An Interchange Standard and System for Browsing Digital Documents

Andrew Jonathan Kass

Submitted to the Department of Electrical Engineering
and Computer Science in Partial Fulfillment of the
Requirements for the Degree of

Master of Engineering in Electrical Engineering and
Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1995

Author ____
Department of Electrical Engineering and Computer Science
May 22, 1995

Certified by ____
Jerome H. Saltzer
Thesis Supervisor

Accepted by ____
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

An Interchange Standard and System for Browsing Digital Documents

by

Andrew J. Kass

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1995

In Partial Fulfillment of the Requirements for the Degree of Master
of Engineering in Electrical Engineering and Computer Science

ABSTRACT

With the advent of fast global digital communication networks, information will increasingly be delivered in electronic form. In addition, as libraries become increasingly more computerized, not just card catalogs but entire books will be stored on-line. When delivering digital documents from on-line libraries, two problems must be addressed; delivering a document which may be available in multiple formats and preserving relationships among the different formats. This thesis describes a system for delivering digital documents, called digiments, which can be in multiple arbitrary formats, while preserving the relationships between the parts and meta-data about the document itself.

Thesis Supervisor: Jerome H. Saltzer
Title: Professor, Department of Electrical Engineering and Computer Science

Acknowledgements

I would like to thank:

Professor Jerry Saltzer, my thesis advisor, who gave me guidance and support throughout the entire thesis process, without which I would never have succeeded.

Mitchell Charity, whose all-around expertise helped me through many problems, and whose opinions helped shape this work. He is probably the greatest single reason I am still sane while trying to get around Perl, AIX, the CSTR project and LCS in general.

Jack Eisan and *Mike Cook,* who were stalwart friends in Document Services and helped shape many of my ideas with their comments. I also wish to thank them for the afternoons at the Muddy Charles, which did as much as anything to keep me going throughout the year.

Jeremy Hylton, my officemate, and *Eytan Adar*, who helped me get my other research work done and supported me during the year. Jeremy especially was helpful in making sure I procrastinated enough.

The entire fifth floor of the Lab for Computer Science for having to repeatedly listen to excruciatingly detailed progress reports on my thesis.

Scott, *Liz*, my brother *Jim*, and especially *Kate*, who were all good friends to me and helped me through the tough times.

and most of all, *Mom* and *Dad*, who made it possible for me to be here in the first place, and have always stood by me. This work is for you.

This work was supported in part by the IBM Corporation, in part by the Digital Equipment Corporation, and in part by the Corporation for National Research Initiatives, using funds from the Advanced Research Projects Agency of the United States Department of Defense under grant MDA972-92-J1092

Table of Contents

1 Introduction....9
- 1.1 The Library 2000 Project....9
- 1.2 The Need for a Digital Document....9
- 1.3 Digital Documents....10
- 1.4 MIME types....11
- 1.5 Digiment Browser....12
- 1.6 Research Goals....13

2 Related Work....15
- 2.1 Existing Research on Digital Documents....15
- 2.2 Document Delivery Systems....15
- 2.3 Digital Document Systems....17
- 2.4 MIME....19
- 2.5 OpenDoc and OLE....20

3 What is a Digiment?....21
- 3.1 The Library 2000 Model....21
- 3.2 The Digiment Concept....22
- 3.3 The Digiment Concept Versus the All-in-one Concept....26
- 3.4 Requirements for a Digiment....28
- 3.5 How to Create a Digiment....31

4 The Digiment MIME types....33
- 4.1 What is MIME?....33
- 4.2 MIME Headers....36
- 4.3 The multipart/digiment type....37
- 4.4 The application/digiment type....37

5 A Prototype Digiment Generator....47
- 5.1 General Overview....47
- 5.2 PIF Files....49
- 5.3 The Create-digiment Program....50
- 5.4 Generating Page-lists and Page-maps....51
- 5.5 Generating the Other Digiment Parts....53
- 5.6 Putting It All Together....55

