....	7
Chapter 2 Background	8
2.1 Endicia	8
2.2 DYMO	9
2.3 Technology.....	9
2.3.1 XAML.....	9
2.3.2 SVG	9
2.3.3 XML	10
2.4 Layout Editors	11
2.4.1 Aviary.....	12
2.4.2 Inkscape	13
2.4.3 Microsoft Expression Blend.....	13
2.4.4 Microsoft Visio.....	13
2.4.5 Omnigraffle	14
2.4.6 Sketsa SVG Editor	14
2.5 Label Server	14
2.5.1 Label Server Operation	15
2.5.2 Rendering Engine	16
2.5.3 Drawing Fields	18
2.5.4 Binding Fields	19
Chapter 3 Requirements	20
The Demo	22
Chapter 4 Tools	24
4.1 Visio	24
4.1.1 Visio Stencils.....	24
4.1.2 Visio Masters	25
4.1.3 Visio XML Schema.....	27
4.1.4 Microsoft Visio Software Development Kit (SDK)	28
4.1.5 Microsoft Visio Add-In	29
4.1.6 Microsoft Visio User Manual.....	31
4.2 XML Serialization.....	31
4.3 Markup Languages.....	32
4.4 PDF2Picture	34
Chapter 5 Program Design.....	35
5.1 Program Architecture.....	35
5.2 Program Flow	36
5.3 Add-In Side Operations	37
5.4 Server Side Operations	39
5.5 Round-tripping.....	39

5.6 Maintainability.....	41
5.7 Testing	42
Chapter 7 Results and Conclusion	44
References	46
Appendix A	50
Appendix B	52
Appendix C	59

Chapter 1 Introduction

“The Postal Service delivers mail six days a week to nearly 140 million addresses. Every year this number increases by 2 million.” -Congressman Joe Baca. [75]

Even with the speed and convenience provided by the modern innovations of email and cell phones, there is still a need for the United States Postal Service. Thousands of packages and documents are shipped every day to locations around the globe; and as such, there is a demand for easing the process of using the postal service. Endicia aims to fill one of these demands. Founded in 1982 as a consulting company, Endicia broke into the postage industry with the release of an address confirmation software called DAZZle. Now, they strive solely to solve the needs of the users of the postal industry, developing software that prints postage and labels.

Our goal in this project was focused on the API that Endicia uses to print labels, called the Endicia Label Server. We were asked to develop software to reduce the amount of time that Endicia employees spent designing labels by providing the ability to use a third party graphical editor with the Endicia Label Server. In this paper, we describe the process of creating the product that provided a solution to Endicia’s label design troubles. We will discuss the preliminary research we performed when developing the skills necessary to complete the project, the requirements given to us by our advisor, Tiberiu Motoc, the tools we used in order to design the final product, and finally the process and design of our code. All our work in this project was implemented before we left Endicia and is currently used by Endicia employees and, as of the publication of this paper, will soon be used by Endicia customers as well.

Chapter 2 Background

2.1 Endicia

Originally founded as a technology consulting company in 1982 under the name PSI Associates, Endicia is a software developer whose products are aimed towards simplifying the postal services. Although their products had been in the making since the founding of PSI, Endicia officially entered the software postage industry in 1989 (while still known as PSI Associates) with the release of the product Envelope Manager and the invention of Dial-A-ZIP Address Verification. PSI continued to release products for the postage industry until the year 2000, when a subsidiary of PSI Associates was formed to manage a product called the Endicia Internet Postage, and has produced shipping technology since. In 2007, Endicia reported their software having two billion dollars worth of postage since the year 2000. [1]

Endicia's product line consists of software solutions like the Endicia Label Server (which is designed to allow customers to print shipping labels) [2] and physical products such as their USB postal scales [3] [4]. One of their products is the printable postage series. When combined with a DYMO brand printer and DYMO printable postage sheets, Endicia's software is capable of printing out postage stamps of any value for First-Class Mail, Priority Mail, Express Mail, Media Mail, Library Mail, or Parcel Post, domestic or international. [5]

Endicia is partnered with a large number of companies in order to provide their software and hardware products. Three of the most notable are the United States Postal Service, Microsoft, and DYMO. [6]

In 2007, Endicia was acquired by Newell Rubbermaid, a company with branches in office supplies, household products, and hardware products [7]. This merger allowed Endicia to partner with other Newell Rubbermaid acquisitions to provide hardware pairings with their software. DYMO, a subsidiary of Newell Rubbermaid, fit this need with their brands of printers. [1]

2.2 DYMO

DYMO is a company based in California, and was established in 1958 as an embossing company [9]. In 2005, DYMO was henceforth acquired by Newell Rubbermaid [10]. DYMO provides a wide range of products for the office, including label makers, computer-connected products, software and online service. DYMO includes four different branches: CardScan, Mimio, Endicia, and Rhino [8]. Endicia collaborated with DYMO on shipping products, such as Endicia's Shipping Labeler project.

2.3 Technology

We shall now review the basics of two potential file formats that we could be interacting with during the project. These two file formats, XAML and SVG, are designed for use in specifying a layout.

2.3.1 XAML

Short for "Extensible Application Markup Language", XAML is a declarative markup language [27] intended to be used when creating Graphical User Interfaces (GUIs) [28]. XAML is used to simplify the process of creating UI in a .NET Framework application. Endicia considered XAML as a potential language for use in the Endicia Labeler.

2.3.2 SVG

Scalable Vector Graphics (SVG) is a XML-based specification to describe two-dimensional graphics. SVG consists of the XML-based file format and an API for application development. Vector graphic shapes, images and text are supported [30]. Graphic files generated by SVG are compact comparing to JPEG [31]. They are also zoom-friendly, which means that a SVG image can be enlarged without being pixelated [32]. Printing SVG images is independent of devices. They maintain their original quality while printed with printers of different resolutions [31]. Since SVG is text-based, it can be edited with any text editor [32]. Mouse and keyboard event handling is available for all SVG image objects [30].

SVG is applied to various industries, including mobile, print, web application, user interfaces and Geographic Information System. Its device-independent support of shapes, text and images provides an advantage in printing business [30].

There are open-source and commercial SVG editors available. Inkscape is an Open Source vector graphics editor using SVG as its native format [33]. SVG-edit is a web-based SVG editor driven by Javascript, HTML5 and CSS without server-side functionality [34]. Sketsa SVG Editor is a vector image drawing application developed by Kiyut [36]. It allows users to draw SVG image visually or directly edit XML format code [35].

SVG was released in 1999 by World Wide Web Consortium (W3C) [37], which is an international community developing open standard for the Web [38]. Before that, there was no Web standard available for vector graphics. In 1998, Adobe, IBM, Netscape and Sun submitted Precision Graphics Markup Language (PGML) to W3C. In the mean time, HP, Macromedia, Microsoft and Visio submitted Vector Markup Language (VML) to W3C. As a result, they formed the SVG working group and developed requirements for SVG. [39]

2.3.3 XML

Extensible Markup Language (XML) is a markup language that is used to structure, store and transport data [39]. It derived from Standard Generalized Markup Language (SGML), which is a system for defining markup languages published in the 1980s [44]. As an application profile of SGML, XML was designed in 1996 to meet the need of structuring and transferring large-scale documents over the web [45]. The development goals for XML focuses on the ease to read, create, or use XML documents. Files generated in XML are in text format and have .xml as their file name extension. They can be viewed with text editors or web browsers. XML files are readable for both human and computer [40]. Users are able to figure out the structure and meaning of the information stored in an XML file by reading it.

The format in which data is structured is not pre-defined in XML standard. Instead, it is created by the author of the XML document. The author defines the markup, which organizes and labels the content stored in the XML document. The markup that starts with a left angle bracket (<) and ends with a right angle bracket (>) is called a Tag [41]. An XML component that begins with a start-tag, such as <note>, and ends with a matching end-tag with a slash, such as </note>, is an Element. The content of the Element exists between these two tags and it can be strings of characters or a set of child elements. The string that follows the left angle bracket in a tag stands for the type of the element [43]. For some elements, the start-tag may also define the attributes of an Element. For example, <note title="This is an example"> is a start-tag for the "note" Element and its title is "This is an example". Since an Element may contain multiple child

Elements, an XML document is in a tree structure with each Element defined by its boundary tags.

XML is widely used as a mechanism for data storage and exchange. Since the data is in plain text, it is independent of the databases or devices used for storage. Data sharing among different systems is made easy [46]. Exchanging data over the web between applications using incompatible data formats can lead to be complex. XML is an effective solution since it can be read by different applications. Another advantage of using XML is that, when experiencing hardware or software system upgrade, the need of data conversion is minimized. The possibility of losing or corrupting data during conversion is reduced. Considering its simplicity and compatibility, we listed XML as an alternative for data transfer between the label editor and Label Server.

2.4 Layout Editors

A preliminary task for our MQP was to select an appropriate layout editor for use within the project. Based on the recommendation of Tiberiu Motoc [11], our contact within Endicia, we looked at two editors, Inkscape and Microsoft Expression Blend, and then looked at various competitors of the two products. We were looking for different options for use within this project- from the price of the product to the ability to export to SVG or XAML. Overall, we discovered over twenty different competitors of Inkscape and Microsoft Expression Blend, and we narrowed the list down to 8 products to perform further research upon based off of basic project descriptions. The results of our preliminary research are compared below in the form of a table (Table 1). We were able to eliminate two out of these eight, Adobe Fireworks CS5 and sK1, as their export abilities did not meet Mr. Motoc's guidelines for file extension exports. The remaining six we systematically tried and evaluated to see whether or not they would be useful to our project, our evaluations concluding in either a recommendation for use or a rejection.

Table 1 Layout Editors Features

Layout Editor	Company	Price	Operating Systems	Scripting support	File type exports
Adobe Fireworks CS5	Adobe	\$149/\$399	Windows XP/Windows 7/Mac OS X v. 10.5.8 or greater	Yes	Neither SVG nor XAML
Aviary	Aviary	Free	Any OS	No	SVG
Inkscape	Inkscape	Free	Windows/ Mac OSX/ Ubuntu	Yes	SVG
Microsoft Visio 2010	Microsoft	\$250/\$550/\$1000	Windows	Yes	SVG
Microsoft Expression Blend	Microsoft	\$599/\$349	Windows	Yes	XAML
Omnigraffle 5	Omnigroup	\$150/\$300	Mac OSX	Yes	SVG
sK1	sK1 Project	Free	Ubuntu	Unknown	PDF
Sketsa SVG editor	Kiyut	\$89.00	Windows/ Mac OSX/ Linux	Yes	SVG

[15]-[27]

2.4.1 Aviary

Aviary is the first product that we examined as a potential layout editor. Aviary has a variety of features that we considered potentially useful for the purposes of our project- it has the ability to export to SVG, is free, and will operate on any modern operating system. Additionally, Aviary had a feature that we looked at explicitly with the Endicia Labeler in mind- Aviary is an

online product. This functionality would mean that it would be possible for any computer with internet access to be able to edit a layout- an invaluable resource for a company with employees on the move. However, two aspects of this program conflicted with our interests in the project- the program did not run quickly, and the user interface was difficult to use and not advantageous in comparison with the other editors we used. With these two factors in mind, we rejected Aviary as a potential layout editor.

2.4.2 Inkscape

As one of the initially recommended tools by Tiberiu Motoc, we investigated Inkscape as a potential layout editor. We considered Inkscape as it meets basic requirements: it's free to download, operates on Windows, Mac OSX, and Ubuntu Linux, and it is able to import/export/edit SVG files, and therefore is able to communicate directly with the Endicia Label Server. Inkscape has a straightforward and easy to interact with user interface. We were able to replicate a label with approximate likeness to an actual label within a span of 30 minutes, and then we were able to export the image into an SVG file and import it into a different editor. Additionally, we were able to import SVG files made in separate editors into Inkscape, and were there able to edit these files. This functionality is all that is needed for the project, and as such we recommended Inkscape as a potential layout editor.

2.4.3 Microsoft Expression Blend

Microsoft Expression Blend is a user experience design tool that we can use to build a label layout editor. It does not serve as a layout editor itself, but it is an option to develop the front end of our project and work collaboratively with Visual Studio, which can be used for the development of our core "translator". It is the only software we found that can export to XAML file format. One of the advantages of Expression Blend is that it works with .NET framework, which the projects at Endicia are built upon. Applications can be developed both visually and on a code base. As this product is not itself a layout editor, rather a tool for building such devices, we rejected Microsoft Expression Blend.

2.4.4 Microsoft Visio

Microsoft Visio is another product that has been considered as a potential layout editor. It is a professional tool for diagramming with pre-drawn shapes. Its user interface is similar to Microsoft Office, so it is easier for users who are familiar with Office to recognize and navigate

the menu and tool bars. It also draws lines, shapes and text boxes the way that Microsoft Office does. Files generated by Visio can be saved in many different formats, including SVG and pdf. The task of editing labels can be accomplished with the very basic functions of Visio. Although Visio contains greater functionality than is necessary for this project, we still consider it to adequately cover the requirements, and as such, we recommend Microsoft Visio.

2.4.5 Omnigraffle

Another editor that we brought up for consideration was Omnigraffle. Unlike the other products, we did not have the opportunity to experiment with Omnigraffle, as the only operating system it operates upon is Mac OSX. In spite of the features that Omnigraffle possesses, no member of our group currently has access to a Mac. As a result, we were forced to reject Omnigraffle as an option for this project.

2.4.6 Sketsa SVG Editor

One of the tools that we decided to test was Sketsa SVG Editor. We considered Sketsa to be a reasonable choice for layout editor because of its following features: ability to import/export/ and edit SVG formatted file, ability to edit SVG source code directly. Sketsa is also available cross platform: Windows, Mac OSX and Linux, and the price is reasonable. Since Sketsa is a commercial product, we were able to download and test the trial version of the software. We tested this product for a while, and after careful consideration, we decided to reject Sketsa based off of issues with the user interface's usability.

2.5 Label Server

Endicia Label Server is a web-based Application Programming Interface (API). The Label Server allows customers to integrate US Postal Service into their application [2]. The Label Server operates independently of the customers' system. The main function of the Label Server is to produce a shipping label upon a customer's request. The Label Server's users send requests to the Label Server in XML format, and then the server will generate a shipping label complete with postage, verified addresses and service barcodes. Other than the Label Server's main function of generating shipping labels, the Label Server also includes other postal features such as address verification and postage rate calculation.

2.5.1 Label Server Operation

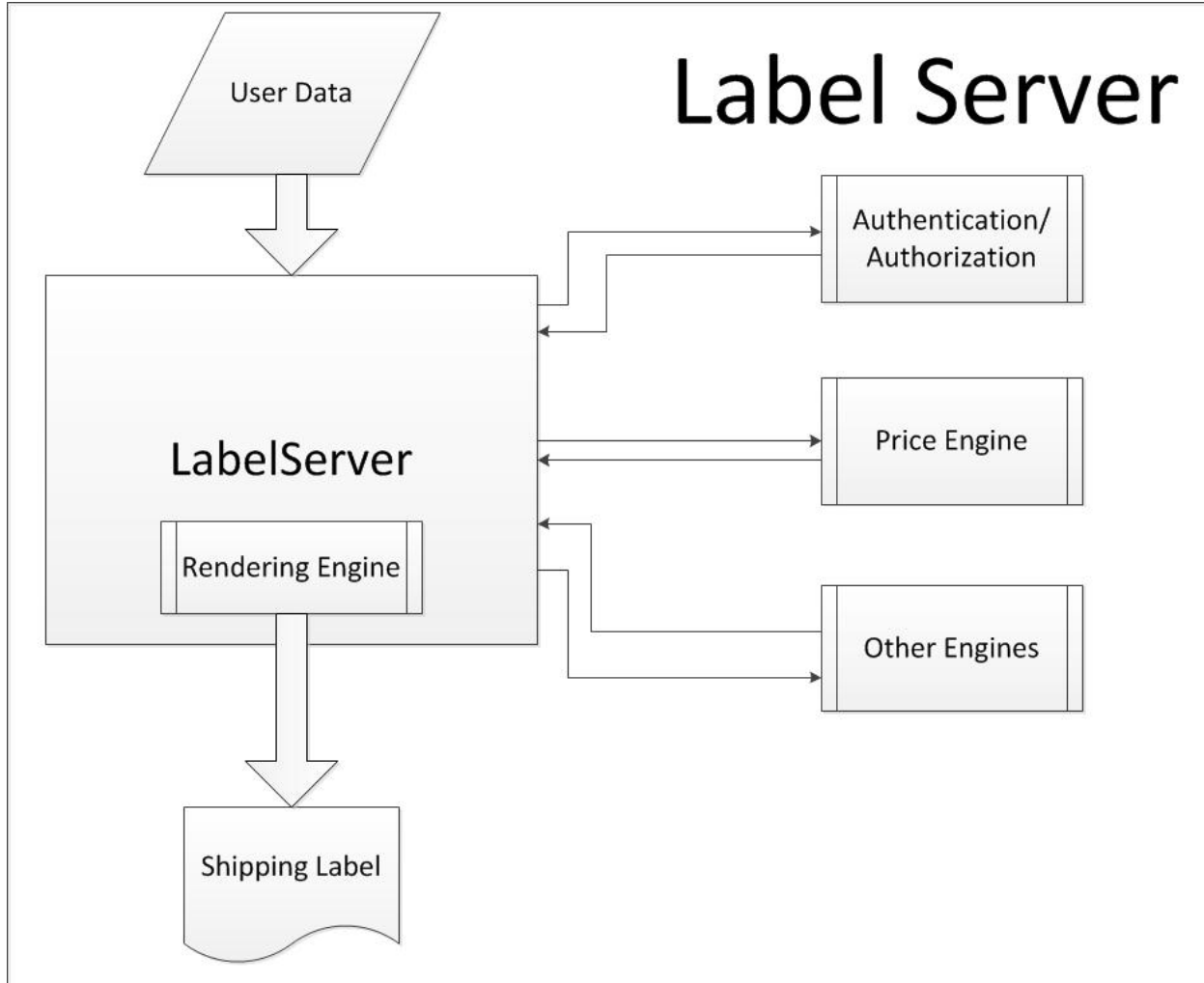
The main functionality of Endicia's Label Server is to produce shipping labels based on users' requests. The user's request contains information necessary for the Label Server to generate a shipping label, such as the address of the destination. Upon receiving a request for a shipping label from a customer, the Label Server will communicate with a variety of other servers to authorize the user, determine the price for the shipping label, verify addresses, and other necessary functions. The following Figure 1 demonstrates the main operation of the Label Server.

When a request for a shipping label is sent to the Label Server, the request contains additional user data. The user data contains necessary information to create a shipping label such as package type information, ship to address, ship from address, etc. The Label Server utilizes the information to create the shipping label that satisfies the user's request.

After the Label Server receives a request for a shipping label, it will send a request to the authorization engine to authorize the user. The Label Server will not continue if the authorization engine does not send back a message to confirm the user. After user authorization, the Label Server will send the request's information about package type, addresses of destination and origin to the pricing engine to determine pricing for the package. After the price has been determined, the Label Server will continue to communicate with other engines similar to the pricing engine to collect information necessary to create the shipping label. Examples of other necessary information include address verification and tracking information.

After the Label Server has finished collecting information about the shipping label, the rendering engine within the Label Server generates the shipping label. The rendering engine will load the specified label layouts, bind user's data, and then the rendering engine will generate a shipping label in a format specified in the user's request.

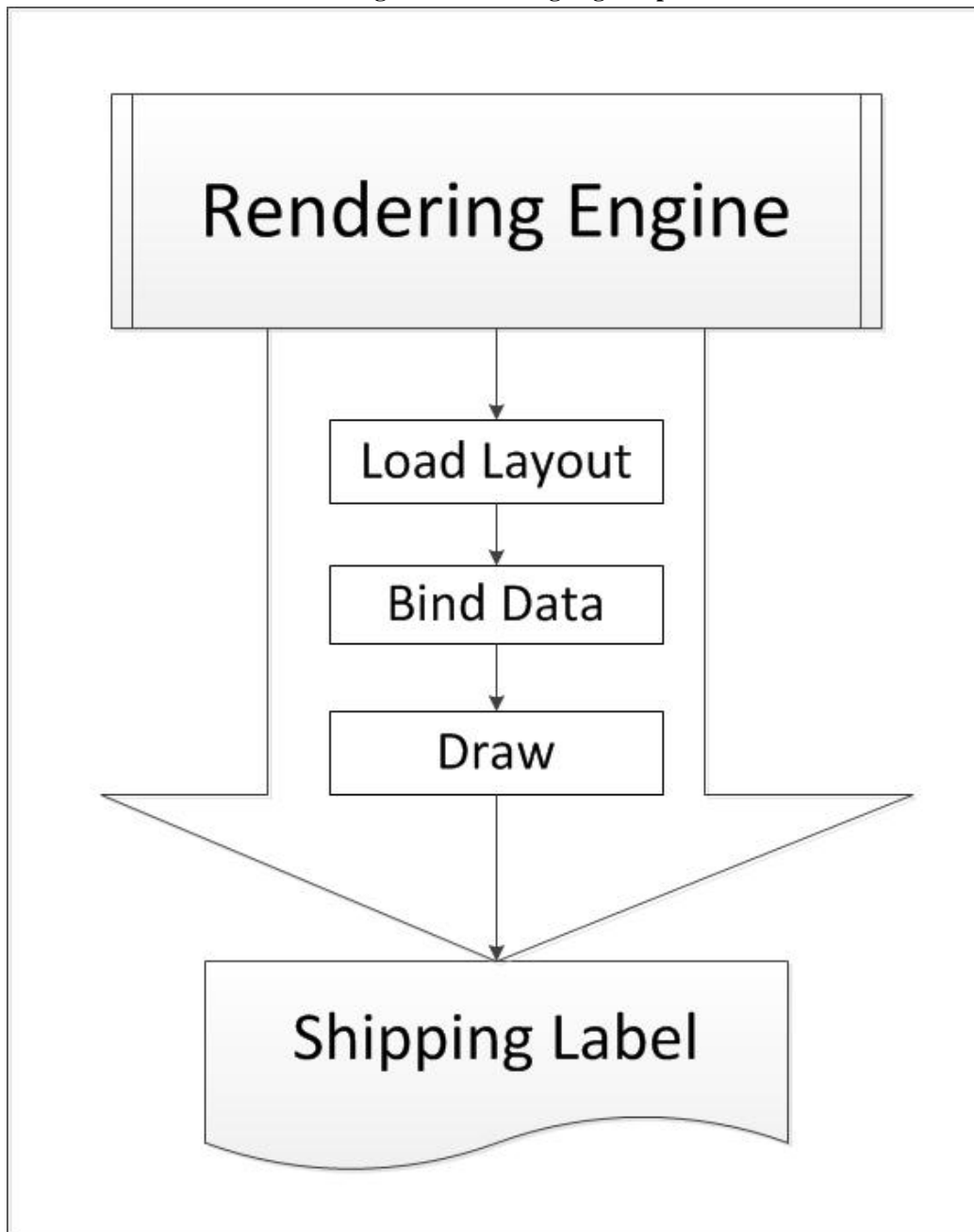
Figure 1 Label Server Operations



2.5.2 Rendering Engine

The rendering engine is a component of the Label Server. It is responsible for creating shipping labels after the Label Server has finished collecting data about the label. Figure 2 demonstrates the rendering engine's process for creating a shipping label.

Figure 2 Rendering engine operation



The first step of the rendering engine's process is to load the shipping label's layout. Different types of labels have different layouts; therefore, the rendering engine is in charge of determining which layouts to load based on the user's requests. The label layouts are hard coded in the Label Server. There are approximately fifty label layouts in the Label Server. Each label layout is represented in the Label Server as a unique class which contains information about the dimension of the label, the name and the type of the label. Each layout class also contains a list of components. These components represent aspects of the label such as a line, a rectangle, a text

box, a barcode. They are called label fields. The rendering engine loads a label layout by instantiating the class which corresponds to the layout being load. During instantiation the dimension of the label, the name of the label are set to the layout class; also, the list of label fields is added to the class.

After the layout of the label has been loaded, the rendering engine proceeds to the second step. This step is called the data binding step. The data specific to the user's requests is bound to the label during this step. The rendering engine will traverse the list of label fields in the layout, identify the fields that represent the user's data and replacing the data in those fields with the user's data. For example, the destination address of a shipping label is the data specific to a user's request; the rendering engine will look for the text box field representing the destination address, then the rendering engine will replace the text value of that field with the destination address of the user's request. The label fields whose data will be changed during this process are referred to as binding fields.

The last step in the rendering engine's process of generating shipping label is the drawing step. A shipping label is generated after this step in one of these formats: EPL2, ZPLII, GIF, JPEG, BMP, PNG, PDF [2]. During this step, the rendering engine traverses the list of label fields and draws each of the field on a virtual canvas. The canvas afterward is converted into one of the formats listed above.

2.5.3 Drawing Fields

A label field represents a component of the label layout. The component can be as simple as a line, a rectangle, or as complicated as a barcode or indicia. Each type of label field is represented in the Label Server as a class. Each label field class contains data specific to its type. This data is used when the rendering engine draws the shipping label. For example, the label field class that represents a line contains coordinates of the two points of the line, and the class also keeps information about the style of the line (dotted line vs. solid line).

Every label field class has a method which will draw the label field on a specify canvas. For example, the draw method of the line class draws the line on a canvas. The draw method is called during rendering engine's last step. The rendering engine traverses the list of label fields, and for each label field it calls the draw method to draw the field on the canvas. These draw functions are specially made to be able to handle different types of output formats of shipping labels. For example, EPL2 is an output format that the user can request. EPL2 is a form of

programming language for printers with a small memory buffer [66]. If the user requests a shipping label in EPL2 format, the draw functions of label fields will not draw an image but instead generate code in EPL2 language.

2.5.4 Binding Fields

A binding field is a special type of label field. Binding fields act as placeholders in a shipping label. These placeholders are replaced with the user's data during the rendering engine's data binding step. The binding fields serve as a marker of where some specific element of the shipping label will be.

The binding fields are separated from the normal label fields by their Field Identification Number (FID). All label fields have an FID, and all the label fields which are not placeholders have an FID of negative one. A binding field has an FID unique to what the field is a placeholder of. For example, the binding field for barcode has an FID of 101. During the data binding step, the rendering engine will look for the label fields that have the correct FID and then replace the data in that field with the user's data. In another example, the rendering server has the barcode data for a user's request, and the rendering engine will look for a label field with an FID of 101, and then will replace the placeholder barcode data with the correct barcode data.

Chapter 3 Requirements

Endicia's Label Server has, since its conception, used hard coded definitions for its labels. The process of creating or editing a label was an enormous task, that required members of the Endicia Label Server team to spend hours of work designing a visual label by filling out coded object definitions. The goal of our project was to develop a tool that would allow anyone, regardless of technical skill, to modify or create USPS labels. They wanted to be able to use an external vector editor (such as Inkscape or Microsoft Visio) to generate an file written in a markup language, which the label server could read and parse into the Label service's data structure elements. Our role in this project is to accomplish two tasks, the first to select an appropriate vector editor and customize its settings to allow non-engineers to create labels without assistance from the engineering department, and second to write the software to translate the output from the editor to the coded structures in the label server.

For the first aspect of the project, we were given two sets of requirements. During the PQP, when we were only starting the project, we were given a single requirement for the Layout Editor during a conversation with Tiberiu. He asked us to generate a list of editors that were well suited for designing labels. With this as our guideline, we generated a list of a few editors, made recommendations based on our research, and passed the list along to Tiberiu in our report. When we arrived at Endicia, he asked us to enhance upon our research by comparing our top two choices, Microsoft Visio and Inkscape, with an editor he had recommended when conceptualizing the project, Microsoft Expression Blend. He met with us and gave us a specific list of what was required of the layout editor for completing this project.

Table 2 shows the results of our experiments in Microsoft Visio, Microsoft Expression Blend, and Inkscape when compared against the list of features requested to us by Tiberiu. This table was generated by working with the products to accomplish specific tasks that would test for specific features, i.e. to create a basic label to test ease of use, and then discussing with the group about the successes and failures of the specific product. We attempted to base our answers from the mindset of a label designer and from the mindset of software engineers trying to develop our project or expand upon a currently existing base, and as such all our results are fair, but subjective. While some of these categories are self-evident, some of these features required additional research and experimentation to determine.

Table 2 Layout editor requirements

Requirements	Microsoft Visio	Microsoft Expression Blend	Inkscape
Ease of use	X		X
Availability of new features	X	X	
File output parsability		X	X
Supports good software engineering practices.	X	X	
Supports creating basic shapes	X	X	X
Supports design of new shapes	X	X	
Allows preview of label before production	X	X	X
Supports preview of custom-defined shapes	X		
Supports vertical text	X	X	X
Supports rotation of grouped items	X	X	

When we examined the availability of new features, we looked at whether or not it was possible to create custom buttons (Like an export button), or to create dropdown menus for quickly usable items. We found that Visio and Expression Blend had capabilities for modifying their UI in ways that would assist a designer in creating labels. Visio had the best interface of all, with its stencil system allowing quick access to shapes [47], and a customizable ribbon feature gave users the ability to modify or expand upon the functionality of the existing UI [48][49]. Microsoft Expression Blend, while also capable of customizing ribbons [50] and providing quick access to shapes, showed a critical failure in that their quick access bars did not provide an adequate preview of the item that was to be inserted. Inkscape, while open source [16], was deemed deficient in this category as the process of acquiring the functionality of either customizable ribbons or quick access menus was difficult to look up and more so to develop.

The file output parsability category was a critical category to us, the developers. Our ability to access the information contained within these files was necessary in all aspects of the project, i.e. defining label field properties. We were able to test this category by designing an example with multiple types of shapes, and many different entries, saving them into an appropriate file type, and then manually examined the output to determine whether or not we (and our program) could determine the type of shape described and the properties contained within. In this category, Microsoft Expression Blend was the strongest competitor, as the XAML file it produced was clean and easy to read. Inkscape's files were a close second, with a readable SVG file. Microsoft Visio, however, had by far the worst output, with an SVG file that was cluttered, overfed with information, and poorly organized.

One of the largest concerns that Tiberiu had with the project is the potential that an upgrade to the layout editor software could ruin functionality. For the supports good software engineering practices category, we looked at whether or not the product posed a risk of version control. We determined that Visio and Expression Blend would be less dangerous when it came to upgrades. In the event of software upgrade (i.e. Microsoft Visio 2015), the developers could upgrade if it did not interfere with the functionality of our project or remain with Visio 2010 if upgrading would impair functionality. However, Inkscape is open source software and can undergo major structural changes rapidly. Furthermore, we discovered that finding previous versions of Inkscape to be difficult, if not impossible, to locate. This could lead to a disastrous situation in which a software upgrade could render the entire system useless.

The remaining categories refer to the ability of the user to have the capabilities necessary to design a label completely and quickly. The ability to draw basic shapes, such as a rectangle or a text block, was a core piece of functionality on which no editor failed to perform. However, when it came to the ability to design custom shapes, Inkscape was thoroughly unable to perform, and Microsoft Expression Blend ran into some usability problems in that the custom shapes did not come with a preview option. Visio's interface provided the best functionality in terms of allowing users to easily create labels.

After our testing, we compared the performance of all three editors. Inkscape was not a valid option, as its general lack of features rendered it difficult to use and failure to maintain good software practices was a liability to the system. Microsoft Visio and Microsoft Expression Blend were closely matched, Visio outperforming Expression Blend with its superior UI, but Expression Blend exceeding Visio in terms of the file output. Ultimately, we determined that the UI of Microsoft Visio was the closest to what our mentor had envisioned for the final product, and presented it as our choice. Our mentor, after being shown the capability of Visio, gave us permission to continue with Visio as our layout editor.

The Demo

After choosing a layout editor to operate with, we presented our mentor with our findings, and declared Microsoft Visio to be the best layout editor out of the available choices. In order to prove our case, we needed to show our mentor the strengths and capabilities of

Microsoft Visio, and to that end, we designed a short demo to show how we envisioned the final product to appear.

The demo was a short demo, mostly running within a 5-10 minute range. We wanted to demonstrate a number of critical Visio features which had led us to choose Visio. The first and foremost capability that we wanted to show off was that the user designing the label was able to see the design while creating it. This was the issue that led the Endicia Label Server team to ask us to design this product, so we designed our demo to not only show a completed label, but also to show the process of completing a label. When we gave the demo, we started with a blank page and constructed a simplified Domestic First Class 4"x6" label.

We also wanted to demonstrate features of the product that were explicitly focused on shape creation. We wanted to show that users can save instances of custom-defined shapes onto a structure called a stencil. These shapes can then be dragged and dropped onto a label. We used this feature to create a small stencil explicitly for demos, which contained a handful of basic shapes. When all of the stencils are used, the majority of the recognisable features of a 4"x6" label are visible. We also showed how to add stencils created by another person could be inserted into each user's stencil, which allows a user's stencil set to be updated whenever needed.

Chapter 4 Tools

4.1 Visio

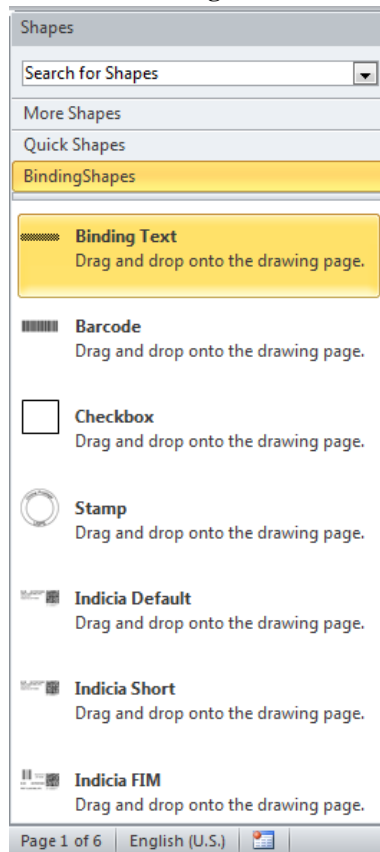
As discussed in Chapters 2 and 3, after comparing multiple layout editors, we came to the decision to work with Microsoft Visio for use in our Endicia Label Editor project. The following paragraphs are intended to reveal how we modified Visio in order to accomplish our goal.

4.1.1 Visio Stencils

One of the features that initially attracted us to Microsoft Visio was their stencils. In Microsoft Visio, a stencil (.vss file) is a tool that allows access to a collection of related shapes [62]. Within Visio, it's possible to create, save, and share custom stencils [63]. Additionally, we are able to design and create our own custom shapes. [64]

We determined that the functionality described above would enable us develop and store customized toolsets for Endicia label designers to use. We decided to develop a system for Endicia designers based off of these stencils. Our final product is shown below in the graphic (Figure 3). With this stencil, the user can drag a generic item of any of the binding shapes (such as a checkbox or text field) and then specify the width, height, binding field data, and other such properties.

Figure 3: The Endicia Binding Addresses Stencil



4.1.2 Visio Masters

In addition to the value of Visio Stencils, we liked Visio after seeing the customization of the stencil's masters. A shape or group of shapes dragged onto a stencil becomes a master. After a master is placed onto a stencil, it can be opened and edited. Masters can have any property modified, from size, text, line thickness, and, most importantly, the shape can contain pieces of information called shape data [65], similar to a variable in programming. After a master has been established, it can be used within the label design by clicking the master and dragging it onto the Visio page. This process creates an instance, which has all the properties of the master, but is not directly linked to the master and does not alter the master if modified.

Figure 4: The FromState binding field master opened in Edit Mode

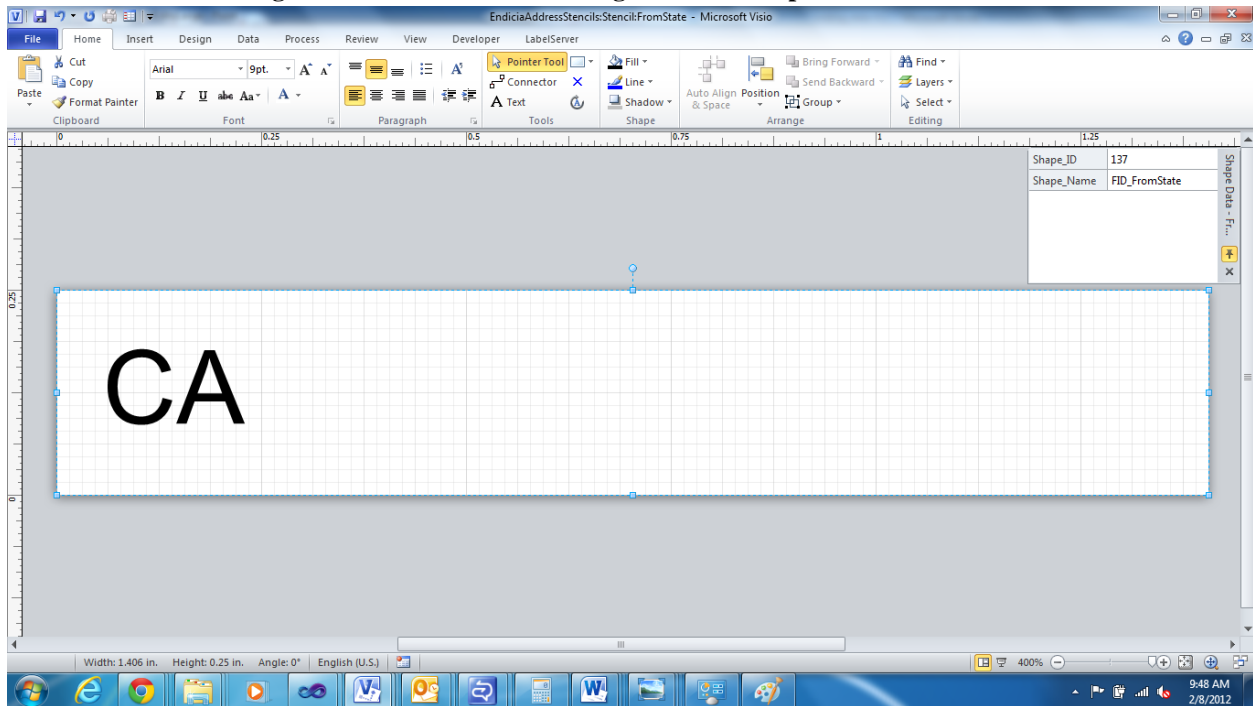


Figure 5: Shape Data for the "FromState" Binding Field

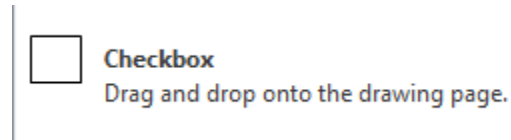
Shape_ID	137	Shape Data - Fr...
Shape_Name	FID_FromState	

Using this functionality, we were able to set up a system for determining the identity of each binding shape. Basic shapes, such as lines or rectangles, contain no data. A binding field contains two or more pieces of data which allows us to identify and instantiate the correct binding field. The two fields that are present in every binding field are Shape_ID and Shape_Name. The Shape_ID is an absolute identification of the binding field, and is directly passed into the label server. The Shape_Name is for assisting the user in identifying the type of field. Additional data fields are generated to assist in creating label fields that are not easily represented in Visio, such as checkboxes or barcodes.

We also discovered techniques for assisting the user to visually identify the information conveyed by the master. Within each master, it is possible to design a description and an icon

associated with the information contained within. In the example (Figure 6), the checkbox stencil is denoted with a simple rectangle, assisting in its identification as a checkbox.

Figure 6: The Checkbox Master as seen in the BindingShapes Stencil



4.1.3 Visio XML Schema

Microsoft Visio is capable of producing its output in different formats, such as SVG or XML. The Visio XML schema is a tool for parsing the XML outputs produced by Visio. We tried SVG as the output format from Visio; however, the SVG was complex and difficult to parse. After our advisor Patrick Farry learned about the complexity of SVG file format, he suggested that we use XML Serializer (Section 4.4) with the Visio XML Schema to easily parse the XML output from Visio.

The Visio XML Schema defines that structure of the XML output from Visio. When we used the schema along with XML Serializer, we were able to convert the XML output into programming language objects. The objects that the Visio XML file is converted to are instances of classes defined by the schema. The main class that represents a Visio document called VisioDocument_Type. Each document class composes of the attribute of the document and a list of pages that the Visio document have. Every shape that appears in a Visio page is represented as a Shape_Type class. For example, the XML segment in Appendix A is a representation of a rectangle shape in a Visio XML. The Shape_Type class is capable of containing all the information included in the example. Specifically, the Shape_Type class would contain <Geom> data which corresponds to the <Geom> tag in the XML segment above. After the XML Serializer converted the Visio XML file to objects, we were able to access all the attribute of the shape.

Ultimately, we decided that even though the Visio XML Schema provides us a good way to parse the XML file output that the Visio produce, the information contained in the XML file were more than what we needed. In the example above, to translate a rectangle shape into the Label Server's representation of a rectangle, we only need the coordinate and the dimension of the shape. We can derive this information from the <XForm> data of the Shape_Type class. All

information other than the <XForm> data are excessive to our needs. We decided not to use the Visio XML Schema for that reason.

4.1.4 Microsoft Visio Software Development Kit (SDK)

The Microsoft Visio SDK is a free software package for developers to create customizable content to integrate Visio into their solutions. Using the SDK, developers can write programs to read, manipulate, and write the contents of any Visio generated files from within a .NET framework. Additionally, the SDK provides tools to allow developers to create or modify functionality of Visio itself. [51]

There were many desirable features that the Visio SDK provided. It allowed us to write programs in all Visual Studio .NET languages to develop Visio applications [52]. Since the software development team at Endicia uses .NET framework, C#, which is a programming language used to build secure and robust applications on the .NET framework [53], was highly recommended to us by our mentor. We were able to use C# as our Visio custom application development language with no difficulty.

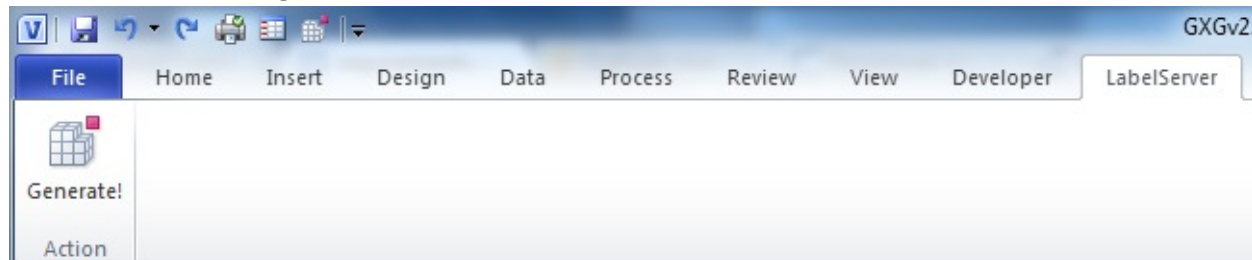
Another use of the Visio SDK was that we could read and modify the Visio objects and their members [52]. The Visio objects that we interacted with the most are Document, Page, Shape and Cell. A Document object represents a Visio format file open in an instance of Visio [54]. Every Document contains one or multiple drawing pages, which are represented by Page objects [55]. A Shape object is an item drawn on the Page. It can be a basic shape, such as a circle, or a group of items [56]. The Cell objects are used to store properties of the Shape objects. The value of each property is saved in a Cell specified by the name of the property [57]. Using the methods provided by the SDK, we were able to manipulate the properties of these objects, such as the font size of a textbox or the thickness of a line. In addition to manipulating objects already existing in a Visio page, we were also able to draw new shapes onto the page. This feature is useful for the process of converting labels in Endicia Label Server data structure to Visio documents.

Visio applications can be created as a Visio Add-On. A Visio Add-On is a software component that extends the functionality of Visio. Our Add-On application took an open Visio file, translated all the shapes on it to the data structure defined by Endicia Label Server and serialized it into a XML file which would be transported to the server. Our Add-On could be run

by clicking a custom button on the Visio user interface. The users can finish both label design and data translation in Visio environment without running additional software.

Through Visio Add-On, we were able to customize Visio by adding a new ribbon which was the tab “LabelServer” on the Visio menu. It contains a button “Generate” in it (See Figure 7). By pressing this button, the translation process will be triggered.

Figure 7 A screenshot taken on a Visio Add-On called LabelServer



These features are an excellent resource for us to use in our project. In the context of our project, the single most useful feature is the ability to read and parse Visio objects and their members, which allowed us to access the information we needed to create the label server fields. Using the Visio XML Schema and the XML Serializer, we were able to convert the data to a XML file. We used the ability to design Visio Add-ons to create a special ribbon called LabelServer, in which we placed a button to generate the XML file the server would read. Some of the functionality will come in use for designing future features as well, such as the ability to draw Visio objects into a Visio file. This functionality will allow a feature called round-tripping to be possible, where we take one of their code defined labels and convert it into a visual representation.

4.1.5 Microsoft Visio Add-In

According to the Microsoft Office Online Support, “An add-in is installed functionality that adds custom commands and new features for Microsoft Office 2010 programs.”[73] To best assist our users, we designed an Add-In that created a new Ribbon, which refers to a menu option in Microsoft Office 2010’s menu management system, to support all Label Server/label development operations. We designed the ribbon to contain as much of the possible operations a Visio label designer could want, from new functionality like creating new static shapes or binding shapes to previously existing operations like font adjustments or alignment options.

When we first began to work with the Visio Software Developer's Kit (SDK), we looked into developing an add-in as part of our functionality. Through the use of the Visio SDK, after we determined that it was possible to create our own add-in [74], we looked into developing a custom solution to issues with SVG/XML parsing where too much unnecessary data was being passed to the Label Server. This led to the creation of our own XML Schema (Section 4.2).