6 A Prototype Digiment Browser....59
- 6.1 Basic Requirements for a Browser....59
- 6.2 Creating a WWW Based Browser....60
- 6.3 How the browser works....61
- 6.4 Parsing the Digiment....65
- 6.5 Displaying the Digiment....65
- 6.6 Limitations of the Current Browser....67

7 Future Directions....69
- 7.1 Improving the Digiment Generator....69
- 7.2 Improving the Digiment Browser....71
- 7.3 New Extensions and Enhancements to the Application/digiment Type....72
- 7.4 An Alternative Form for the Page-list Digiment-type....74

7.5 The Digiment as Interchange Format 75
8 Insights and Lessons Learned 77
8.1 General Conclusions 77
8.2 The Distinction Between the Semantics of a Document and its Representation77
8.3 A Compound Digital Object Specifically Designed for Electronic Distribution and Browsing78
Appendix A The multipart/digiment and application/digiment MIME Types 81
A.1 The Multipart/digiment Type..81
A.2 The application/digiment type 81
Appendix B A Sample Digiment 93
B.1 A Digiment With Multiple Formats..93
Appendix C The CSTR Scanned Document Record 99
C.1 The CSTR Scanned Document Record, version CSTR 1.3.............................99
Bibliography 105

List of Tables

Table 4.1: Registered MIME types 34
Table 4.2: Some MIME type/subtype pairs 35
Table 4.1: Application/digiment Digiment-types 39
Table 5.1: Values for Dienst URL to access a page 52
Table 5.2: Mapping from PIF to page-map fields 53
Table 5.1: Values for Dienst URL to access a page 54
Table 6.1: Parameters to the digiment browser 64
Table A.1: Application/digiment Digiment-types 91

Chapter 1

Introduction

1.1 The Library 2000 Project

With the advent of fast global digital communication networks, information will increasingly be delivered in electronic form. In addition, as libraries become increasingly more computerized, not just card catalogs but entire books will be stored on-line. The natural convergence of these two trends is for interactive, real time delivery of electronic documents from digital libraries. With a fast CPU, cheap memory and a good display, it will be possible to browse documents on-line from a variety of sources [11]. The Library 2000 group is the MIT node of a five university research project into creating distributed digital library systems[1], browsable from any workstation with appropriate display and network hardware.

1.2 The Need for a Digital Document

For immediate transfer of data, a plethora of methods, formats and protocols may be used, both for the transfer, as well as for the content itself. However, for transferring between different systems, or transferring a document that may have content in arbitrary or multiple formats, a standard way of representing a document is needed. In addition, documents have additional information that is not necessarily reflected in the content of the document, such as copyrights, publisher, ISBN number, etc. Furthermore, this information, even if present, needs to be available in an architected way for displaying. This "meta-information" needs to travel with the content of the document, regardless of the format the content is in.

1. Otherwise known as a DDLS, pronounced "diddles".

By defining a digital document and creating a standard for its transmission, it is possible to solve these problems. By allowing the content to be in any format, but having the digital document describe the format, it is possible to use any format, present or future, for the content itself. By creating a standard way to describe the document's meta-information and content format, it is possible to transmit the document without having to worry about what format it is stored in. Finally, documents can be stored and delivered in multiple formats, making it possible to view both the scanned images and ASCII text of a document.

1.3 Digital Documents

In order to create on-line documents, we need to define what a digital document is. In this model, a digital document is data in arbitrary format accompanied by meta-information, which is part of a document, but not necessarily contained in the document itself. In order to easily transfer digital documents, I am creating a set of MIME [1] types, which describe the format of a document and contain the meta-information, while still allowing arbitrary data formats.

What exactly do we mean by a document? A document can be a book. It can be a magazine or periodical. It can be a bibliography of a technical field. A technical report is a document. Instruction manuals, TV listings, newspapers, rulebooks, and catalogs can all be considered documents. What do we mean by a digital document, or a document stored on a computer? A digital document can be composed of a collection of images in GIF, JPEG, TIFF or other format. It can be a Postscript, LaTeX or ASCII text file or set of files. It can be a set of images, the ASCII text, and a text to image map that relates the two. It can include sound. In short, there is no easy definition for a digital document. Nor do digital documents correspond well to what we call documents in the physical world.