As we preferred our custom XML Schema over the available SVG/Visio XML schemas, our program evolved to reside primarily within the Visio Add-In. At first, it was only the basic ribbon with an export button, but as we developed more functionality, we added more options. Soon, the Label Server Ribbon had 8 distinct toolboxes, called Generation, Static Shapes, Binding Shapes, Font, Paragraph, Shape, Edit, and FID Sync.

The Generation toolbox only contains two functions; however, these two functions are the single most important feature provided by our Add-In. These two functions are Export and Import, and handle the process of generating the XML for use within the Label Server and also the round-tripping, the process of converting a data represented label from the Label Server into a Visio document.

The Static and Binding Shape toolboxes are responsible for the generation of elements for a label. While static elements are not subject to change when passed through the Label Server, binding shapes are required to update to reflect the user's request. (Section 2.5.4) The settings for instantiating the binding shapes are coded into the functions in the Binding Shape toolbox.

The Font, Paragraph, and Shape toolboxes are not custom content. They are identical to their respective toolboxes found within the Home Ribbon, but have been copied to the Label Server Ribbon for ease of use.

To allow the user some flexibility over a label that is already designed, an Edit toolbox has been provided for basic requests. With this toolbox, the user is able to eliminate margins from textboxes and change whether or not a shape possesses shape data.

The final toolbox, FID Sync, allows the user to modify the possible range of FIDs (Field Identifications) to include new IDs that do not currently exist. These FIDs are used during in binding shapes, as described in Section 2.5.4.

4.1.6 Microsoft Visio User Manual

As this product is intended to be used by Endicia employees after we have left, one of our main objectives was to make the product easy to use and intuitive. However, as even the best design may leave users confused without good documentation, we decided to create a user manual for our Add-In.

The user manual is a 20-page document that describes, in detail, all functions provided by the Label Service Add-In. It discusses all custom built functionality, all existing Visio functionality vital to the design process, and some useful sections regarding creating some settings on the labels, such as setting the label size. For all other questions regarding Visio functionality, we refer to the free online Microsoft Online Support [72]. The user manual contains an additional “Developers Only” section where it describes the structure of the code behind the Add-In and describes basic suggestions on how to develop additional functionality.

The Visio Label Service Add-In User Manual is included at the end of this report in Appendix C.

4.2 XML Serialization

Based on our Endicia advisors’ suggestion of using XML serialization, we tested XML serialization in the .NET Framework. We decided that XML serialization was a good tool for us to utilize for our project.

The primary purpose of XML serialization is to convert XML documents to programming language objects and vice versa. The process of converting objects into XML documents is called serialization. XML serialization is a technology in the .NET Framework. The class `XmlSerializer`, the central class of the `System.Xml.Serialization` [60], allowed us access to the serialization technology of the .NET framework [59]. XML serialization requires a XML schema definition (xsd). The xsd defines the association between data type information and the elements and attributes in XML documents. The xsd’s associations are bi-directional, allowing the `XmlSerializer` to perform the reverse process of serialization; this process is called deserialization.

We were first introduced to XML serialization by one of our advisors, Patrick Farry. He suggested the technology to us after learning that the SVG output from Microsoft Visio was difficult to work with. We were able to use the `XmlSerializer` in conjunction with the Visio

2010 XML Schema [61] to deserialize the XML file output from Visio. Information from the Visio XML output file was converted into instances of classes defined in the Visio XML schema, and the classes were accessible to other processes. However, we found that the information contained in the Visio XML output was excessive, and that most of the information was not necessary to be used by Label Server. We made a decision not to use XMLSerializer to convert Visio XML file.

When we decided to use Microsoft Visio SDK, we decided that we would utilize XML serialization to create XML file which contained necessary information to be transferred from Visio to Label Server. We used XMLSerializer to serialize information from objects in Visio into XML file format, and then deserialize the XML file back to objects in the Label Server. We based our decision on several reasons. First, we were able to implement the schema for our XML file without any trouble. Also the XML file format is human readable and we were able to debug our program by inspecting the XML output.

4.3 Markup Languages

When a label has been designed in a layout editor, it needs to be represented in some file format that can be safely transported to the Label Server and can be easily translated into the data structure that Label Server uses. We had four potential choices for the file format. They were XAML, SVG, XML with Visio schema and XML with our own schema. These four options are all markup languages, which are designed for structuring and transferring information in text. (Section 2.3 and Section 4.3) We compared them based on the ease of parsing and the clarity of data presentation.

XAML and SVG are the two XML-based specifications to describe graphics recommended by our mentor when our project started. After we researched layout editors that can export files in these two formats, we found out that Microsoft Expression Blend was the only well-developed XAML editor. However, Expression Blend was eliminated from the prospective layout editor list after our performance test of editors, and therefore we could not use XAML as our supported markup language. (Chapter 3 Requirements) The layout editor we chose, Microsoft Visio, supported SVG file formats. As this was one of our recommended markup languages, we attempted to parse Visio's SVG files. However, there was no Visio SVG parsing library

available, so we had to write the parser by ourselves if we were to use SVG. When we attempted to write the parser, we experienced several difficulties. First, there were a lot of overhead in SVG files which increased the time cost, requiring an additional step to remove the overhead properly before reading in the file. Second, when the Visio shapes were grouped together, their corresponding SVG elements were nested. It caused a great complexity for identifying the shapes because the tag names for different shapes are the same. Third, the properties of a shape were represented by the attributes of a SVG element instead of separate elements with their own tag names. For example, a line is defined as “<path d="M0 432 L90.56 432" class="st1"/>”. For some shapes, their attributes contain excessive information for translation. Since the semantics of the tags varies by shape, a string parsing method to screen out necessary information was needed for each shape.

When we brought up the complexity of parsing SVG in our weekly meeting with our mentors, one of our mentors Patrick Farry suggested XML serialization to us. (Section 4.3) Instead of exporting files as SVG, we exported Visio XML. Using XML serialization, we were able to convert the Visio XML output to objects defined in Visio with all the shape information and such information could be accessed by other program. This approach saved us the effort of parsing. However, the converted objects contained more data than we needed to translate into Label Server data structure and an intermediate step needed to be implemented to filter out the required information for translation. After researching more on how XML serialization and XML schemas work, we discovered that we could design our own XML semantics that matched the Label Server Data Structure. (Section 4.1.3) With Visio SDK, shape data was accessible to other programs. (Section 4.1.4) We were able to write a Visio Add-In to convert the graphic label design in Visio into our own XML format, send it to Label Server, and convert it to the exact same structure that Label Server uses to produce labels. The other advantage of using our own XML was that we could use the XML files for debugging. The XML files described the shape data extracted from Visio in human-legible pure text. By reading the XML, users were able to check the shape locations, sizes or styles that would be used to generate labels. Based on its outstanding features and the success of our test program, we chose XML with our own schema as the data exchange format.

4.4 PDF2Picture

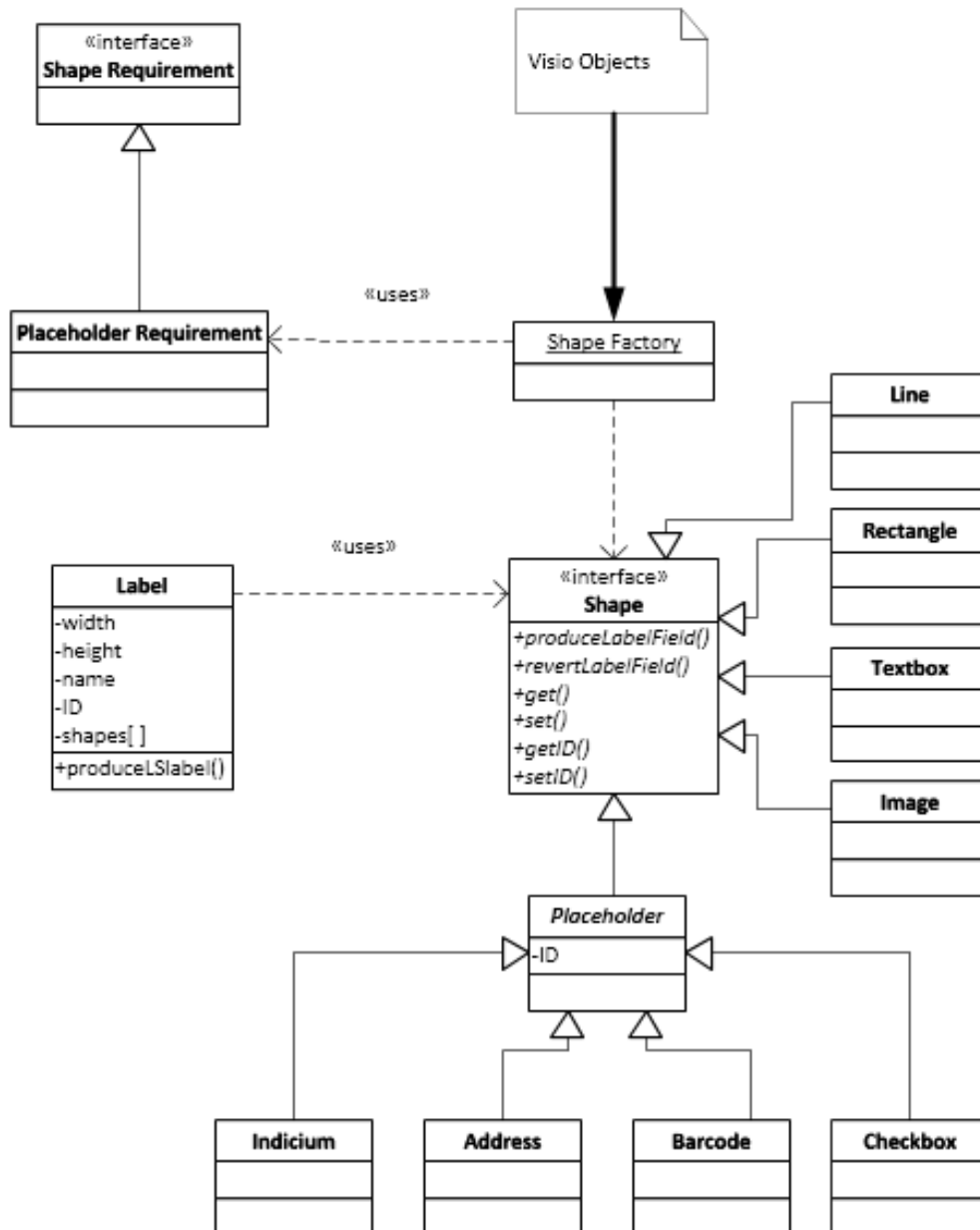
During our creation of Global Express Guarantee (GXG) label, our mentor recommended a tool called PDF2Picture for us to easily convert PDF files to Visio files. PDF2Picture is a commercial software developed by Visual Integrity, whose products focus on file format conversion between incompatible applications [67]. PDF2Picture is used for transforming PDF files to vector graphics or bitmap images that can be opened with Microsoft Windows applications, such as Microsoft Office, Microsoft Publisher and Microsoft Visio. The vector output contains shapes and text available for modification, while the bitmap output is an image of all the PDF content. PDF2Picture can be run on Windows 7 and Windows Vista. Its price is \$59 per unit [68].

We tried out PDF2Picture to convert labels in PDF and recommended it to Endicia as an additional tool for label creation. When we designed GXG in Visio, the only resource to start with was a PDF file of the label provided by U.S. Postal Service. We had to manually measure the locations and sizes of shapes and tested several font sizes and styles for text. Even though Visio was a flexible and user-friendly graphic editor, creating a complex label that looked exactly the same as the PDF version could still cost a large amount of time. Instead of drawing from scratch in Visio, the future user can run the PDF2Picture to convert the PDF label to Windows Metafile (WMF), which is a graphic file format supported on Windows platform. WMF format is also compatible with Microsoft Visio. Due to the limitation of PDF2Picture, the label is not always perfectly converted. For example, a chunk of text may break into several textboxes. This WMF label can only be treated as a draft and it needs manual modification. Using this software can still save time and effort in generating Visio labels.

Chapter 5 Program Design

5.1 Program Architecture

Figure 8 UML Diagram



One of the most difficult aspects of the Visio Add-In's task of translating a Visio layout into the Label Server is how to present the Visio Shapes. As made evident in Section 2.3, the file types that Visio naturally supports are numerous and not designed for what we required. To this extent, we developed a process (discussed in more detail in section 5.4) to convert Visio shapes into a data structure compatible with XML and the Label Server.

Figure 8 shows our structure for handling this process. We used the different Visio shapes to generate a number of different objects that all implement the Shape interface. To separate out which shape we would generate, we implemented a factory pattern.

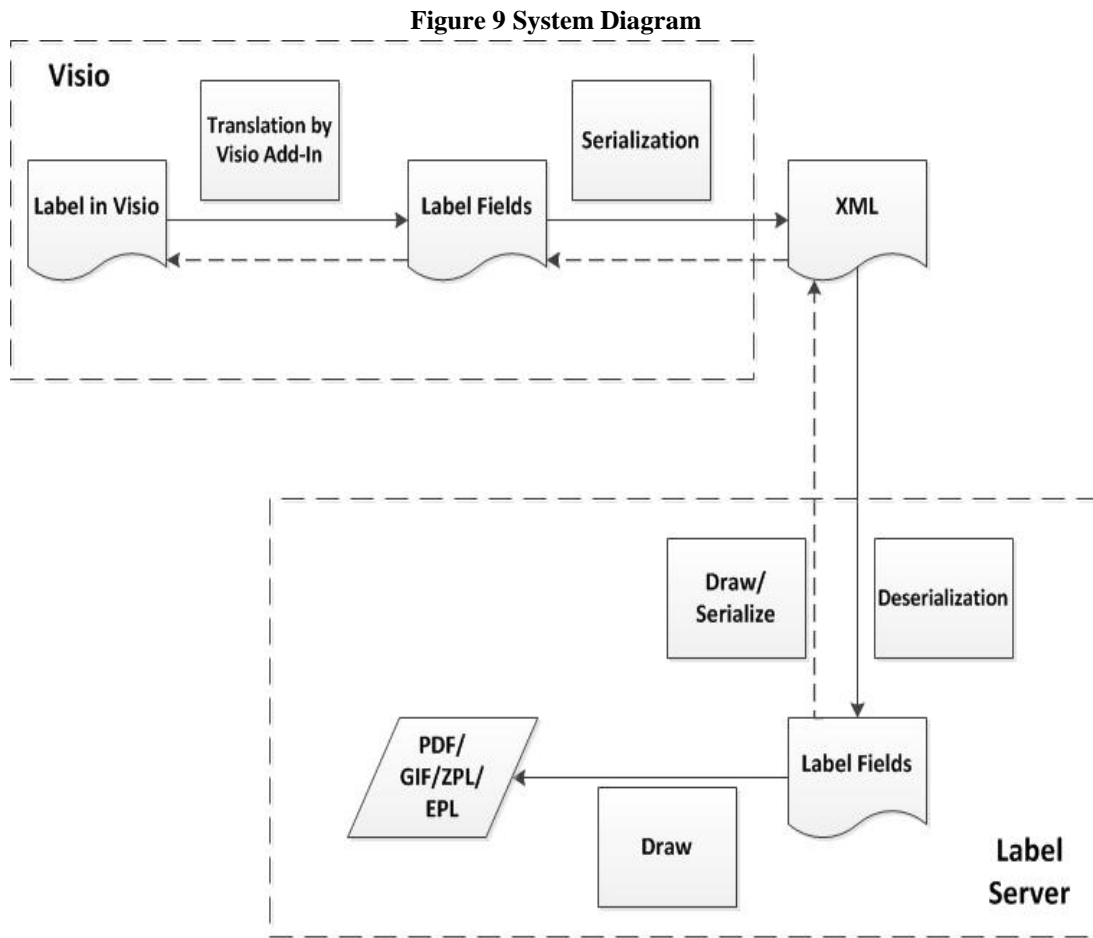
A factory pattern is a commonly used pattern intended to create flexibility in program design by creating a layer of abstraction over the creation and initialization of the product from the client [71]. This allows for the designer to create a system that would be flexible enough to implement unanticipated classes quickly. We were able to design a number of separate classes for the potential types of Visio Shapes, like barcodes, lines, and text. When reading a shape in from Visio, we would run each shape through a Requirement object, which would check the properties of an object and determine the type of shape it was. Once the shape type was known, the factory would return an object of the appropriate shape.

5.2 Program Flow

We used a system diagram to illustrate the program flow of our project. As shown in Figure 9, our program operated on two sides, client side where labels were designed in Microsoft Visio and a server side hosting the Label Server. The translation flow is represented by the solid arrows. Label design was drafted or modified graphically on the Visio side and our Visio Add-In application translated each element on the label into the corresponding Label Field, which was the data structure used in Label Server (Section 2.5.3). The Visio Add-In also serialized the Label Fields into a XML file and it was sent to the Label Server. After that, the Label Server would produce a label with the layout specified in the XML.

We also designed a round-tripping process which went the opposite direction of the forward translation. The round-tripping flow is represented by the dashed arrows. It translated a code-based label to XML and sent it to Visio application. Then Visio application would produce a graphical representation of the same label. The reason that this round-tripping process would be helpful was that Endicia had more than fifty existing labels, and we wanted to provide future

users the ability to make changes to the existing labels in Visio without needing to modify any code. However, it would take a large amount of time to create more than fifty Visio label files from scratch, so we made a tool to convert the code-based labels to Visio files. More detailed explanation about the operations on two sides is provided in Section 5.4 and Section 5.5.



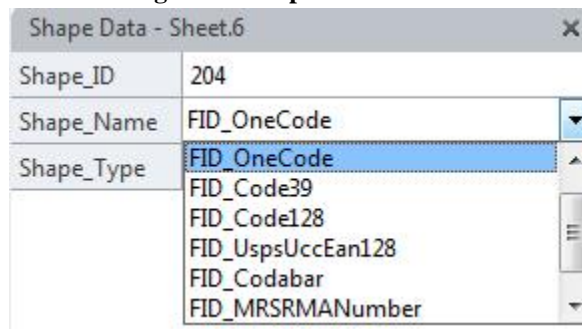
5.3 Add-In Side Operations

As mentioned in Section 5.2, our translation program operates on two sides, the Visio Add-In side and the Label Server side. On the Add-In side, users design or modify labels in Visio and convert the label into the XML format. We added several custom features to Visio to simplify and facilitate the label design process.

Visio Add-In allows users to design label layouts on a graphical interface. Users can draw static shapes, including lines, rectangles, text and pictures. They can also draw binding

shapes, which are placeholders to be filled in with actual content on the Label Server side. Binding shapes can be textboxes, checkboxes, barcodes or indicia. Every binding shape has a shape data specifying its shape type and its identifier on the Label Server. The shape data allows a binding shape to be bound with the correct content when the label is produced by the Label Server. We created a shape name combo box for each shape data to help users identify the shapes. As shown in Figure 10, this is a shape data of a barcode binding shape. To specify what type of barcode it is, users can click the combo box next to “Shape_Name” and select the FID of the barcode type they need. After that, the Shape_ID will be updated to the corresponding value. The user can also type a valid ID in the Shape_ID cell and the Shape_Name will also automatically update to the matching FID. We implemented a FID list synchronization feature to keep this FID list in our Add-In up to date with the FID list on the Label Server. Users can either add or delete an FID entry manually or import the most current FID list from an XML file. We created an additional Visio ribbon for our Add-In. It has two buttons for translation and round-tripping translation. We also added one button for each shape. When it is clicked, the corresponding shape will appear in the middle of the drawing space. We copied the font style buttons and alignment buttons from the Home ribbon to our custom ribbon, so that users do not have to navigate between two ribbons to edit labels. We picked icons that represent functions of the custom buttons the best from Microsoft default icon library.

Figure 10 Shape Data



After a label is designed in Visio, it needs to be translated into XML format to be sent to the Label Server for production. The user clicks the “Export” button on the custom ribbon to start the translation process. Our Add-In application takes the Visio label and traverses every shape on the label. For each shape, the program first checks what type it is based on a series of criteria and generates the corresponding Label Server fields with the same shape style, such as font, rotation or color. After this iteration, a list of Label Server fields is produced. The last step

on the Add-In side is to serialize these fields into XML. The XML file will be sent to the Label Server and the following operations will be explained in the next section.

5.4 Server Side Operations

When Endicia's Label Server receives the XML file which contains information about a shipping label layout, it will deserialize the XML file to obtain a list of label fields that the shipping label contains. The list of label fields is processed when the rendering engine loaded the label layouts. (See Section 2.5.2 for more information about rendering engine)

We created the process within the Label Server to handle the list of label fields obtained from the XML file. Our process traverses the list of label fields and added each label field to the label layout being loaded by the rendering engine. Some of the simpler label fields such as text field, line field or rectangle field can be added straight to the label layout. Other more complicated fields like barcode field or indicia field require a different treatment. For example, the indicia field is a complicated field; it is a combination of different text and picture fields. However, the only information that the rendering engine needed for indicia field is the location of the indicia. So for each indicia field, the location of the field is extracted by our process and then the information is passed to the rendering engine to create appropriate indicia.

After the list of label fields has been loaded to the label layout by our process, the rendering engine is able to process to its next step.

5.5 Round-tripping

As mentioned in Section 5.2, the round-tripping process is the process of translating already existing code-defined label designs into Visio's format. Since all the label layouts in the Label Server are code-defined, modifications to these layout designs are difficult. Our mentor has asked us to design the round-tripping process because modifications to shipping label design are easier using Visio.

The Label Server was able to generate shipping label to different formats. For the purpose of the round-tripping process, we added XML to the output format. The XML file produced by the Label Server has the same structure as the XML file used in the forward translation process. We were able to acquire the XML file from the Label Server and passed it to

the Visio application. The Visio application would then process the XML file to create a graphical representation.

The first step of the round-tripping process is to generate the XML file based on an existing label layout design. When we requested a shipping label in XML format, the Label Server loaded the layout of the label requested. Then the Label Server utilized XMLSerializer to serialize all the label fields contained in the label into a XML file. All the information about the label fields were transferred to the Visio application through the XML file. Also, information about the label's dimension was also serialized into the XML file.

After the XML file was passed into the Visio application, our Visio Add-in deserialized the file into a list of label fields, and the add-in also deserialized dimension information of the shipping label. When deserialization process is done, the add-in set up the dimension of the Visio drawing page to match the dimension of the shipping label. Then the add-in traversed the list of label fields and draw shapes on the drawing space based on data in each label field. For example, for the label field representing a line from point A to point B with the style dotted line and the line width of 0.01, the Visio Add-in draws a dotted line from point A to point B, and the line would have the width of 0.01.

For binding fields in the list of label field, the Visio Add-in has to not only draw their shapes but also add the shape data to those shapes to indicate that they are binding fields. Simple binding fields, such as binding text fields or binding checkboxes, only require the add-in to draw a text box or a rectangle to the drawing space. The binding fields with more complicated shapes, such as the barcode or the indicia, are drawn in the drawing space as a rectangle with text inside to indicate what type of binding field it is. After the binding field is drawn, the shape data is added to the newly drawn shape. (Section 4.1.2 discussed the shape data in more detail)

After the whole shipping label has been drawn in Visio, we are able to make modifications to the label using Visio and its tools. After modification, we can use the forward translating process to transfer the new label design to the Label Server. With the round-tripping, Endicia employees are able to modify existing label designs easier, and they are also able to convert and store all existing label in Visio format.

5.6 Maintainability

In order to promote maintainability, we have enacted several practices in our code. As our program is spread out in three different components (Visio, the Add-In, and the Label Server), setting up a system to allow efficient maintainability was mandatory. Of these three components, we determined that the Label Server and Visio would be easy to maintain. The Label Server undergoes regular maintenance (As Endicia employs an entire team of individuals dedicated to maintaining the Label Server), and although the employees at Endicia have limited control over it, the Visio software would be maintained by the Microsoft Corporation. However, the Add-In presents a small issue. While maintaining the installer is possible, it is not an easy task to enforce the user to maintain version control. Additionally, an out-of-date Add-In can cause problems in the output from Visio that can compromise the flow of the program. We decided that to best suit the user, we needed to implement some measures to prevent the user from needing to update their Add-In unless absolutely necessary.

Before we could design a solution to this problem, we needed to examine the situations in which the Add-In would need to be modified. Primarily, whenever a new binding field is introduced, the Add-In must be able to translate it. For the binding text fields, this was a simple matter- as our XML Schema had a clear definition for textfields, any text field with the appropriate shape data was set as a binding field. Other fields required more elaborate solutions to avoid updating the Add-In. For indicia, which are the electronic postage stamps, they were all constructed in the same way, but with one major difference; there were a number of different styles that determined the elements contained within each indicia. To differentiate between the different indicia without modifying the Add-In required the use of another piece of shape data. This element is called `Indicia_Type`, and is unique to the indicia shapes, which is directly passed into the Indicia label field (within the Label Server). All other shapes are managed through use of a piece of shape data called `Shape_Type`. By identifying the shape type, we are able to call the correct constructor within the Label Server.

As a result of these preventive measures, we were able to narrow down the possible situations in which the user would require the Add-In to be modified to two distinct cases. First, if a bug is found in the Add-In, the Add-In would be forced to update. Second, if the user wishes to modify or create a new type of binding field in the Label Server, the new type would not be supported by the Add-In and would need to be resolved before production. For example, if the

user were to create a TriangleField within the Label Server, the Add-In would not recognize the type “Triangle,” and depending on the implementation, could fail to translate the triangle over. However, this second case is a rare situation and the appropriate documentation is provided on how to handle it, as shown in Appendix C.

For a clearer understanding of this process, examine this example. An Endicia employee is tasked with the creation of a new label. This new label possesses many fields that have been previously generated, such as address fields or barcodes, but also possesses a few new features as well. The first scenario is a new text field. The process for generating a new text field is two parts- First, a (potentially) different Endicia employee familiar with the Label Server must go into the code and define a new Field Identification number (FID number for short). Then, based upon that number, the Endicia employee modifies the Update function inside of the file used to store the new label. The code is modified to add a command that searches the label for a field that possesses the FID specified, and then replaces its information with the user’s specification. As demonstrated here, the process of adding a new type text field requires slight updates to Visio and the Label Server, in neither case avoidable, but it avoids requiring an update to the Add-In, and the user does not have to update their program.

5.7 Testing

To prove that our process from converting a visual label to the server behaved in the expected manner, we were asked to engineer an actual label in Visio, export it into the Label Server, and then finally generate the label through a testing client that Endicia possesses. By coincidence, a new label called the Global Express Guaranteed (abbreviated as GXG) was due to be released at the end of February. Our mentor, Tiberiu Motoc, asked us to create the GXG as a way to test the capabilities of both Visio as a label editor and our translation software for comprehensiveness. A voided copy of the GXG as generated by our label editor can be found in Appendix B.

The GXG is the fastest international shipping service offered by the United States Postal Service (USPS), transported and delivered by FedEx [69]. The service is intended for shipping items capable of fitting in a custom GXG envelope, although any package that is smaller than 46" long by 46" high by 35" wide is able to be shipped [70]. The GXG label itself is 8.5" x 11" long, the size of a standard piece of printer paper, and is designed to be printed on a regular

printer (As opposed to one of DYMO's specialty label printers), and consists of 6 pages, which consist of copies for the sender, FedEx, Customs, the destination post office, the receiver, and USPS. These pages all consist of an array of address fields and customs fields, with additional features such as an indicia and/or barcode. This label is significantly more complicated than the average Endicia label, and would have been a large undertaking if designed with the previous system of label design. We successfully created the label in Visio and were able to export it to the Label Server.

Throughout the design of our project, we used the GXG as a way to test our program's functionality. Whenever a new feature was created in the Add-In, we would apply its effects to the GXG. Then, we would run Endicia's testing client, an in-house application used explicitly for testing the results of label generation through the Label Server. The testing client allowed us to manipulate all potential pieces of information that could be included in a label, from address fields to customs forms. We created settings to test our Visio generated labels, and then began to incrementally build the project, at each step comparing it to the results of the testing client.

As the creation of the round-tripping functionality was created after the development of the aforementioned generation process, the round-tripping testing took place based upon the results of our prior testing. We focused primarily on generating Visio labels from the Label Server. Once we were able to see all of the elements passed into Visio, we ensured the round-tripping process was completed by passing the round-tripped labels back into the label server.

Additionally, once we were done with the processing part, the process and the GXG label we created was passed to Quality Assurance (QA) for testing. As the GXG was needed in production ASAP, it was run through QA first. After several rounds of revisions where QA found several errors relating to placement, punctuation, and accuracy, the GXG was approved and released into production. Due to the limited time that we were at Endicia for, however, we did not get to see the results of QA's feedback.

Chapter 7 Results and Conclusion

During the project, we were able to develop a module that allows Endicia employees to design or modify shipping labels in a drawing program and produce the label with Endicia Label Server (ELS). This project will greatly enhance the efficiency of label design and not require programming skills.

We also successfully designed and generated a new shipping label called Global Express Guaranteed using our program. This label was released to Endicia customers on February 28th, 2012.

At the beginning of our project, we picked Microsoft Visio as the graphical editor and XML as the label layout format. We made this decision based on user requirements and presented our proof-of-concept to our mentor. After our approach was approved, we customized Visio user interface, implemented a Visio Add-In application to translate label layout to ELS data structure.

The solution we built will serve as internal software and make the label editing process much easier. Future work is needed in order to make this solution available to the general public. The customers should be given certain flexibility in designing their own labels under the condition that they follow the specification by US Postal Service (USPS). To ensure that custom labels are legitimate, a validation feature needs to be incorporated into our Add-In application. It should check every element on a label and warn the customer if the design breaks any rule by USPS.

Another possible future work is to support table shape. By the time we finished our project, tables on the labels were separate line and textbox shapes. A built-in table shape will simplify the design process and the data binding of table content. Table shape needs to be defined in both Add-In and ELS.

During our project work, we applied skills and knowledge we learned from computer classes, such as software engineering and design pattern. We also had the real-world experience working with a Product Manager and Quality Assurance. We not only practiced our technical

skills, but also learned how to be team players and good employees. This project will benefit us in our future careers.

References

1. Endicia, (Accessed 11/26/2011). *Company Information*. Retrieved from <http://www.endicia.com/CompanyInformation/History/>
2. Endicia, (Accessed 11/26/2011). *Endicia Label Server*. Retrieved from Server <http://www.endicia.com/Developers/LabelServer/>
3. Endicia, (Accessed 11/28/2011).. *Endicia 25 lb Postal USB Scale (ES2500U)*. Retrieved from <http://www.endicia.com/Store/Detail/?ID=82>
4. Endicia, (Accessed 11/28/2011). *Endicia 75 lb Postal USB Scale (ES2500U)*. Retrieved from <http://www.endicia.com/Store/Detail/?ID=327>
5. Endicia, (Accessed 11/27/2011).. *Domestic Mail*. Retrieved from <http://www.endicia.com/Features/DomesticMail/>
6. Endicia, , (Accessed 11/27/2011). *Strategic Partners*. <http://www.endicia.com/Developers/StrategicPartners/>
7. Newell Rubbermaid, (Accessed 11/27/2011). *All Brands*. <http://www.newellrubbermaid.com/public/Our-Brands/All-Brands.aspx>
8. DYMO, (Accessed 12/12/11). *DYMO* <http://sites.dymo.com/Pages/home.aspx?locale=enUS>
9. DYMO, (Accessed 12/12/11). *Who is DYMO?* http://sites.dymo.com/AboutDymo/Pages/AboutDymo_Landing.aspx
10. Newell Rubbermaid, (Accessed 12/12/11). *DYMO*. <http://www.newellrubbermaid.com/public/Our-Brands/Office-Products-/Dymo.aspx>
11. T. Motoc, (Personal communication, 11/29/11)
12. Adobe, (Accessed 12/07/11). *Fireworks CS5/Features*. <http://www.adobe.com/products/fireworks/features.html>
13. Adobe, (Accessed 12/07/11). *Fireworks CS5/Tech Specs*. <http://www.adobe.com/products/fireworks/tech-specs.html>
14. Adobe, (Accessed 12/07/11). *Scripting*. http://help.adobe.com/en_US/fireworks/cs/using/WSE1FCFAB6-8C6A-4164-8392-EFF705BBC6
15. Aviary, (Accessed 12/07/11). *Raven Vector Editor*. <http://www.aviary.com/online/vector-editor#>
16. Inkscape, (Accessed 12/07/11). *Official Release Packages*. <http://inkscape.org/download/?lang=en>
17. Inkscape, (Accessed 12/07/11). *FAQ*. <http://wiki.inkscape.org/wiki/index.php/FAQ>
18. Microsoft, (Accessed 12/07/11). *Products/Visio*. <http://office.microsoft.com/en-us/visio/visio-2010-buy-page-FX101836377.aspx>
19. Omnigroup, (Accessed 12/07/11). *Omnigraffle*. <http://www.omnigroup.com/products/omnigraffle/>
20. Omnigroup, (Accessed 12/07/11). *Feature Comparison*. http://www.omnigroup.com/products/omnigraffle/feature_comparison/
21. Omnigroup, (Accessed 12/07/11). *Omnigraffle Standard Version*. <http://www.omnigroup.com/products/omnigraffle/features/>
22. Omnigroup, (Accessed 12/07/11). *Extras for Omnigraffle*. <http://www.omnigroup.com/products/omnigraffle/extras/>

23. sK1 Project, (Accessed 12/07/11). *sK1 Illustration Program*.
<http://sk1project.org/modules.php?name=Products&product=sk1>
24. sK1 Project, (Accessed 12/07/11). *sK1 Download Page*.
<http://sk1project.org/modules.php?name=Products&product=sk1&op=download52.html>
25. Kiyut, (Accessed 12/07/11). *Sketsa SVG Editor download*.
<http://www.kiyut.com/products/sketsa/download.html>
26. Kiyut, (Accessed 12/07/11). *Sketsa SVG Editor Features*.
<http://www.kiyut.com/products/sketsa/features.html>
27. Microsoft, (Accessed 12/1/11). *XAML Overview (WPF)*. <http://msdn.microsoft.com/en-us/library/ms752059.aspx>
28. Scriptol, (Accessed 12/1/11). *XAML, language for building graphical user interfaces*.
<http://www.scriptol.com/programming/xaml.php>
29. W3C, (Accessed 11/28/2011). About SVG. Retrieved from <http://www.w3.org/Graphics/SVG/About.html>
30. Adobe, (Accessed 11/28/2011). SVG zone-Overview. Retrieved from
<http://www.adobe.com/svg/overview.html>
31. W3Schools, (Accessed 11/28/2011). Introduction to SVG. Retrieved from
http://www.w3schools.com/svg/svg_intro.asp
32. Inkscape, (Accessed 11/28/2011). About Inkscape. Retrieved from <http://inkscape.org/>
33. SVG-Edit, (Accessed 11/28/2011). SVG-Edit. Retrieved from <http://code.google.com/p/svg-edit/>
34. Kiyut, (Accessed 11/28/2011). Sketsa SVG Editor. Retrieved from
<http://www.kiyut.com/products/sketsa/index.html#.TtV9uWNFuso>
35. Kiyut, (Accessed 11/28/2011). Kiyut Homepage. Retrieved from <http://www.kiyut.com/index.html>
36. W3C, (Accessed 11/28/2011). W3C SVG-History. Retrieved from
<http://www.w3.org/Graphics/SVG/History.htm8>
37. W3C, (Accessed 11/28/2011). W3C Homepage. Retrieved from <http://www.w3.org/>
38. IRT.org, (Accessed 11/28/2011). SVG Brings Fast Vector Graphics to Web. Retrieved from
<http://www.irt.org/articles/js176/#3>
39. W3Schools (Accessed 2/3/12), XML Introduction, http://www.w3schools.com/xml/xml_whatIs.asp
40. W3C (Accessed 2/3/12), Extensible Markup Language (XML) 1.0 <http://www.w3.org/TR/REC-xml/>
41. W3C (Accessed 2/3/12), Extensible Markup Language (XML) 1.0 Terminology
<http://www.w3.org/TR/REC-xml/#sec-terminology>
42. W3C (Accessed 2/3/12), XML Attributes http://www.w3schools.com/xml/xml_attributes.asp
43. W3C (Accessed 2/4/12), On SGML and HTML, <http://www.w3.org/TR/html4/intro/sgmltut.html>
44. SGML Users' Group (Accessed 2/4/12), A Brief History of the Development of SGML,
<http://www.sgmlsource.com/history/sgmlhist.htm>
45. O'Reilly Media (Accessed 2/4/12), A technical Introduction to XML,
<http://www.xml.com/pub/a/98/10/guide0.html?page=2#AEN58>
46. W3Schools (Accessed 2/5/12), How Can XML be Used?, http://www.w3schools.com/xml/xml_usedfor.asp
47. Microsoft, (Accessed 1/31/12), Create, save and share a custom stencil
<http://office.microsoft.com/en-us/visio-help/create-save-and-share-a-custom-stencil-HA101782586.aspx?CTT=1>

48. Microsoft, (Accessed 1/31/12), Customizing the Ribbon
<http://msdn.microsoft.com/en-us/library/gg617997.aspx>
49. Microsoft, (Accessed 1/31/12), Download free customized ribbons,
<http://office.microsoft.com/en-us/visio-help/download-free-customized-ribbons-HA101885823.aspx?CTT=1>
50. Syncfusion, (Accessed 1/31/12), Create Ribbon using MS Expression Blend,
http://help.syncfusion.com/ug_84/User%20Interface/Silverlight/Tools/default.htm?url=Documents%2Fcreatingribbonusingmsexpressionblend.htm
51. Microsoft, (Accessed 1/25/12). *Visio 2010*.
<http://www.microsoft.com/download/en/details.aspx?id=12365>
52. Microsoft, (Accessed 1/25/12). *Visio 2010 Automation References*.
<http://msdn.microsoft.com/en-us/library/ff768441.aspx>
53. Microsoft, (Accessed 1/30/12). Introduction to the C# Language and the .NET framework.
<http://msdn.microsoft.com/library/z1zx9t92>
54. Microsoft, (Accessed 1/30/12). Document Object (Visio)
<http://msdn.microsoft.com/en-us/library/ff765575.aspx>
55. Microsoft, (Accessed 1/30/12). Page Object (Visio)
<http://msdn.microsoft.com/en-us/library/ff767035.aspx>
56. Microsoft, (Accessed 1/30/12). Shape Object (Visio)
<http://msdn.microsoft.com/en-us/library/ff768546.aspx>
57. Microsoft, (Accessed 1/30/12). Cell Object (Visio)
<http://msdn.microsoft.com/en-us/library/ff765137.aspx>
58. Microsoft, (Accessed 1/30/12). About Microsoft Office Visio Add-ons and COM Add-ins
[http://msdn.microsoft.com/en-us/library/aa168138\(v=office.11\).aspx](http://msdn.microsoft.com/en-us/library/aa168138(v=office.11).aspx)
59. Microsoft, (Accessed 1/26/12). *XML Serialization in the .NET Framework*
<http://msdn.microsoft.com/en-us/library/ms950721.aspx>
60. Microsoft, (Accessed 1/26/12). *System.Xml.Serialization namespace*
<http://msdn.microsoft.com/en-us/library/e123c76w.aspx>
61. Microsoft, (Accessed 1/26/12). *Welcome to the Visio 2010 XML Schema Reference*
<http://msdn.microsoft.com/en-us/library/ff768724.aspx>
62. Microsoft (Accessed 2/07/12). *Use the Shapes window to organize and find shapes*.
<http://office.microsoft.com/en-us/visio-help/use-the-shapes-window-to-organize-and-find-shapes-HA010369721.aspx?CTT=5&origin=HA101782586>
63. Microsoft (Accessed 2/07/12). *Create, save, and share a custom stencil*. <http://office.microsoft.com/en-us/visio-help/create-save-and-share-a-custom-stencil-HA101782586.aspx?CTT=1>
64. Microsoft, (Accessed 2/07/12). *Add a shape to a stencil*. <http://office.microsoft.com/en-us/visio-help/add-a-shape-to-a-stencil-HP001231185.aspx?CTT=1>
65. Microsoft, (Accessed 2/07/12). *Add data to shapes*. <http://office.microsoft.com/en-us/visio-help/add-data-to-shapes-HA101791941.aspx>
66. EPL Programming Guide. Retrieved February 10, 2012,
http://www.servopack.de/support/zebra/EPL2_Manual.pdf
67. Visual Integrity, (Accessed 2/17/2012), About Us,
<http://www.visual-integrity.com/about.htm>
68. PDF2Picture, (Accessed 2/17/2012), Change PDF files into editable graphics,
<http://www.pdf2picture.com/>
69. United States Postal Service, *Global Express Guaranteed*.
<https://www.usps.com/ship/gxg.htm>, Accessed 2/28/12
70. United States Postal Service, *Global Express Guaranteed Shipping Kit*.

https://store.usps.com/store/browse/productDetailSingleSku.jsp?categoryNavIds=catGetMailingShippingSupplies%3asubcatMSS_International&categoryNav=false&navAction=push&navCount=0&productId=P_GXGKIT&categoryId=subcatMSS_International. Accessed 2/28/12.

71. Gopalan Suresh Raj, *The Factory Method (Creational) Design Pattern*. <http://gsraj.tripod.com/design/creational/factory/factory.html>. Accessed 2/28/12.
72. Microsoft Office, *Visio Help and How-to*. <http://office.microsoft.com/en-us/visio-help/>. Accessed 2/28/12.
73. Microsoft Office, *View, manage and install add-ins in office programs*. <http://office.microsoft.com/en-us/access-help/view-manage-and-install-add-ins-in-office-programs-HA010354315.aspx?CTT=1>. Accessed 2/28/12.
74. MSDN, *Customizing the Ribbon in Visio 2010 by using a Visual Studio 2010 add-in*. <http://msdn.microsoft.com/en-us/library/gg617997.aspx>. Accessed 2/28/12.
75. Joe Baca. (n.d.). BrainyQuote.com. Retrieved February 28, 2012, from BrainyQuote.com Web site: <http://www.brainyquote.com/quotes/quotes/j/joebaca295461.html>

Appendix A

Visio XML File

A shape in the Visio XML file is represented as followed.

```
<Shapes>
  <Shape LineStyle="3" FillStyle="3" TextStyle="3" ID="1" Type="Shape">
    <XForm>
      <PinX>3.375</PinX>
      <PinY>7.5</PinY>
      <Width>3.5</Width>
      <Height>2</Height>
      <LocPinX F="Width*0.5">1.75</LocPinX>
      <LocPinY F="Height*0.5">1</LocPinY>
      <Angle>0</Angle>
      <FlipX>0</FlipX>
      <FlipY>0</FlipY>
      <ResizeMode>0</ResizeMode>
    </XForm>
    <Geom IX="0">
      <NoFill>0</NoFill>
      <NoLine>0</NoLine>
      <NoShow>0</NoShow>
      <NoSnap>0</NoSnap>
      <MoveTo IX="1">
        <X F="Width*0">0</X>
        <Y F="Height*0">0</Y>
      </MoveTo>
      <LineTo IX="2">
        <X F="Width*1">3.5</X>
        <Y F="Height*0">0</Y>
      </LineTo>
      <LineTo IX="3">
        <X F="Width*1">3.5</X>
        <Y F="Height*1">2</Y>
      </LineTo>
      <LineTo IX="4">
        <X F="Width*0">0</X>
        <Y F="Height*1">2</Y>
      </LineTo>
      <LineTo IX="5">
        <X F="Geometry1.X1">0</X>
        <Y F="Geometry1.Y1">0</Y>
      </LineTo>
    </Geom>
  </Shape>
</Shapes>
```

```
</Geom>  
<v14:Geom IX="0" xmlns:v14="" >  
  <v14:NoQuickDrag>0</v14:NoQuickDrag>  
</v14:Geom>  
</Shape>
```

Appendix B

Global Express Guaranteed



International delivery
by FedEx Express



U.S. POSTAGE REQUIRED

1. From
Date: 3/5/2012

Sender's Name: John Doe Phone: (650) 321-2640
Company: Endicia
Address: 365 Sherman Ave
Address: PALO ALTO State: CA
City: PALO ALTO State: CA
Country: United States of America ZIP Code: 94306

2. To
Recipient's Name: Jane Doe Phone: 1234567890
Company: Fax: (if applicable)
Address: 1-10-5 Akasaka, Minato-ku
Address: Dept/Floor
City: Tokyo State/Province
Country: Japan ZIP Code: 107-8420

Recipient's Tax ID number for Customs purposes (if applicable)
e.g. GST/REG/VAT/IN

3. Shipment Information

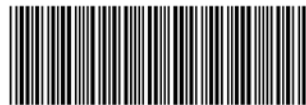
Documents: Correspondence and printed matter
 Non-Documents: All other items. Recipient may be required to pay import duties and taxes. This shipment may be subject to inspection.

Specific Description Including Number of Each Item (and harmonized code if known) REQUIRED	Country of Manufacture	Value for Customs REQUIRED
Documents	CN	\$0.01
Warning: These commodities, technology, or software were exported from the United States in accordance with Export Administration Regulations. Diversion contrary to U.S. law prohibited.		Total Value for Customs (US \$)
		\$0.01

4. Required Sender's Name
(We agree that the USPS terms and conditions (on the back of the Sender's Copy of this Air Waybill and in the Global Express Guaranteed Service Guide) and certain international treaties, including the Warsaw Convention, where applicable, apply and limit the liability of USPS and FedEx for loss and damage. I/We understand that USPS and FedEx DO NOT TRANSFER OR ACCEPT LIABILITY for this package does not contain any hazardous or restricted materials prohibited by postal regulations and does not require the filing of Electronic Export Information (EIE) (formerly Shipper's Export Declaration (SED)), and that the particulars given in the Shipment Information section are as declared. Submission of false information may result in civil or criminal penalties (18 USC 1001, 31 USC 3802). NoEEI § 30.37(a)

Sender's Signature: John Doe

Postal Use Only	Date In	Time In	Scheduled Delivery Date	PO ZIP (+4) Code	Employee Initials
		<input type="checkbox"/> AM <input type="checkbox"/> PM			
Dimensions in Inches (length, width, height) (Non-Documents Shipments)		Weight	Postage	Insurance Fee	Total Postage & Fees
0 0 0		0 lbs. 1 oz	\$ \$52.00	\$	\$ \$52.00
Packaging Customer Supplied		8983 9500 1131			
<input checked="" type="checkbox"/> OXG Envelope or Pak 1 <input type="checkbox"/> Box 1 <input type="checkbox"/> Envelope 1 <input type="checkbox"/> Other					



USPS
Tracking
Number

83 950 011 34

SENDER Copy

International Air Waybill

Sender's Copy (1 of 6)

Fold on dotted line.