For these reasons, we need to create a new type of object, called a digiment, for digital document. Defining a digiment will allow us to achieve some of the goals expressed above. It will provide a standard way to pass around digital documents, regardless of the format in which the content itself is stored. It will provide a standard for archiving and retrieving digital documents. It will provide for the use of multiple representations of a single "document" that can be linked together, as with a text to image map. It is intended to accommodate the use of future data formats transparently. And it will provide meta-information about a document, such as author, publisher, and copyrights. For comparison, in a book, the text on the pages of the book is the content, while the meta-information is found on the inside of the cover page, such as ISBN number, copyright and publisher.

The digiment standard is not a specific data format for storing or transmitting the content of digital documents. Rather, it is a container for transmitting or storing documents in arbitrary formats. A digiment consists of the data itself, which can be in any form, along with some associated meta-information structural information. This additional information is what differentiates a digiment from a simple set of images or data files. The structural information specifies the format that the data itself is stored in, whether it is image, text, Postscript, etc. It also specifies how the different data parts of the digiment are related to each other. The meta-information includes bibliographic information about the digiment, such as the author, publisher, copyrights, distribution rights, and relationships between different formats. By specifying the digiment in this way, it is possible to use any type of format for storing the actual data, including formats which may be created in the future.

1.4 MIME types

In order to transmit digiments, a new MIME type needs to be created. MIME stands for Multipurpose Internet Mail Extensions, and was originally developed to allow Internet email messages to contain images, sounds, applications and other data types besides plain text. However, MIME types can be used to describe the content of any type of data transfer and is increasingly used when transferring data over the internet, as with HTTP. This allows a program, such as Mosaic or Netscape, to receive arbitrary data, and then determine how to interpret the data based on the MIME type.

By creating a digiment MIME type, we can separate the concept of a digiment from the actual representation that the data uses. The digiment MIME type contains a description of the format that the actual data uses, other formats that are available, how the formats are linked, and all the other meta-information that is contained in the digiment.

MIME types consist of a type and subtype pair, in the form “type/subtype”. The type specifies the main class that the MIME object belongs to. The subtype specifies which specific type the object is within that class. The MIME standard already has a class that describes objects with multiple parts; “multipart”. By defining a subtype “digiment”, we are able to define a new MIME type, “multipart/digiment”, that describes an object with multiple objects inside. We also define a new type that corresponds to one specific part of a digiment; “application/digiment”.

A given MIME type/subtype specifies the format of the data that is being transferred. By specifying a data transfer as “multipart/digiment”, a client would know to launch a digiment-aware browser that could then interpret the digiment as a true document, instead of simply a set of pages of text.

1.5 Digiment Browser

In order to read digiments, one must have a digiment browser. The digiment browser accepts the digiment MIME type and displays the digiments on the user’s workstation. A digiment browser must be able to accept digiments in any format and either display the digiment itself, or use another application to display the digiment. In addition to displaying the content of the digiment, the browser uses the meta-information contained in the digiment. This may include showing copyright information with the content, mapping image numbers to physical pages for scanned documents, providing references to other digiments, or mapping the text of a digiment to page images.

A typical scenario for the use of a digiment browser is as a helper application for Mosaic. When Mosaic (or any other WWW browser) receives data with a digiment MIME type, it passes the data stream to the digiment browser, which can then display and manipulate the digiment. A good digiment browser should be optimized for display speed and ease of use, and support the full digiment MIME specification.

The digiment browser described in this document uses a WWW browser as its front end. The browser itself runs as a script on a WWW server that takes a pointer to a digiment as an argument. The script then generates HTML pages that can be displayed by any WWW browser, along with generating buttons to navigate around the digiment. The advantage of this approach is that it is platform independent, as you can view the digiment from any platform that has a WWW browser available.