Postal Customer - Instructions for completing the International Air Waybill.

1. Verify sections 1 through 4 of the Air Waybill are valid and complete (required to be eligible for the Money-Back Delivery Guarantee).
2. Fold each of the 6 copies in half and arrange in numerical order with the Sender's Copy on top.
3. Place all six folded copies in pocket of clear pouch with information showing*. - DO NOT SEAL POUCH.

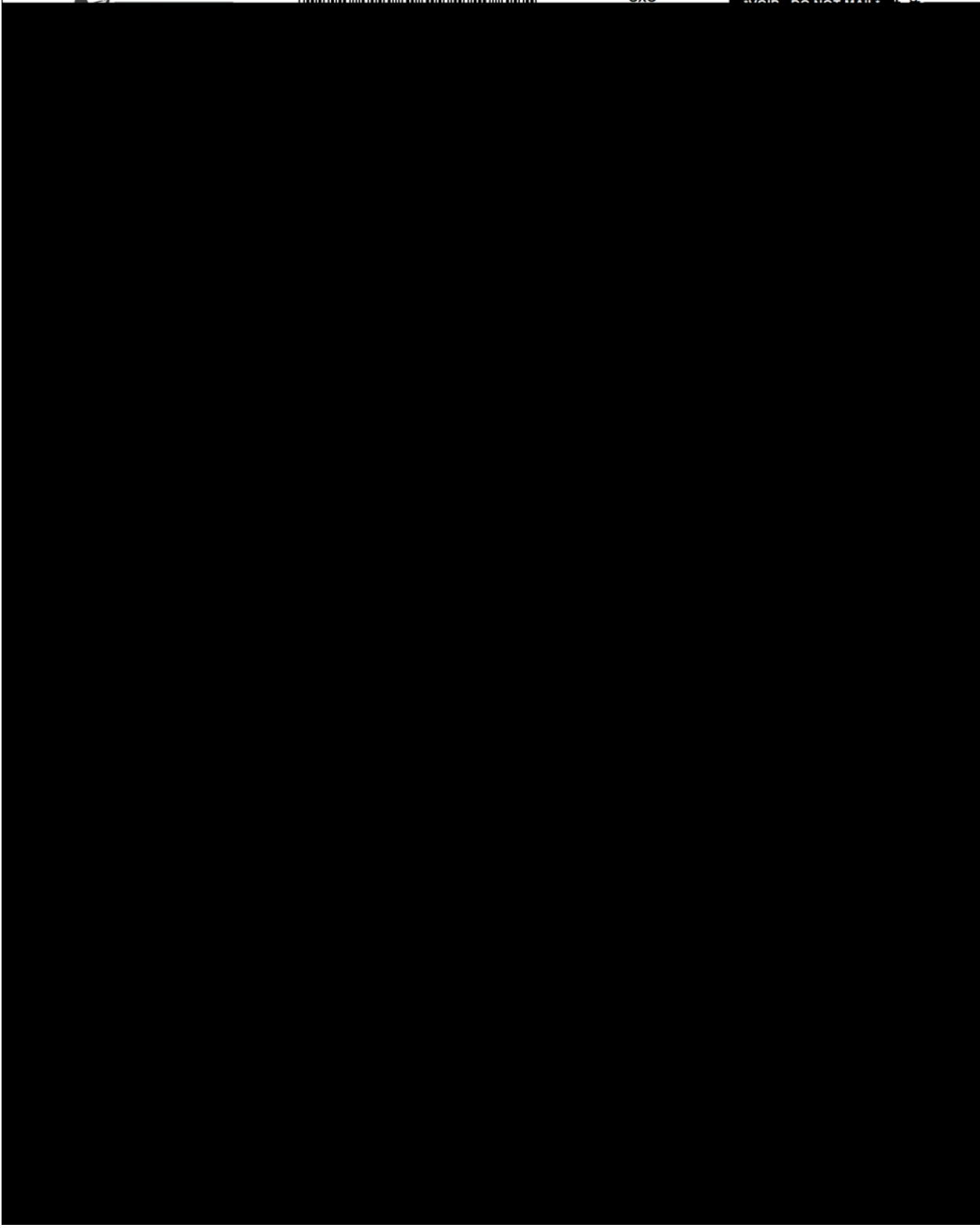
 GLOBAL EXPRESS



\$52.00 US POSTAGE
GXG



due

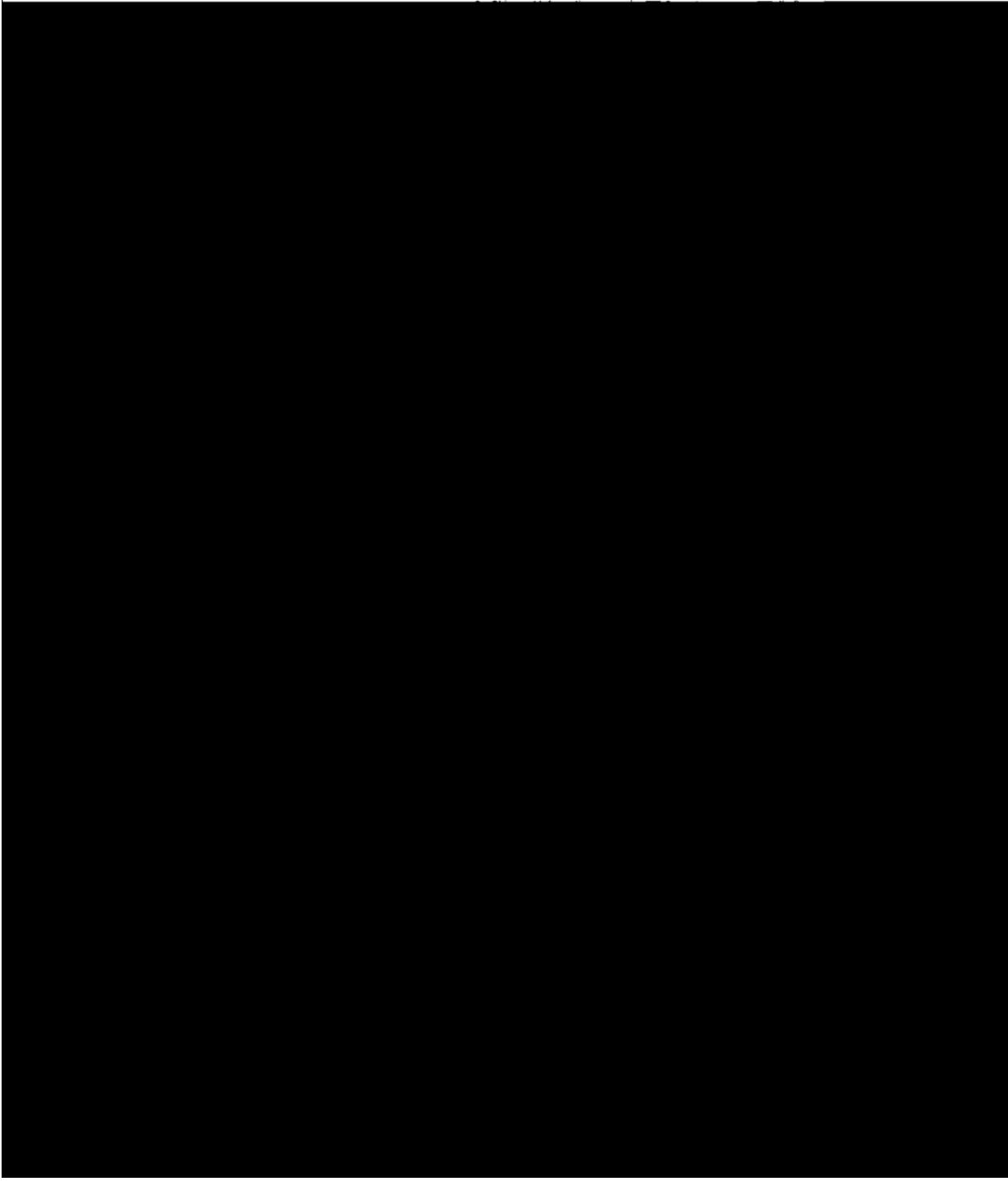




International delivery
by FedEx Express



U.S. POSTAGE REQUIRED





International delivery
by FedEx Express



U.S. POSTAGE REQUIRED

1. From
 Date 3/5/2012
 Sender's Name John Doe Phone (650) 321-2640
 Company Endicia
 Address 365 Sherman Ave
 Address
 City PALO ALTO State CA
 Country United States of America ZIP Code 94306

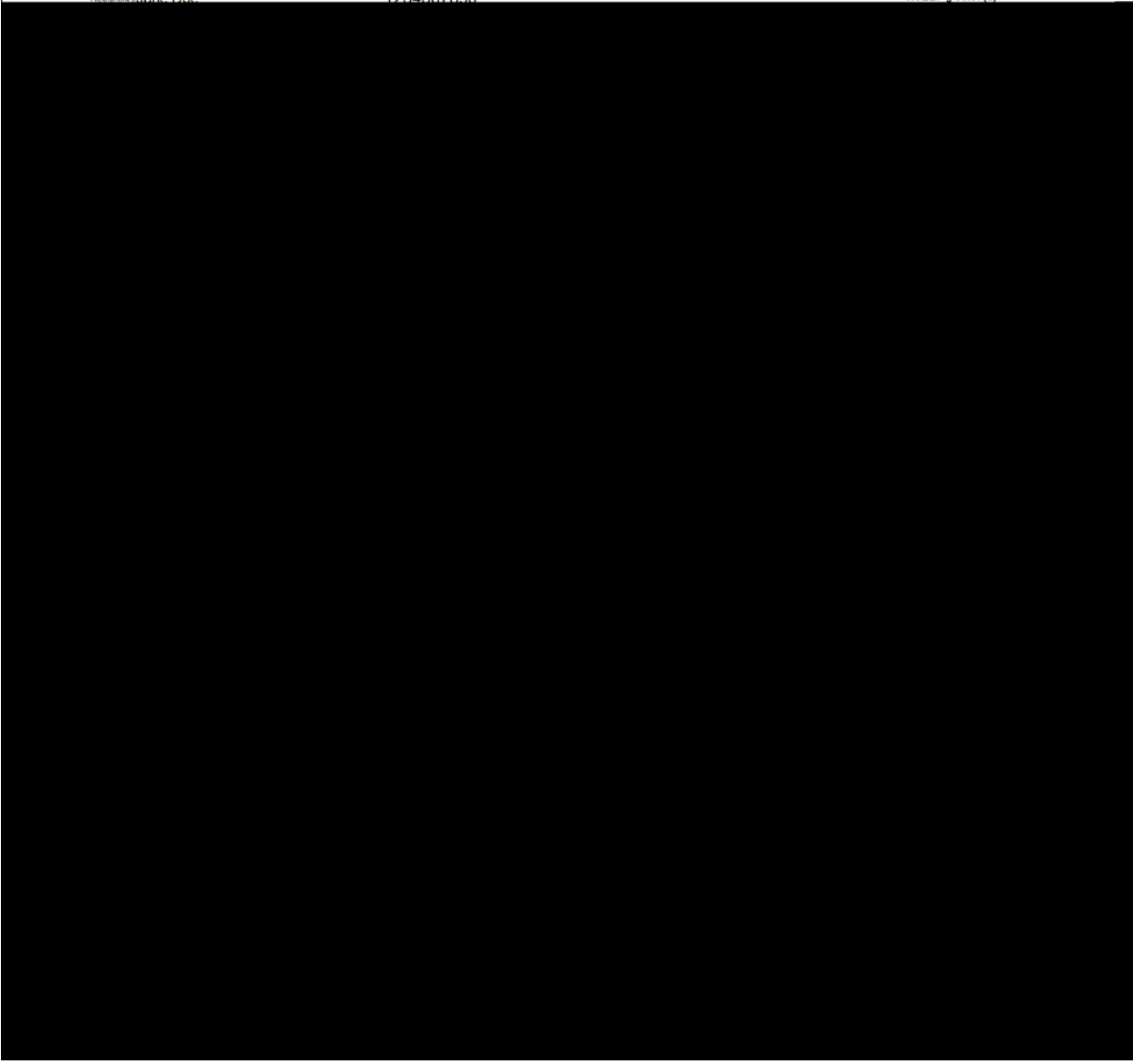
2. To
 Recipient's Name Jane Doe 1234567890

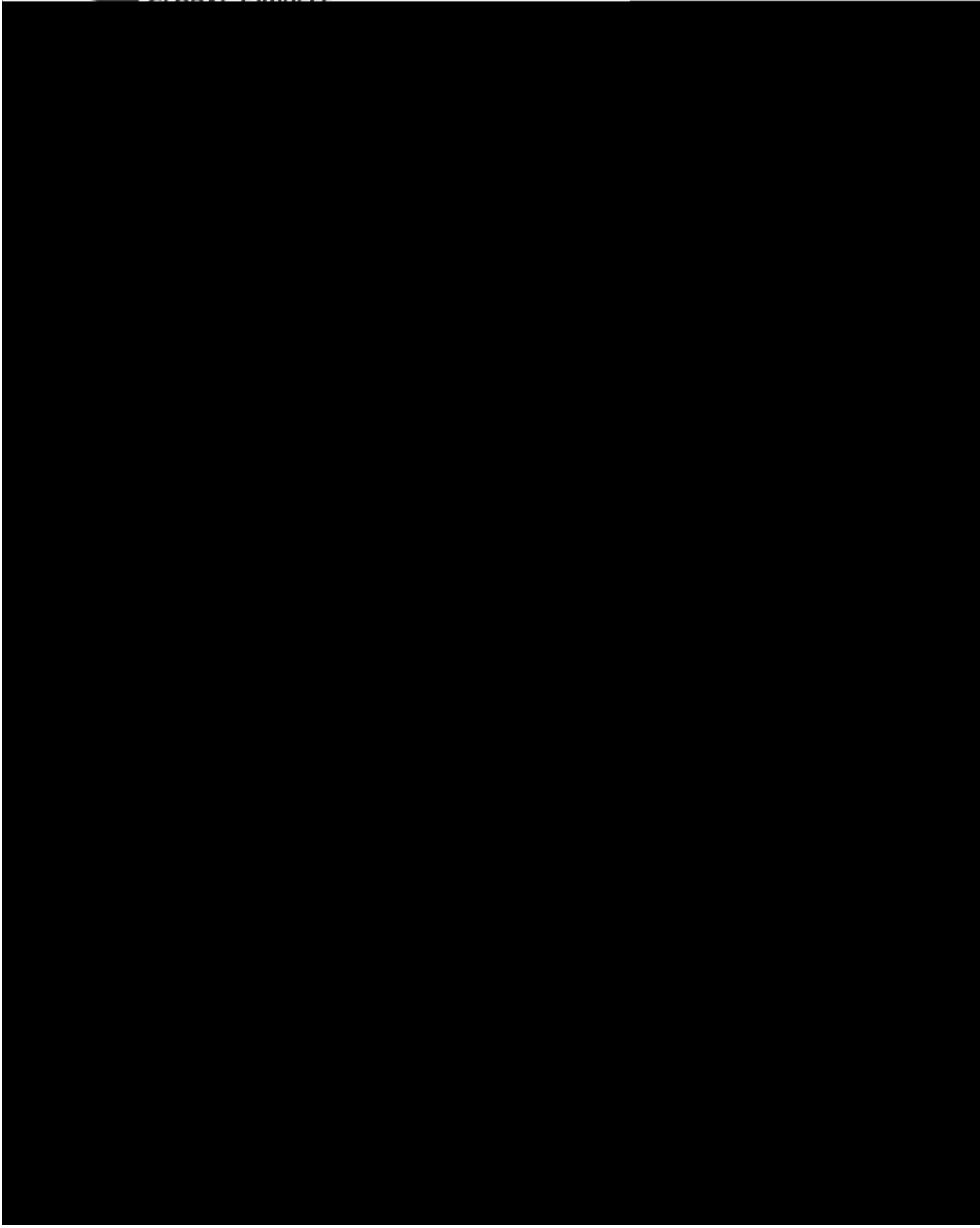
3. Shipment Information Documents: Correspondence and printed matter Non-Documents: All other items. Recipient may be required to pay import duties and taxes. This shipment may be subject to inspection.

FedEx Intl Priority

Specific Description Including Number of Each Item (and harmonized code if known) REQUIRED	Country of Manufacture	Value for Customs REQUIRED
Documents	CN	\$0.01
Warning: These commodities, technology, or software were exported from the United States in accordance with Export Administration Regulations. Deviation contrary to U.S. law prohibited.		Total Value for Customs (US \$) \$0.01

4. Required Sender's Name
 (We agree that the USPS terms and conditions (on the back of the Sender's Copy of this Air Waybill and in the Global Express Guaranteed Service Guide) and certain international treaties, including the Warsaw Convention, where applicable, apply and limit the liability of USPS and FedEx for loss and damage. We understand that USPS and FedEx DO NOT FINANCE OR CASH. (We certify that this package does not contain any hazardous or restricted materials prohibited by postal regulations and does not require the filing of Electronic Export Information (EEI) (formerly Shipper's Export Declaration (SED), and that the particulars given in the Shipment Information section are as declared. Submission of false information may result in civil or criminal penalties (18 USC 1001, 31 USC 3802) NoEEI § 30.37(a)







International delivery
by FedEx Express



U.S. POSTAGE REQUIRED

1. From
Date: 3/5/2012

Sender's Name: John Doe Phone: (650) 321-2640
Company: Endicia
Address: 365 Sherman Ave
Address:
City: PALO ALTO State: CA
Country: United States of America ZIP Code: 94306

2. To
Recipient's Name: Jane Doe Phone: 1234567890
Company: Fax (if applicable)
Address: 1-10-5 Akasaka, Minato-ku
Address: Dept/Floor
City: Tokyo State/Province:
Country: Japan ZIP Code: 107-8420

Recipient's Tax ID number for Customs purposes (if applicable)
e.g., GST/RFC/VAT/IN

For tracking, visit usps.com or call
1.800.222.1811.

3. Shipment Information Documents: Correspondence and printed matter Non-Documents: All other items. Recipient may be required to pay import duties and taxes. This shipment may be subject to inspection.

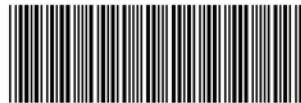
Specific Description including Number of Each Item (and harmonized code if known)	REQUIRED	Country of Manufacture	Value for Customs REQUIRED
Documents		CN	\$0.01
Total Value for Customs (US \$)			\$0.01

Warning: These commodities, technology, or software were exported from the United States in accordance with Export Administration Regulations. Diversion contrary to U.S. law prohibited.

4. Required Sender's Name
I/We agree that the USPS terms and conditions (on the back of the Sender's Copy of this Air Waybill and in the Global Express Guaranteed Service Guide) and certain international treaties, including the Warsaw Convention where applicable, apply and limit the liability of USPS and FedEx for loss and damage. I/We understand that USPS and FedEx DO NOT "PICK UP OR CASH" I/We certify that this package does not contain any hazardous or restricted materials prohibited by postal regulations and does not require the filing of Electronic Export Information (EEI) (formerly Shipper's Export Declaration/SED), and that the particulars given in the Shipment Information section are as declared. Submission of false information may result in civil or criminal penalties (18 USC 1001, 31 USC 3802) NoEEI § 30.37(a)

Sender's Name: John Doe

Postal Use Only	Date In	Time In	Scheduled Delivery Date	PO ZIP (+4) Code	Employee Initials
		<input type="checkbox"/> AM <input type="checkbox"/> PM			
Dimensions in inches (depth, width, height)	0	0	0	Weight 0 lbs. 1 oz.	Postage \$ \$52.00
				Insurance Fee \$	Total Postage & Fees \$ \$52.00
Packaging <input checked="" type="checkbox"/> G/XG Envelope or Pak 1 <input type="checkbox"/> Box 1 <input type="checkbox"/> Envelope 1 <input type="checkbox"/> Other					



USPS Tracking Number

83 950 011 34

USPS Copy

International Air Waybill

USPS Copy (6 of 6)

Fold on dotted line.

*Order Supplies online at <http://supplies.usps.gov/> by FAX at 1-800-270-6233, by phone at 1-800-610-8734 or at a Global Express Guaranteed participating post office.



International Air Waybill

International delivery
by FedEx Express

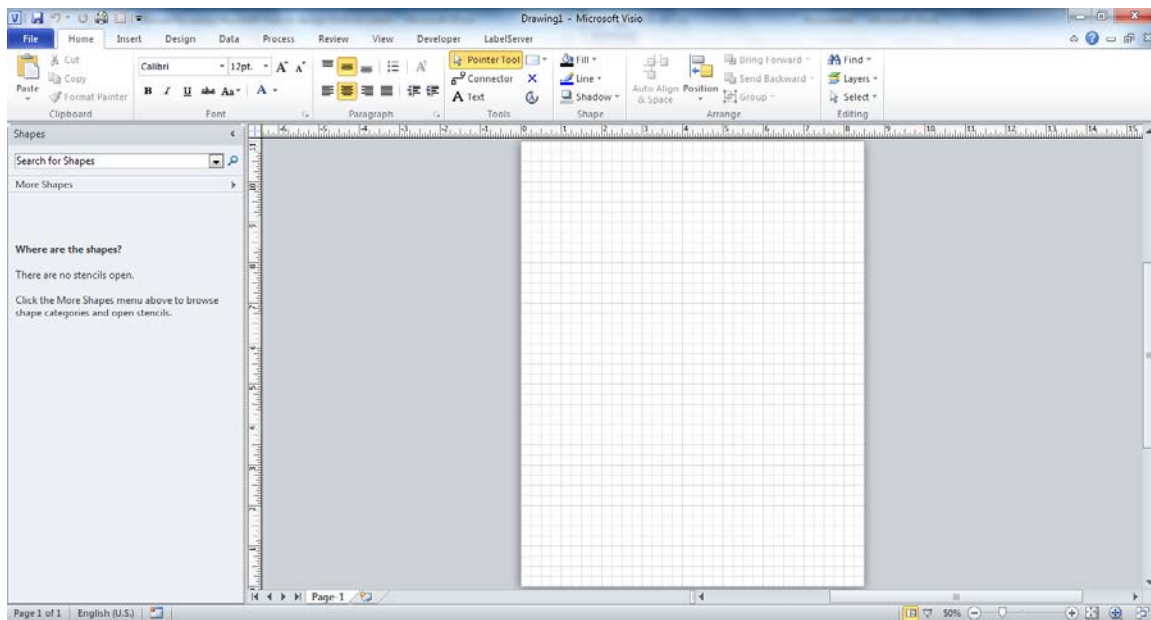


Appendix C

Visio User Manual

The process of generating labels created by the USPS for the Endicia Label Server has been greatly simplified with the use of Microsoft Visio. In this manual, we shall review the available functionality of Microsoft Visio, from the basics of label design to creating new binding fields. This documentation is intended to clarify some of the basics of Microsoft Visio and to cover all content exclusive to the Endicia Label Server Visio Add-In. It will go into detail on aspects of the Add-In and touch upon aspects of Microsoft Visio that are important when designing an Endicia Label Server label. For additional information on how to operate Microsoft Visio, visit <http://office.microsoft.com/en-us/visio-help/> to access Microsoft's free online help.

The Visio Interface

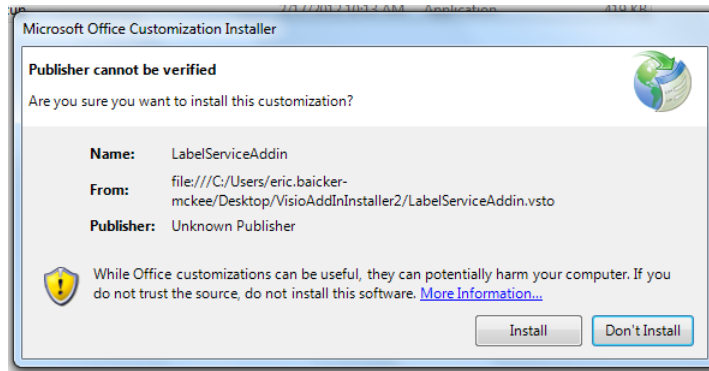


When creating Endicia labels through Microsoft Visio, it is important to apply some basic settings to enjoy the best experience.

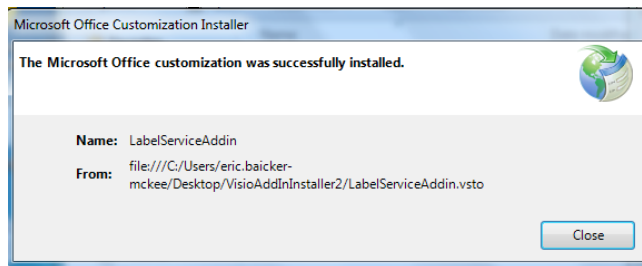
Installing the Endicia Label Service Add-In

The first and most basic step for using Microsoft Visio is to download the Label Service Add-In. This Add-In gives the user a basic toolbar for all the functionality that they will need in the course of designing a project, and contains some custom-defined content to export labels from Visio into the Endicia Label Server. The Add-In is currently supported on Windows platforms, on versions of Visio 2010 or greater.

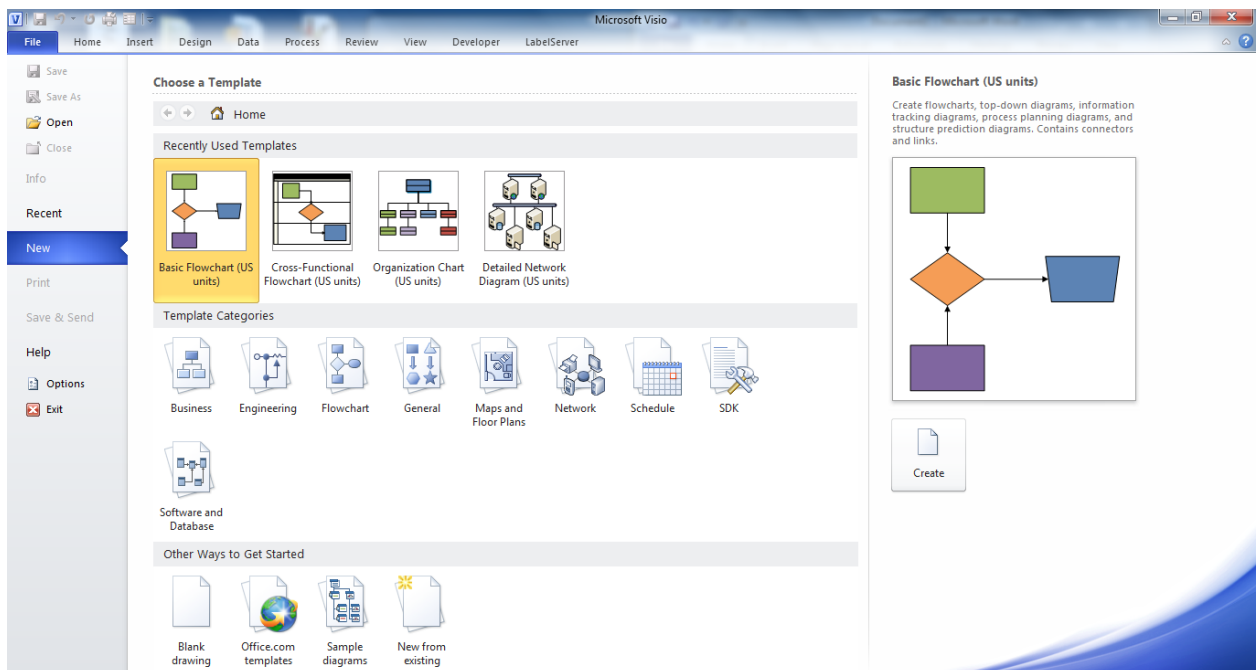
To install the Add-In, unzip the folder called "VisioAddInInstaller" and run the application called "setup". You will be prompted with a Microsoft Office Customization Installer window, confirming the installation. Select, "Install," and wait for the installer to complete running.



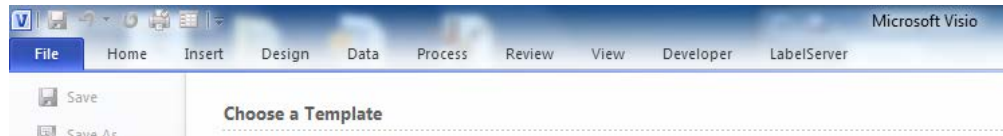
When completed, a window will appear similar as below, and the Visio Add-In will be successfully installed. If Microsoft Visio is currently running, please re-start the application to apply the Add-In content.



After opening Microsoft Visio for the first time after installing the Add-In, the window should look like this:

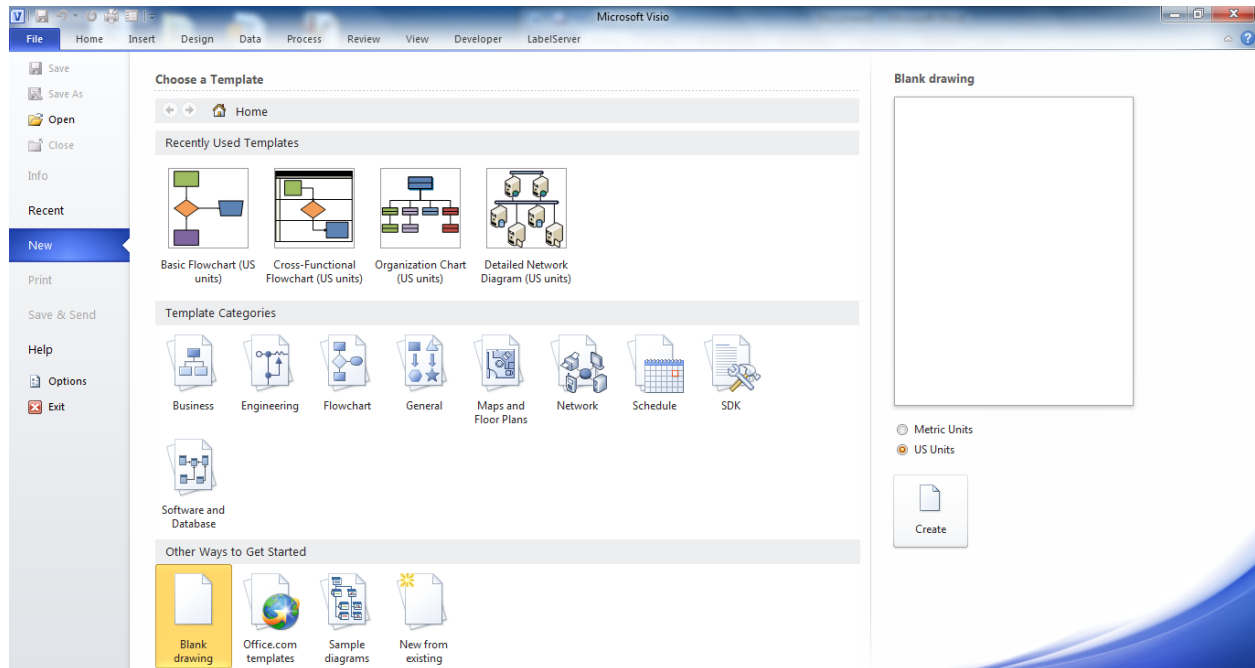


If you have used Microsoft Visio before, the largest change you can see is in the menu tabs at the top of the screen (called “Ribbons”), you can see that a new ribbon has been added called, “LabelServer.”

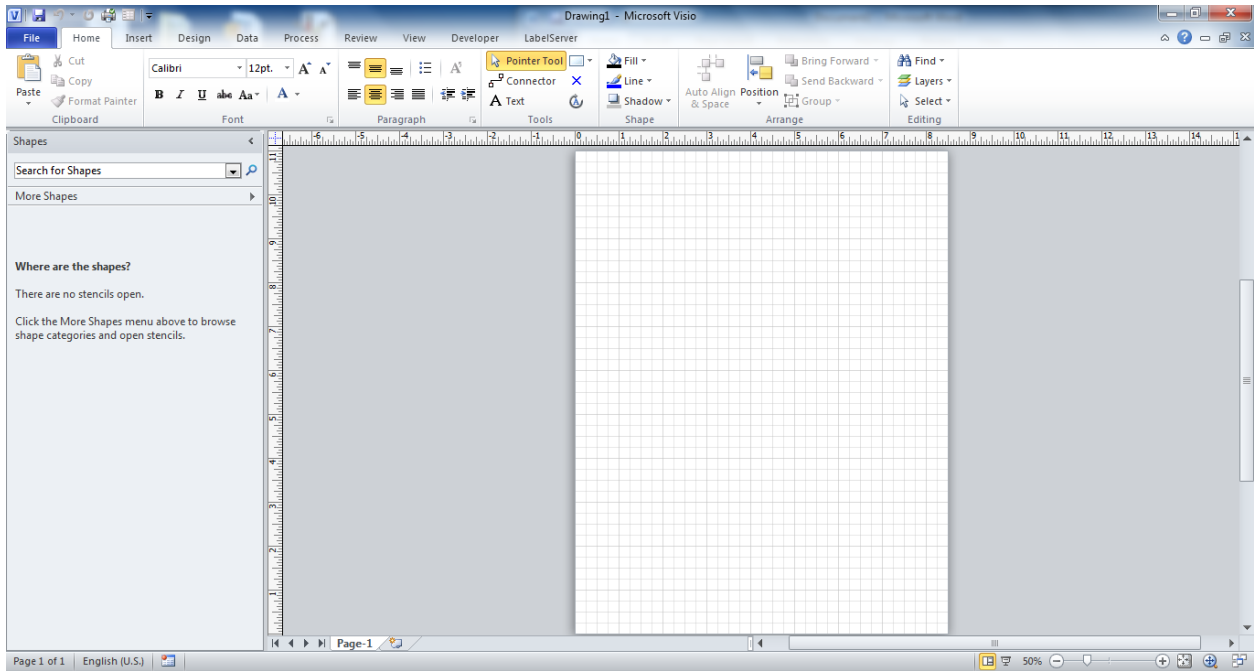


Creating a new Visio label

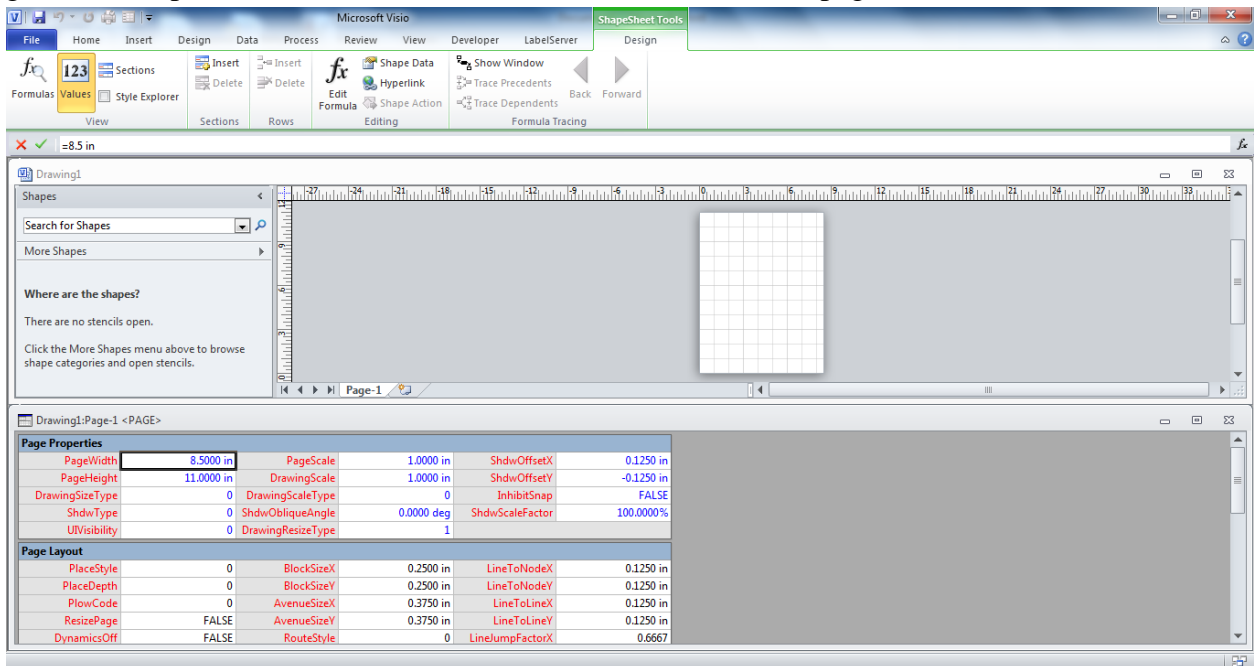
Before its possible to create an Endicia Label, the drawing area must be created. From the File ribbon, select the New tab, and select Blank drawing.



If you are working from a previously existing file, use the “Open” tool instead. Once the new “Blank drawing” file has been created, you’ll see a screen that looks like this:

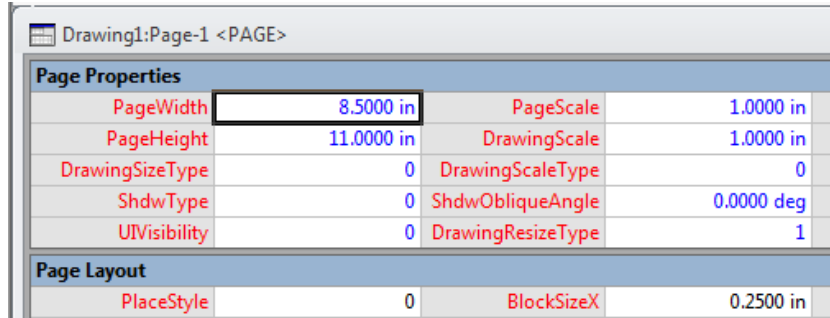


The white space in the center of the screen is the size and shape of the label to be created. The default size is 8.5” x 11”, the size of a standard piece of printer paper. This value can be modified by holding the Ctrl button and dragging the sides of the label, or, for greater precision, by right clicking on the white space and selecting the “Show Shapsheet” option. This will generate a separate window with additional information for the page.



The height and width of the page can be modified by altering the PageWidth and PageHeight values. If the width and height are modified, be sure to change the last item within the Page

Properties options, DrawingResizeType, to the number 2. This will prevent the size of your image from inappropriately resizing.

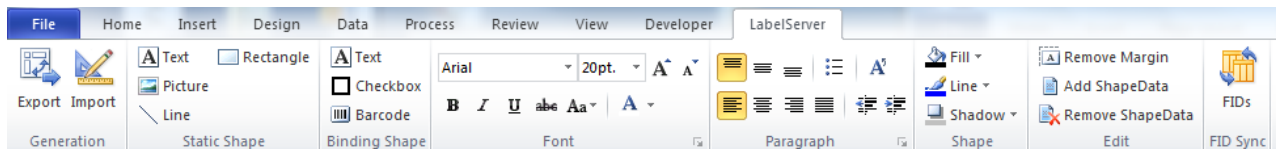


Page Properties			
PageWidth	8.5000 in	PageScale	1.0000 in
PageHeight	11.0000 in	DrawingScale	1.0000 in
DrawingSizeType	0	DrawingScaleType	0
ShdwType	0	ShdwObliqueAngle	0.0000 deg
UIVisibility	0	DrawingResizeType	1

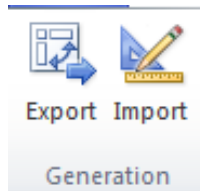
Page Layout			
PlaceStyle	0	BlockSizeX	0.2500 in

Shapesheets can be used to modify the page or specific elements within it. For more information about Shapesheets, refer to the Microsoft Office online documentation, available at <http://office.microsoft.com/en-us/visio-help/>.

The LabelServer Ribbon

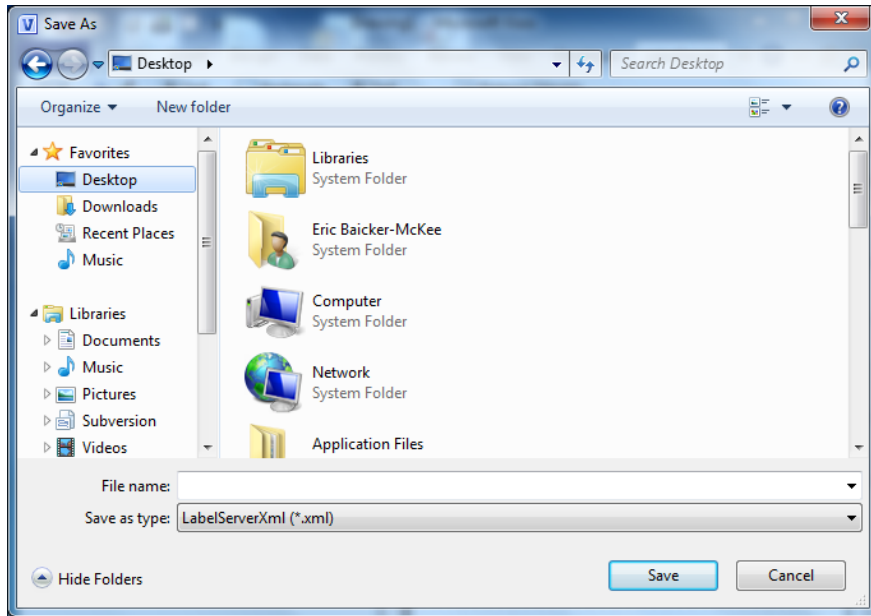


The LabelServer Ribbon contains the majority of the custom functions to generate Endicia Labels, and requires a separate installation from Visio.



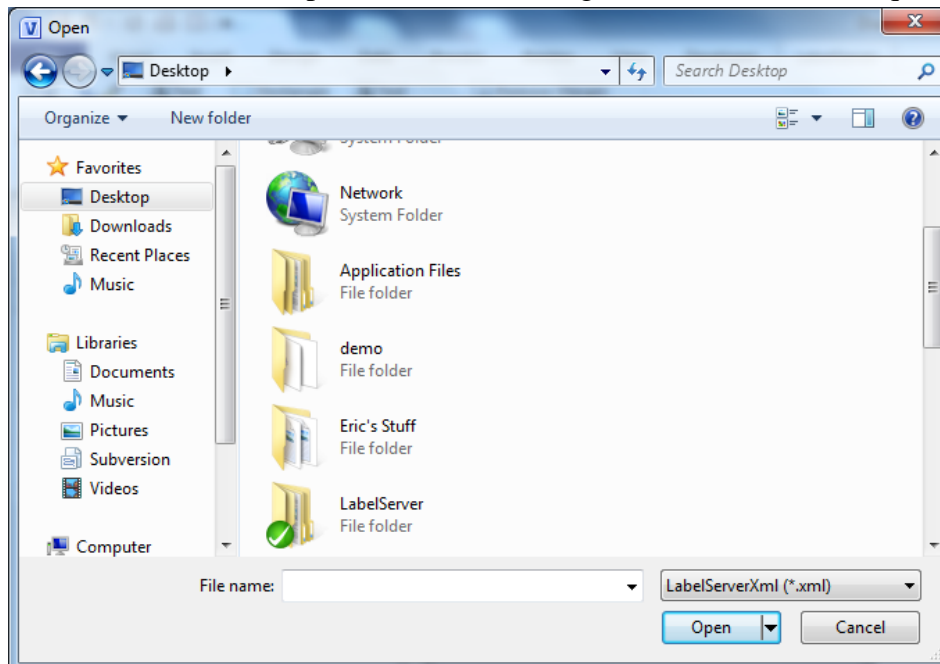
The Generation Toolbar

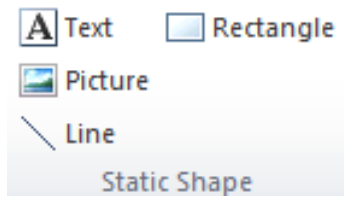
The first and most important toolbar of Microsoft Visio is the Generation toolbar. The two tools in this section contain the functionality that exports Visio files into the Endicia Label Server and imports Endicia Labels into Microsoft Visio. When initiating the process of converting a Visio Label into an Endicia Label, select the “Export” button. A window will appear titled “Save As.”



The file type should ALWAYS be listed as “LabelServerXml (*.xml)” Other filetypes are not supported, and attempting to overwrite the filetype may cause errors. After filling in a file name and saving to the directory of your choice, a file will appear with the .XML extension. This file is used to generate the label on the Label Server and should be delivered to the Label Server team.

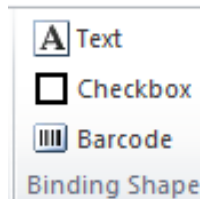
The second button, called “Import,” is an essential tool for the process of converting the existing Endicia Labels from their initial code-defined format into a Visio Label. It reads in an XML file (like the file created using the “Export” button) and draws the information into a visual layout. To import a file, select the “Import” button and navigate to the XML file in question.