1.6 Research Goals

This research has two main goals; defining the minimum specification for a digital document, and creating a system for transferring and displaying them. This involves defining what a digiment is and how to transfer them, and creating a browser capable of using them. This research is designed to produce both a digiment MIME type and a usable digiment browser. During the course of this research, I have two design criteria.

1. The system is designed to be widely and cheaply distributed in a short time frame. The browser will be made available at the conclusion of the research. This system is intended to be a standard for digital document use.

2. The system will be extensible to future data types and meta information. This document describes version 1.0 of the digiment MIME type, which will be the current definition of a digital document. However, the standard will be written in such a way that it will be possible to create newer versions of the MIME type that incorporates additional data and types, transparently to existing servers and client.

The remainder of this work consists of seven chapters plus an appendix. Chapter Two describes related work in this field. Chapter Three contains a description of what a digiment is from a conceptual level and the decisions that went into defining its format. In Chapter Four I define both the multipart/digiment and the application/digiment MIME types and describe how to use them. Chapter Five describes the system that I have created to generate digiments on the fly from Technical Reports on-line as part of the Library 2000 project. Chapter Six discusses the design and implementation of the WWW based digiment browser. Chapter Seven is a discussion of future directions for this work and possible extensions. Chapter Eight analyses the process of defining a new standard and trying to define what a digital document is and should be. Finally, Appendix A contains a formal description of the application/digiment and multipart/digiment MIME types, while Appendix B contains a sample digiment.

Chapter 2

Related Work

This chapter describes prior and ongoing work that is related to the transmission and viewing of digital documents.

2.1 Existing Research on Digital Documents

In recent years, the use of digital documents has started to become practical. With high bandwidth networks and a proliferation of workstations that are able to view digital documents, research into both delivery and viewing of digital documents has increased dramatically. This research can be classified into two broad categories; electronic delivery systems and digital document systems.

However, virtually all of this work is focused on directly mapping current definitions of documents into the digital domain. Most systems are only concerned with replacing traditional paper objects with an electronic version. Most document delivery systems are concerned with transmitting a single version of a document to a remote location, to either replace or supplement a paper copy. Most digital document systems are focussed on optimizing different parts of the browsing process in order to simulate reading a traditional paper document. With the exception of System 33, one of several described below, none of these systems has really attempted to use new paradigms that digital documents can now embrace, such as removing the semantic content of a document from its representation.

2.2 Document Delivery Systems

Electronic document delivery systems are systems that are designed to transfer electronic versions of documents from one computer to another. However, most systems simply attempt to replace the delivery of paper documents with electronic versions, instead of using any expanded notion of what documents are.

2.2.1 GEDI

The Group on Electronic Document Interchange is a consortium of 7 libraries and other parties who have defined a standard for electronic document interchange, called the GEDI Recommendations [2]. The goal of this group was to create a standard way to distribute electronic documents between participants. Most of their work went towards defining standard delivery protocols and mechanisms, as well as choosing a standard representation for the document. A GEDI document is a series of 300 DPI TIFF images, using CCITT G4 compression, along with some header information describing the document.

However, the GEDI system was only designed to deliver a single representation of a document at a time. Moreover, it was not designed to create a definition for a digital document, but simply to designate common protocols to transmit some version of a document from one party to another.

2.2.2 TULIP

The TULIP (The University LIcensing Project) is joint project between Elsevier Science Publishing Group and several Universities [8]. TULIP supplements paper subscriptions to journals with electronic versions of the journals. The TULIP project again is not attempting to create a new digital document system, but to create an electronic version of existing publications, specifically research journals. Because of this goal, the TULIP data structures are geared towards representing journals, and not documents in general. In fact, the TULIP project does not even have the notion of a document, but of a dataset, which can contain one or more journals. Although each journal and article has bibliographic content associated with it, the structural content described is limited to journals, articles and pages. Because of these design restrictions, TULIP cannot be easily generalized to a generic digital document system.