The Static Shape Toolbar

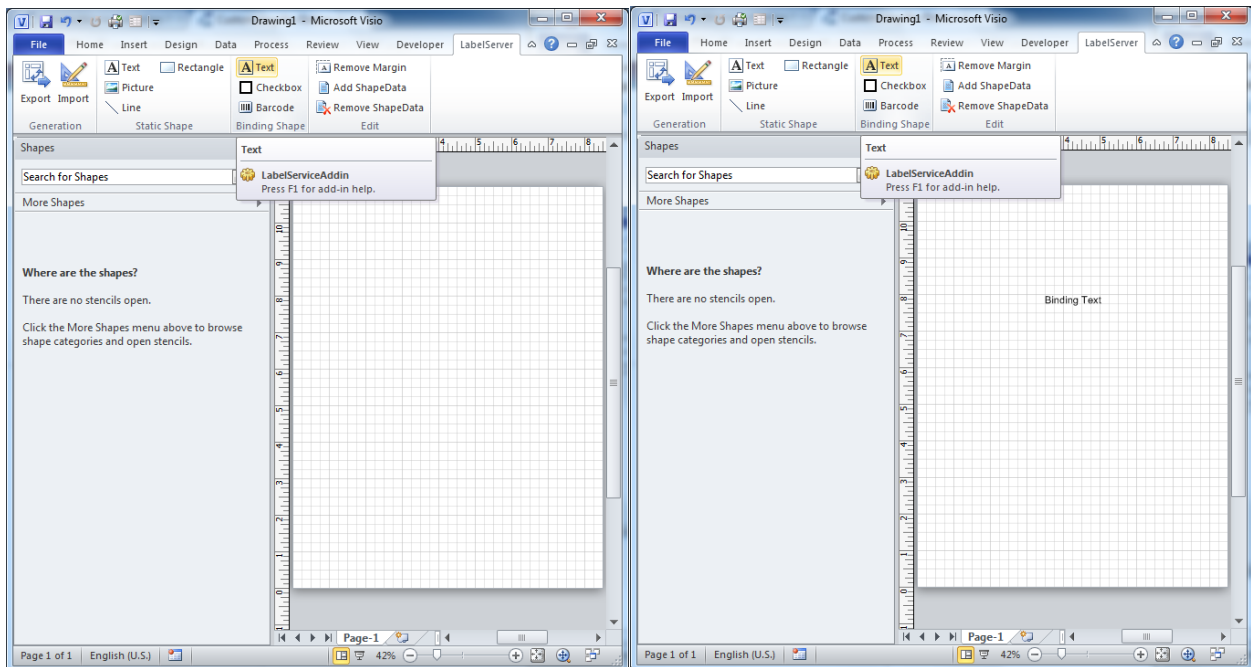
The Static Shape Toolbar is a useful tool for generating shapes that do not need modification upon a user's request. Examples of this are text fields that do not change and basic shapes (Lines, rectangles, etc.) As the most commonly used static shapes for labels are the four shown above, no others have been included. However, static shapes that are not listed above are still supported, and can be inserted.



The Binding Shape Toolbar

The binding shape toolbar contains a series of tools that are useful when designing content that will change after the user's input. For example, address fields need to be updated to display the addresses the user put in.

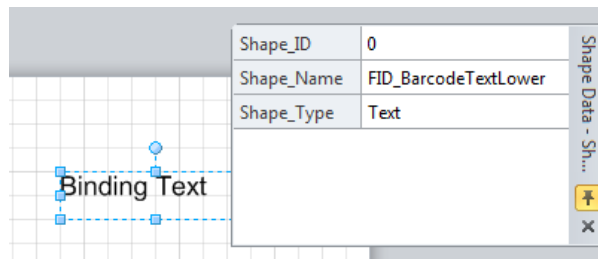
To use the binding shapes provided in the Binding Shape toolbox, all that is needed is to single click the desired binding shape. The binding shape will appear on the screen with preset sizes, fonts, etc. These will have to be manually set to your specifications.



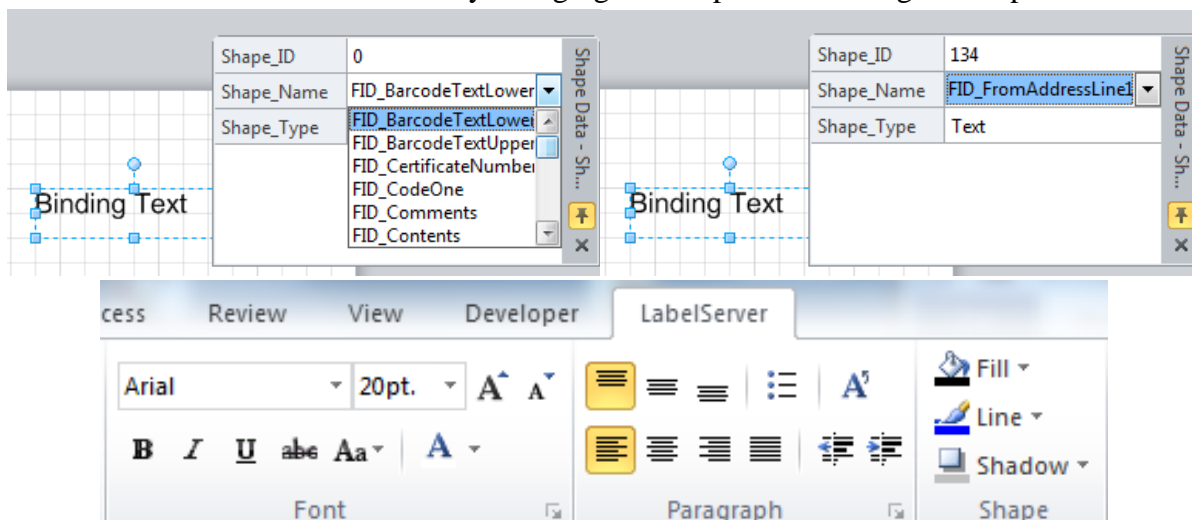
Before

After

These binding shapes contain Shape Data, which can be accessed by right clicking on a piece of binding text and selecting Data->Shape Data. This pulls up a window (As shown below) that contains information regarding the ID, name and type of a binding shape. In the example below, the binding shape has not specified, and is filled in with an empty data set.

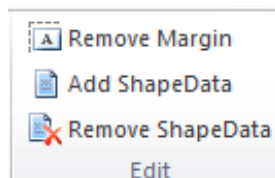


These fields can be modified by changing the Shape_Name using the drop down menu.



Font, Paragraph, and Shape Toolbars

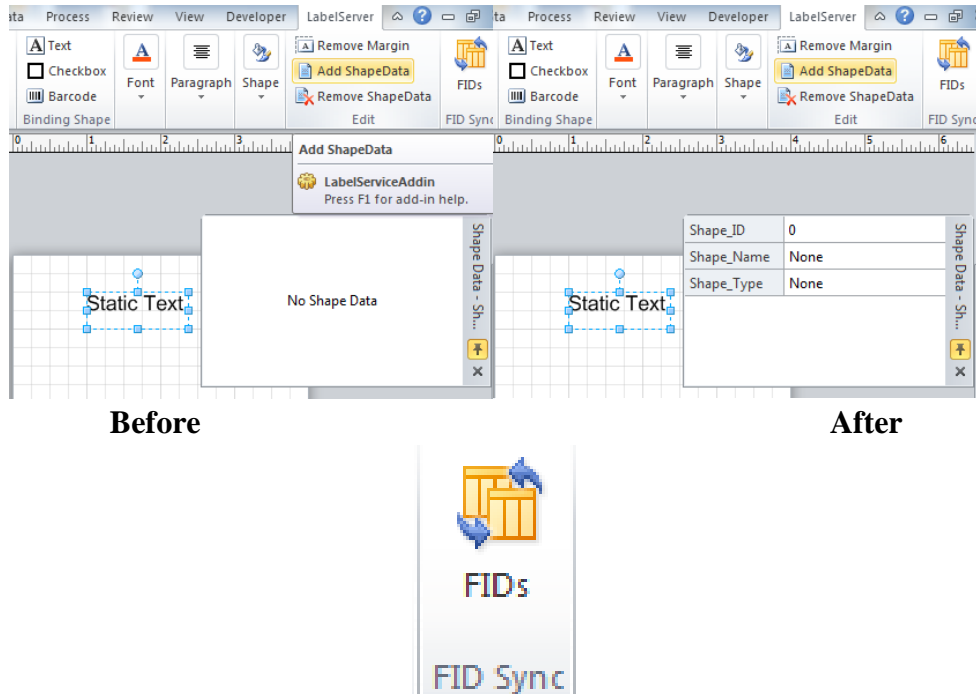
These three toolbars are identical to toolbars contained within the Home ribbon. They have been included here for the purposes of “quick use,” but do not have any altered or enhanced functionality. For more information, refer to the free Visio Online Support at <http://office.microsoft.com/en-us/visio-help/>.



The Edit Toolbar

The Edit toolbar provides three additional useful tools for designing labels. The first of these tools is the Remove Margin button. In order to ensure the chances that a static text field will misalign when translating to one of the many output options, the “Remove margin” feature sets all the margins of a text field to 0. If using one of the provided techniques for generating text fields, this tool is not needed. However, when using Visio’s original text tool (found in the

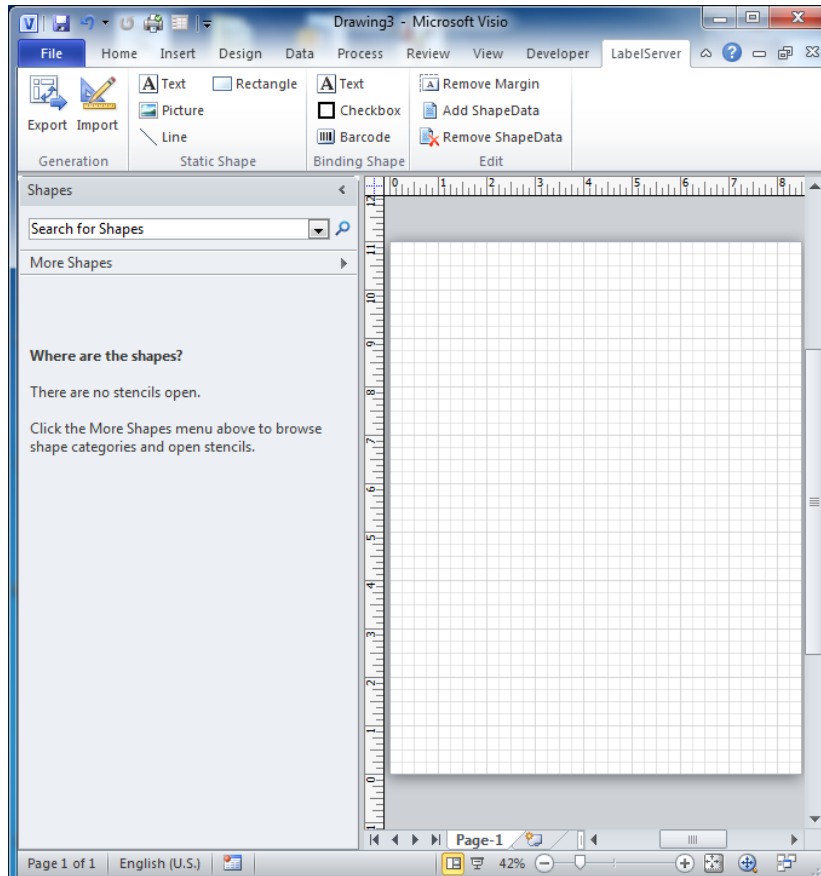
Tools toolbar in the Home Ribbon), it is recommended that the “Remove Margin” tool is used to prevent positioning errors. The second and third tool available are the “Add/Remove ShapeData” tools. These two tools allow for a static field to become a binding field and vice-versa, so that an inappropriately designed label can be easily corrected. An example of the “Add ShapeData” functionality is provided below.



FID Sync

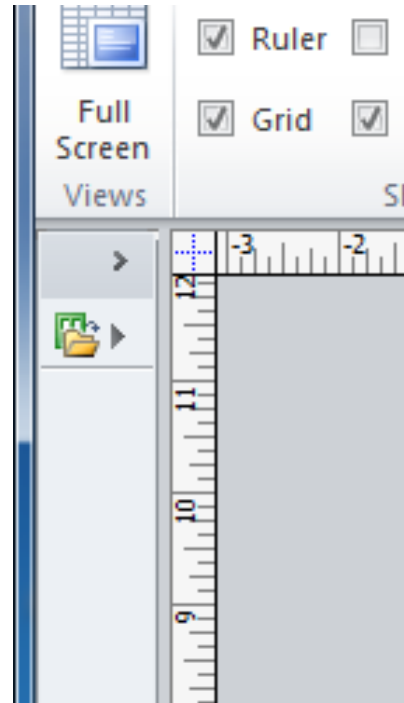
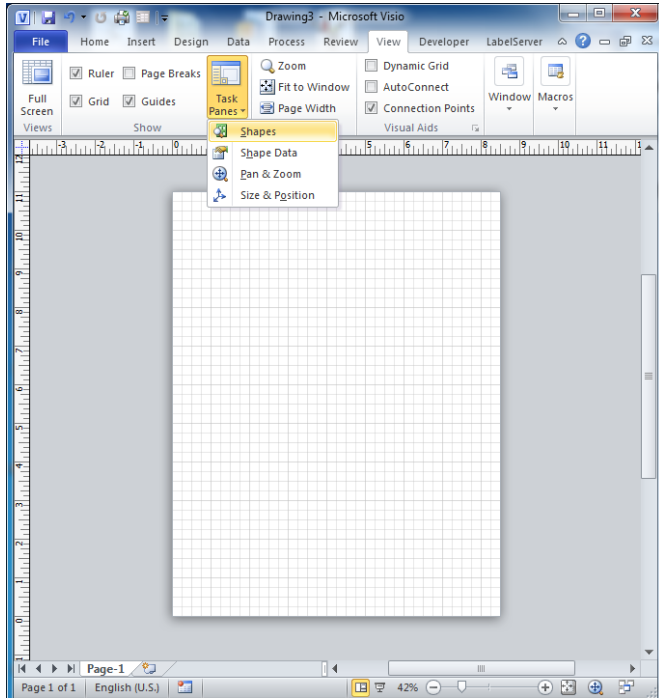
The last toolbar, the FID Sync bar, is a tool that allows users to manage their own binding fields. As shown in the example above, a binding field operates with additional data- a Shape ID, Name, and Type. These three values determine how the translation process operates on a case-by-case basis and allows the user to specify the type of data to be bound and the Stencils

In the above sections, we discussed how it is possible to generate binding shapes through use of the tools above. This section suggests an alternate solution to generating binding shapes through use of a tool specific to Visio called the Stencil tool. On the left hand side of the screen, there is a small window titled “Shapes.” This window is the home of all our Stencil functionality.

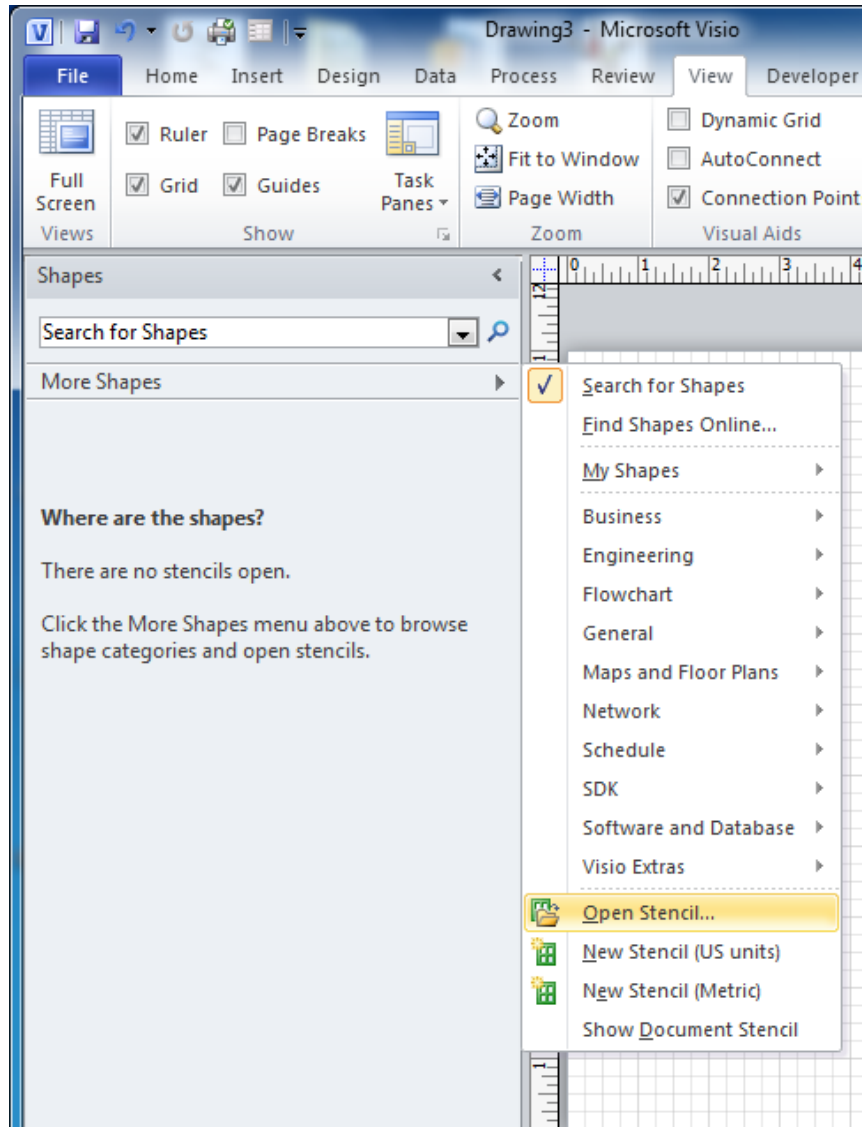


If the shapes pane is not visible, go into the “View” ribbon, and inside of the Task Panes menu, ensure that the “Shapes” option is selected, as shown on the left below.

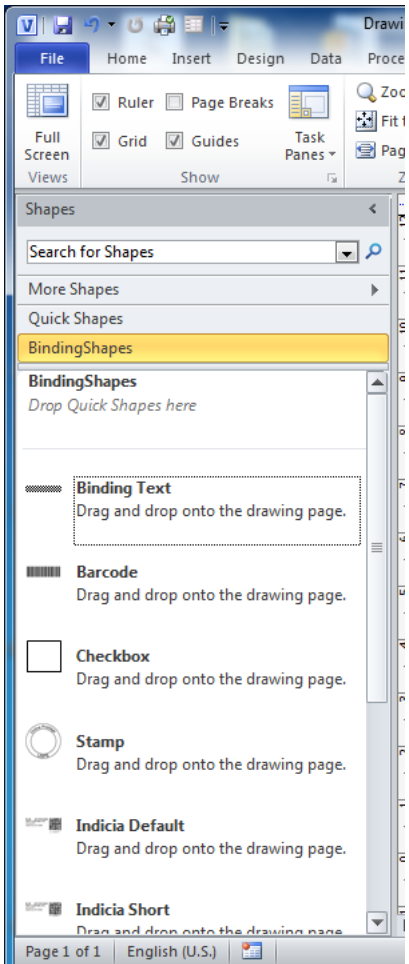
If the shapes pane is visible, yet collapsed, select the “>” sign located in the upper right corner of the window, as seen on the right below.



Once the shape window is opened and expanded, it is possible to open a stencil. To access the open stencil interface, select the filled in triangle at the far right of the tab with the text, "More Shapes." A menu will appear, and the "Open Stencil" option will be available towards the end of the menu.



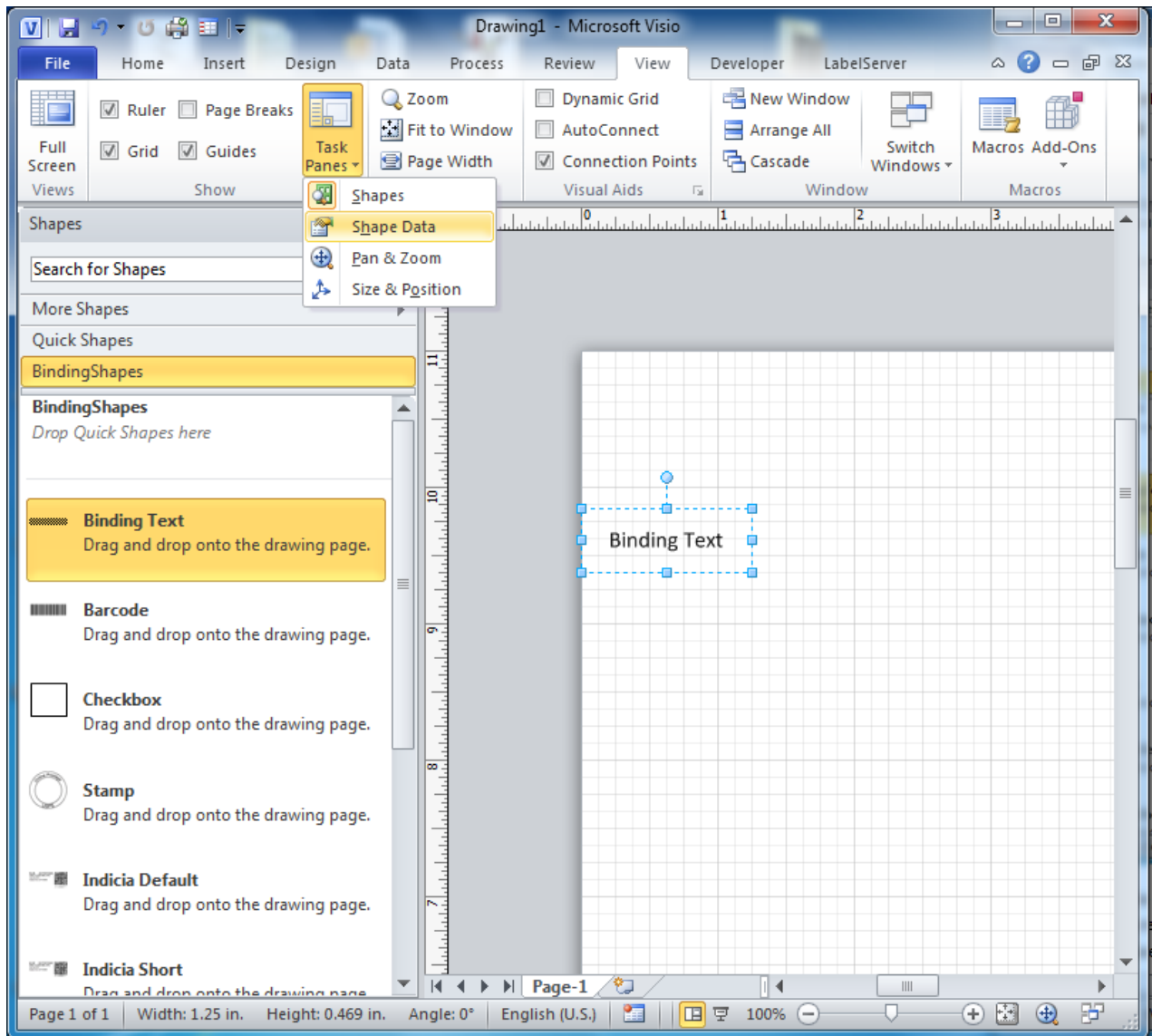
The customized stencil that has been created for the Endicia Label Server is saved under the name of BindingShapes.vss. This file is included in your installation package. After BindingShapes.vss is open, a menu that looks like the picture below should be visible.