2.3 Digital Document Systems

The other broad category of research is entire systems to browse digital documents. However, most of these systems are mostly concerned with optimizing parts of the browsing process. This leads to the development of proprietary document formats that allow only specific representations that are optimized for display speed, resolution independence, etc. In addition, virtually all the systems are designed with the goal of replacing traditional paper documents, and only support scanned images and text. In contrast, a digiment is meant to be a generic form of a digital document that can contain any type of data, including proprietary formats that have been optimized for some property, as well as other types such as video or audio.

2.3.1 System 33

System 33 is a system developed at Xerox PARC designed to provide electronic document services [10]. Of all the systems surveyed, System 33 seemed to incorporate more of the concepts that were used in the digiment format than any other system. System 33 was designed to differentiate the abstract notion of a document from the actual representations of the document, which is one of the key concepts that drove the digiment format. In System 33, users can request different "presentations" of a document, which is a specific representation of a document. The server will then convert the document to the correct format on the fly.

However, System 33 does not create an actual object to represent a document. Documents are accessed by means of a handle; by presenting the handle to the server, a client can retrieve various information about a document, such as the available presentations or a description of the document. By presenting the handle along with presentation specific information, a client can retrieve the location of the particular presentation. The difference between this and a digiment is that a digiment is an object that contains all of this information, and can be returned to a client as a single object. In addition, the digiment contains structural information about the document, with System 33 only provides at a rudimentary level.

2.3.2 RightPages

The RightPages project is an electronic library prototype developed by AT&T Bell Laboratories [6,12]. The main focus of the project is to allow users to browse through an electronic database of journals and view the articles on-line. This system allows users to view scanned images of the journals' contents, as well as text that has been derived from Optical Character Recognition software.

However, this system is limited to only two representations of the documents; a bitmap image and the text content. In addition, the system keeps a mapping of "logical fields" in the document, such as the location of the Table of Contents entries. Because this system was designed only to view scanned images, it was not designed to be extensible to arbitrary data formats.

2.3.3 Lectern

Lectern is a document browser that is being developed as part of the Virtual Paper project at DEC Systems Research Center [7]. The primary goal of this project is to make a document browser that is good enough that people prefer to use it rather than paper documents. Lectern documents are stored in two representations; a proprietary compressed image format that has been optimized for speed, and the text of the document, with word positioning information with respect to the images. The system also includes a simple "property list" ability; documents are allowed to have arrays of (name, value) pairs of strings.

Like the RightPages system, this system focuses on replacing traditional paper documents. As such, the Lectern viewer only works with images in its proprietary format and its associated text version. This system was not designed to be a generic digital document transmission and browsing system, but to develop a specific format that could be used for certain types of documents.

2.4 MIME

MIME, or Multipurpose Internet Mail Extensions, was developed to allow the sending of data objects in arbitrary formats through standard Internet email. The current standard for Internet email, which is defined by RFC822 [5], does not make provisions for 8-bit data, lines longer than 79 characters, or multiple objects within a single message. The MIME format, which is defined in RFC1521 [1] and RFC1522, was designed to circumvent these limitations.

One of the main purposes of MIME, defining an encoding for binary data into 7-bit lines of 79 or fewer characters, is not very useful for most non-email network connections, which place no restrictions on the type of data that is being transferred. However, the MIME standard does contain two features that make it useful for many non-email based applications.

The first feature is the ability to label data objects with types and IDs. The type label allows a program to manipulate an object without having to actually understand the format itself; the program keeps a list of "helper applications" that can understand different types. When the program receives a data object that it does not know how to handle itself, it can look at the object's type and determine which helper application understands that type, so it can pass the object to that application. MIME type labels are currently the method of choice for determining the data type of a WWW transaction, for example. The ID label allows data objects to refer to each other with an unique name, so objects can build relationships among themselves.

The second feature is the ability to embed multiple objects within one "multipart" object. This allows multiple, independent objects to travel together as a single larger object.

The digiment format uses both MIME labels and the MIME multipart mechanism. By defining a new MIME type for a digiment, existing MIME compliant software will be able to treat a digiment as simply another MIME object. This allows them to hand a digiment over to a digiment browser without any modifications. All that is needed is to add a mapping from the digiment MIME type to the digiment browser. The digiment format also uses the MIME multipart mechanism to embed many different types of data within a single digiment object. Finally, by using the MIME ID label, the individual data parts within the digiment can refer to each other.