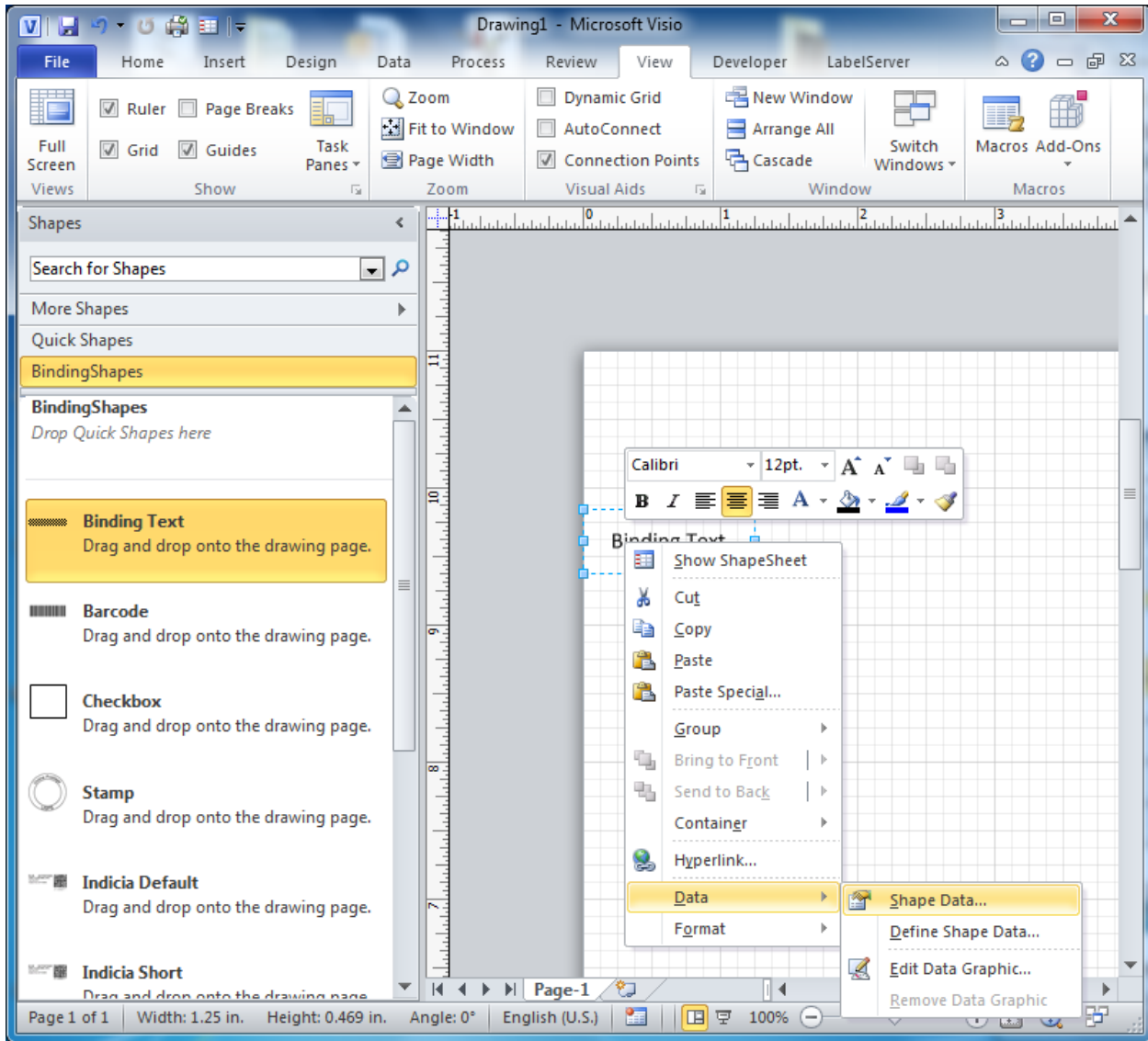


Shapes on this stencil can be added into a Visio document by simply dragging and dropping the desired shape onto the visible screen.

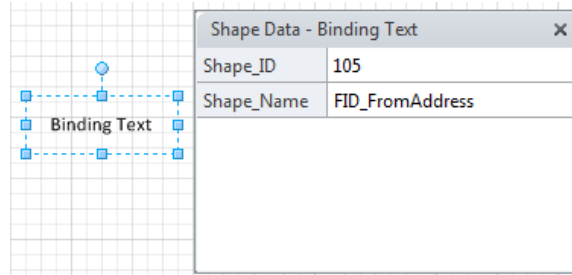
The shapes that have been provided for you in this list represent the array of possible binding shapes. These are textboxes, barcodes, checkboxes, stamps, and Indicias. While certain elements of these shapes are customizable (Such as font or size), the content of these items, like the interior text or whether or not a box is checked, are overwritten at run time. Textboxes can have their contents set to give the appearance of a completed field for design purposes, but the contents of barcodes and checkboxes are pre-set and should not be modified.

When adding a textbox, barcode, or checkbox, it is possible to further specify the contents by specifying what information will be placed in the box. The first and most important step of this process is to access the Shape Data window. This can be accomplished by either selecting “View->Task Panes->Shape Data” or by right clicking on the object in question and selecting the item “Data->Shape Data.”



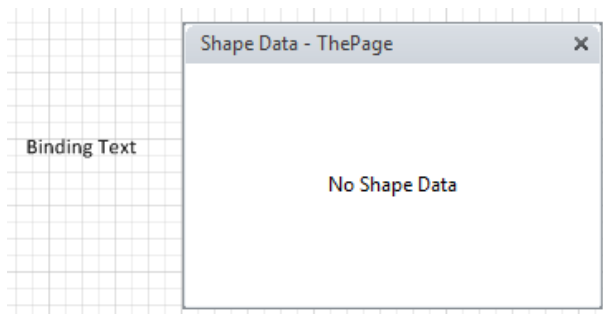


The shape data form will look like the image below:

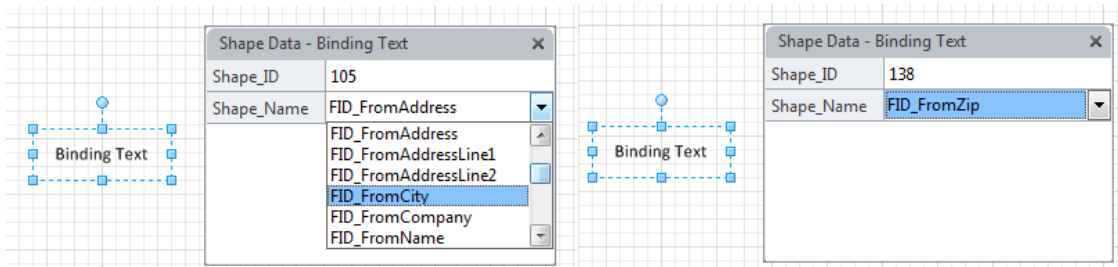


For a text box, only two pieces of shape data are required, the Shape_ID, a unique identifier used by the Label Server to assign content to elements, and a Shape_Name, which is a way for the user to identify the content. An additional piece of Shape Data, although not shown here, is Shape_Type, a description used by the Label Server. The Shape Data window will only display the data of the selected element, which can be identified by the resizing nubs around the

text box. If no element is highlighted, the Shape Data window will display data attached to the current page (Information attached to the page is not used in the binding process and exists solely for the user's benefit)



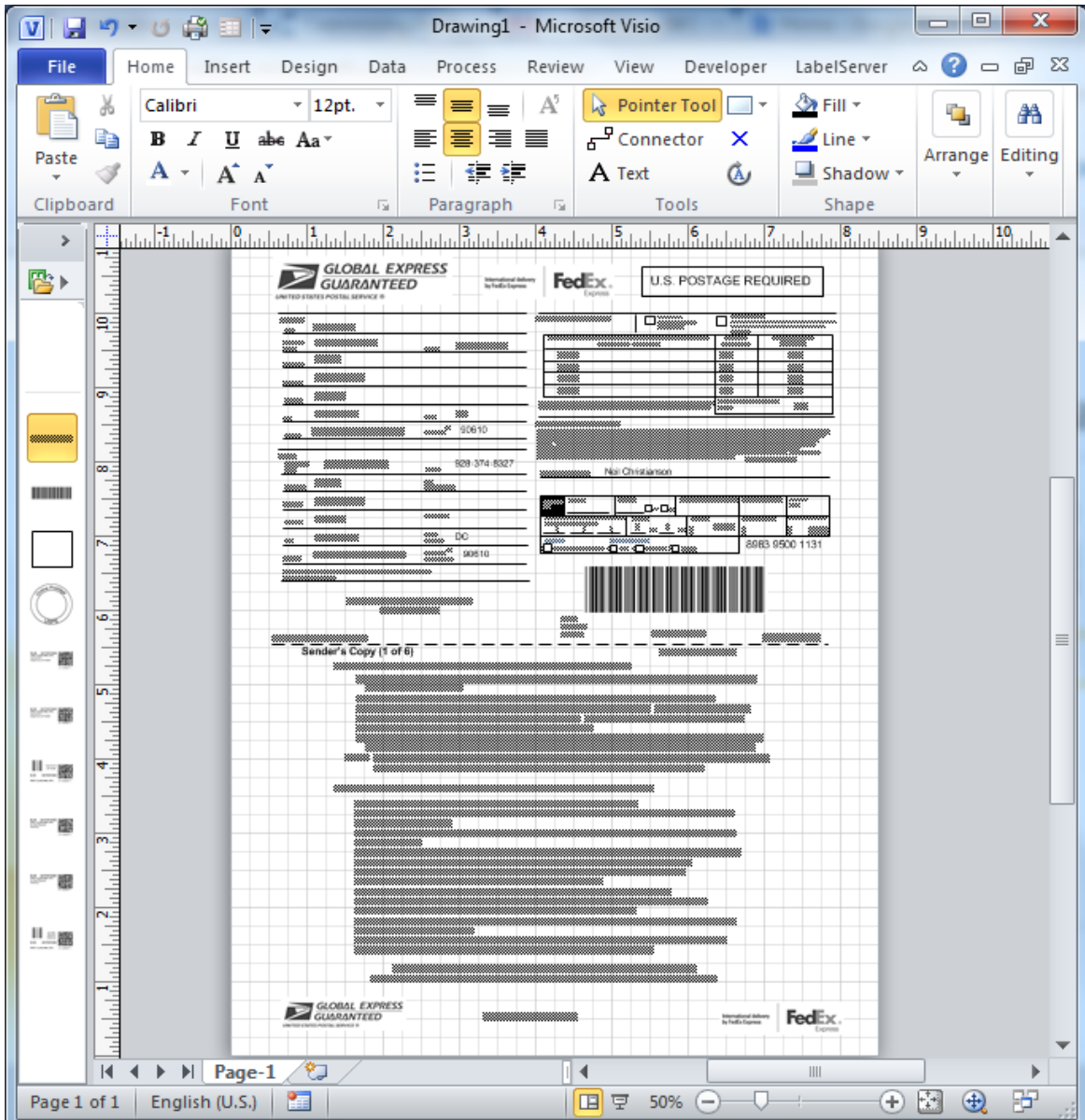
As the default shape data is specified to the From Address (FID_FromAddress), the user may wish to assign the shape data to a different value. To assign a different set of shape data, select the Shape_Name item within the Shape Data window. A dropdown arrow will appear on the right. Clicking on the dropdown arrow will allow the user to access a list of Shape_Names, and selecting a Shape_Name causes the other fields to update as well.

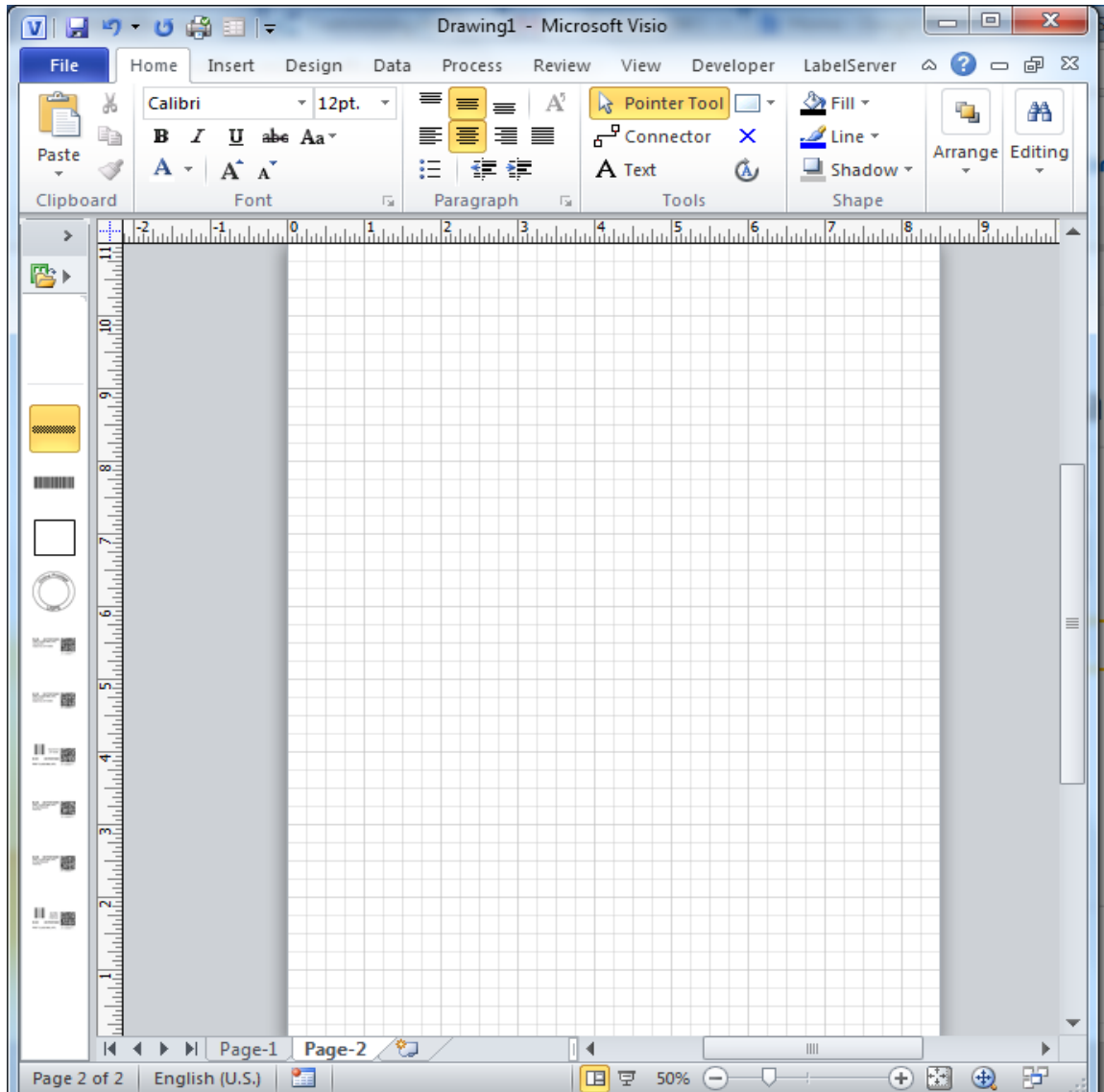


Unlike textboxes, barcodes, or checkboxes, the size, font, and content of the Indicia cannot be set. To ensure that the label appears as designed, Indicia cannot be modified, and all possible Indicia variations have been provided for use.

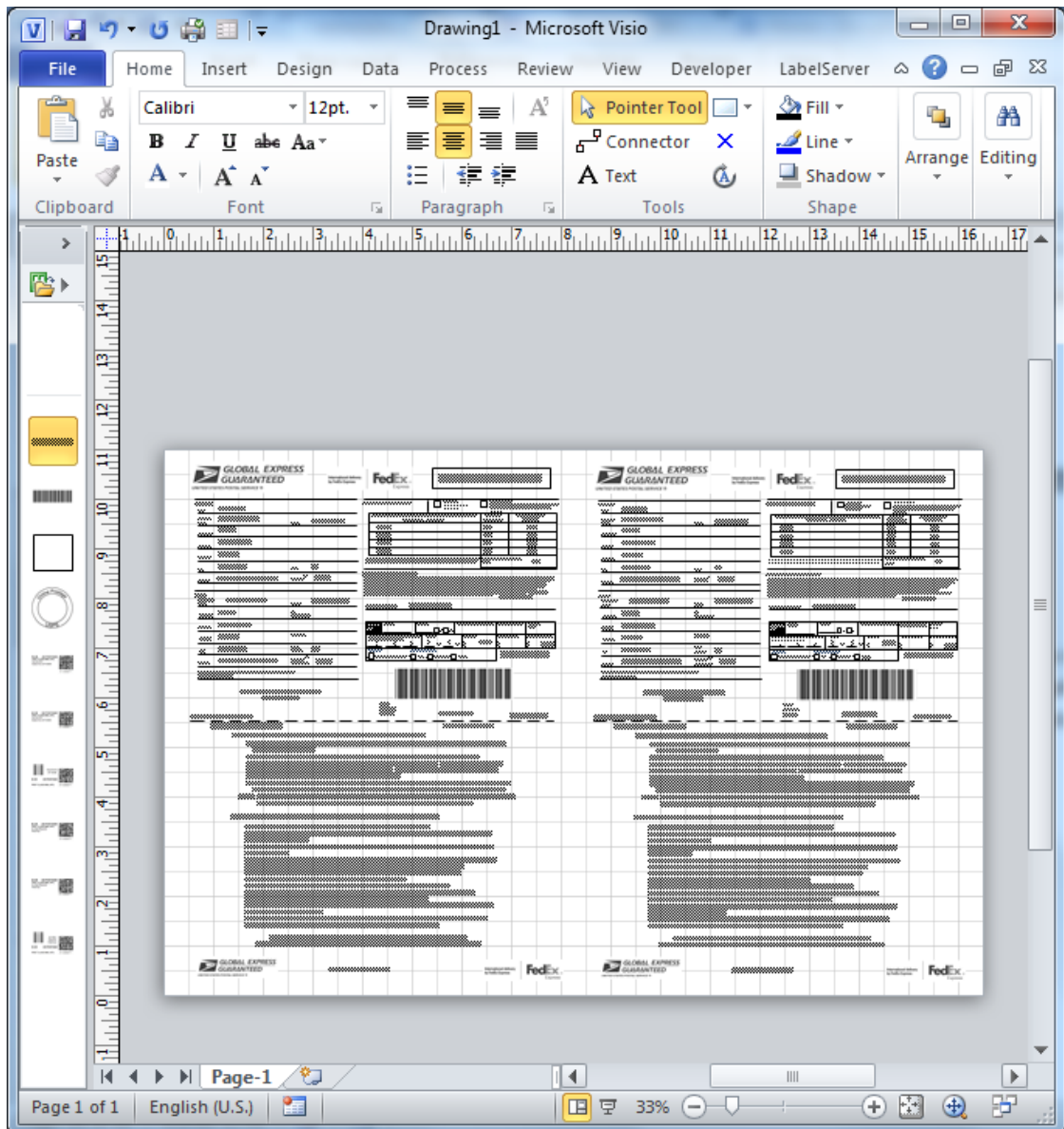
When designing a label with multiple pages, there is an easy way to keep each page separate. At the bottom of the Visio interface, to the left of the horizontal scroll bar, is a pages tab. Similar to the tabs of an internet browser, this pane allows the user to store and utilize multiple pages of a document. In the example below, only a single page exists, but it is very easy to create more. By clicking on the icon that looks like a folder with a golden asterisk a new page appears in the same size and shape of the previous page, as shown in the images below.







It is important to note that while there are not limitations on the size of any given page, it is not recommended that the user attempt to build multiple pages on a single page, as shown in the example below. If multiple pages of a label are compressed onto one Visio page, a range of incongruencies may occur from empty binding fields to absent pages.



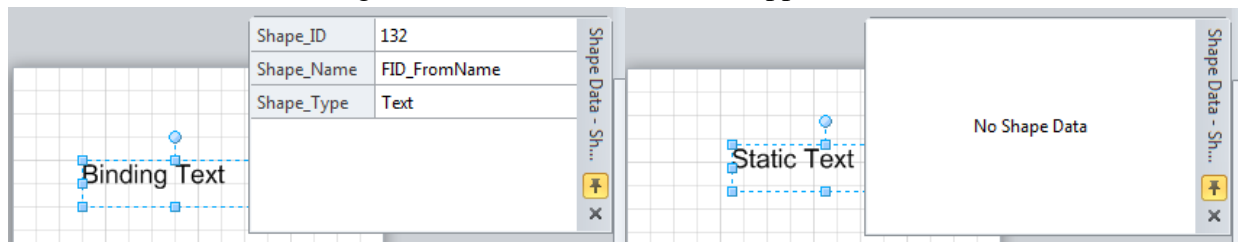
Don't stack your label's pages like this!

Binding Shapes

This section is to cover a few important tips about one of the Visio Add-In for the Label Server: Binding shapes. When designing a label for Endicia, every designer is going to want to specify a field that the user fills in. This can be text (Like an address field), images, barcodes, indicium, checkboxes, and more. However, without some assistance from Visio and the Label Designer, the Label Server isn't going to be able to tell the difference between a static shape and a binding shape.

So what's the largest difference between a static shape and a binding shape? The short answer is: Shape Data. All binding shapes possess some form of shape data to identify it to the

Label Server. In the example below, the Shape Data window has been made visible, and the difference between a binding text and a static text is made apparent.



There are a few good rules to follow when designing your binding shapes.

First, having a well-defined Shape_Name will allow your Endicia Label Server Team member to be able to quickly and correctly bind the data for this label. It will also be useful for other designers down the road to identify what shape is what.

Second, Do Not Change The Shape Type. While technically feasible, it's difficult from a design perspective to understand what the designer has created when text fields are, in fact, checkboxes. If you need to change the type of an element, either only use types that make logical sense, or delete the old element and start from scratch.

Third, only use one of each FID per page. Any instance in which two identical FIDs are found on a single page causes the second to not bind, instead binding the first item twice. A more appropriate solution is use a different binding field. If no appropriately named binding field exists, contact your Endicia Label Server Team member to create a new binding field. However, each page (See the pages section above) is a clean slate, and so a multiple-page label can possess the same field multiple times, once on each page.

Fourth, and finally, remember that there is NO DIFFERENCE between a binding shape created from the Binding Shape toolbar and a binding shape from the Stencil- whichever technique suits the designer better is acceptable.