2.5 OpenDoc and OLE

OpenDoc [9] and OLE [3] are both relatively recent systems that are aimed at defining compound documents at the system level. These systems allow the creation of documents that can contain multiple data parts of any type. Data parts can even be programs themselves. Such systems build upon object oriented programming techniques to define all data as objects, which can have attributes and other objects embedded within them. For example, a word processing document can have a spreadsheet embedded within it somewhere; clicking on the spreadsheet will automatically use your spreadsheet program to edit it. This allows documents to actually contain other data types, instead of just taking a "snapshot" of another format at a specific time and converting it to the document's format.

However, these systems are simply frameworks. They do not specify any structure for the data itself, simply a way for other programs to do this. A digiment could be created as a special form of an OpenDoc or OLE document.

Chapter 3

What is a Digiment?

This chapter describes the driving motivations behind creating a digital document, and particularly the digiment format. It also explores the differences between the digiment concept and the traditional concept of a digital document. Finally, it discusses some of the requirements a digiment must have in order to accomplish these goals.

3.1 The Library 2000 Model

To understand how a digiment is used and what it is for, it is helpful to understand the Library 2000 document model. In the Library 2000 system, documents are stored on servers called **repositories**. Clients who wish to access a document obtain a **handle**, which points to a specific document stored on some repository. A handle is an identifier that will uniquely specify a document when presented to a repository. Handles can be obtained from a variety of sources, including other clients, cross references from other digiments, or a **discovery service**, the equivalent of a library card catalog.

A typical scenario for viewing a document may start with a user contacting a discovery service to search for an article on distributed systems. The discovery service returns a list of handles to documents, along with the repositories which store the documents. The user chooses a handle and then presents it to the repository with the intent of looking at the document. At this point, however, what does the repository return? It could return an arbitrary representation of the document. It could return a list of representations available. Perhaps it requires the client to request a specific action when presenting a handle to the repository. However, the most useful thing would be to return some object that completely described the document which is stored on the repository. This is where the digiment becomes useful.

3.2 The Digiment Concept

The idea of a digiment mainly came out of attempting to figure out how to request digital documents from on-line repositories, how to incorporate the information obtained during the scanning and processing process of a document, and more to the point, what digital documents were anyway. In designing a distributed digital library system, we were going to be storing digital documents in order to deliver this information to clients. However, this requires some definition of what a digital document really is, and how to deliver this information in the best way.

3.2.1 Original motivations

When developing the idea of a digiment, there were three main motivations. These ideas drove the concept of creating the digiment format and are essential to its design.

The first motivation came from attempting to answer the question of what a digital repository should return when a client asks it for a document. A repository may have the document in multiple formats; should it require that the client specify a particular format to use? If the document is available as scanned images or in another page based form, should it return the first page, require the client to specify a page, or return something else? Should the repository return bibliographic information or a list of formats? What if accessing the document will result in a charge for the client? What if this repository only contained the Japanese version of a document; should the user have to attempt to read the document before realizing that he needs to look elsewhere? Even more to the point, the creation of Universal Resource Names (URN's) is supposed to create a universally unique namespace for electronically addressable data which can be resolved into a pointer to the data. What part of the document would a URN for a digital document point to?

Clearly, there is a need to define some sort of “default” return type when requesting information about a document from a repository. This type should contain information about the different formats in which the document is available, provide bibliographic information about the document, explain how to access the different versions of the document that are available and what the access charges are, and furnish any other information that would allow the client and/or user to make intelligent decisions about whether and how to view this document.

The second motivation comes from looking at the same question in the specific case of scanned images. What if a client wants to transfer an entire document from a repository in image form? How can this be accomplished when each page is stored as a separate file? One possibility is that the client must discover the number and names of all the files that make up a copy of the document and retrieve them individually. However, existing solutions, such as multipart TIFF images, work only for that specific image format. Defining a compound document that contains each of the individual images would be a much better solution.

Finally, when scanning documents into a computer, we captured a lot of information that could be useful, most notably page maps. Each scanned image was associated with a page number assigned by a human operator, attempting to capture the original page numbering scheme as completely as possible. Furthermore, when processing the original scanned images into different formats for viewing and printing, we know which images in different formats were derived from the same original page, and therefore are alternate versions of the same page. Both these types of information could be very useful to a browser attempting to display a digital document.

3.2.2 What are documents?

When trying to think about digital documents, it helps to define what a traditional document is and how a digital document differs from that definition. This is not such an easy task, however. If you ask five different people what a document is, you will get five different answers.

What are some examples of documents? Certainly a letter, a memo, a report or even a thesis are all types of documents. The definition starts to get blurry when talking about such things as books, magazines, and brochures. But what about things like post cards, instruction manuals, catalogs, flyers and other printed material. By virtue of containing printed information, they can all be considered documents. But what about a book on tape? If the book can be considered a document, is the tape of someone reading that book also a document?

An even worse ambiguity of the term is seen when trying to determine if two documents are the same. Would a Japanese language version of this paper be considered the same document? Most people would say no, but the actual information is identical, just the representation is different. What about a fancy formatted version of this paper versus a printout of just the plain text. Would it still be the same document? Now even the representation is the same, but the layout is different. Is the version of this paper as I originally typed it the same as the version I saved after I ran a spelling checker on it? They are simply different revisions of the same paper.

The problem that we run into is that the definition of a document is closely linked to its physical representation. We are all familiar with the printed page as conveying information, so we can use the term document for most forms of printed material. So when looking at two versions of the same paper that in different languages, you can clearly see that they are not the same and, therefore, must be different documents! However, the real value of a document is the information that it conveys. Thus, it is better to think about a document in the abstract form of its contents, instead of the traditional physical object that we look at and call a document.

In order to try to avoid the ambiguities and differing meanings associated with the term document, I have adopted the word digiment. This allows me to define a new concept that is not loaded with preconceptions, such as the idea of a document.

3.2.3 What might a digiment be?

The idea of a digiment differs significantly from that of a traditional printed document in that it defines a document as the abstract information that it contains rather than the actual physical object. Thus, multiple representations of the same data could be considered to be part of the same digiment. This allows an extraordinary amount of flexibility in created digiments with different concepts of “sameness”. One repository may consider different representations of the same document to be separate digiments. It would store one digiment for the scanned images of a document, and another digiment for a Postscript version of the same document. Another repository might consider all representations of the same document to be part of the same digiment. It would have one digiment that contains both scanned images and Postscript. This idea is not just limited to representations. For example, one version of a digiment may contain Japanese, English and Korean versions of a text. However, this could also be broken down into three separate digiments. Depending on whether you consider an object in different languages to be the “same”, you can either get a digiment with multiple languages, or a digiment with just the language you want.

Digiments also do not limit the type of data they contain to just texts. A digiment can contain video clips, sound clips, images, text, or anything else. These objects can either be differing representations of the same thing, such as high and low resolution images of the same page, or they can be complementary, such as a book about animals with sound clips of each animal.

3.3 The Digiment Concept Versus the All-in-one Concept

There has been a lot of work recently on trying to create a method for transferring digital documents. Adobe has created the Acrobat Portable Document Format, or Acrobat PDF, which attempts to capture enough information about a document such that it can be viewed or printed even without the application that created it. Apple is attempting to integrate a similar system into its MacOS operating system with QuickDraw GX. No Hands Software created Common Ground, yet another portable document format. However, all of these formats have one thing in common; they define a new, proprietary format and require that all the data that they contain be converted to that format. These formats are what I describe as the all-in-one format; they attempt to define their own formats for every type of data they contain and store all of its contents in these formats.

However, this approach is not really any different than any application defining its own storage format. The only difference between a PDF and the format in which WordPerfect stores its documents, for example, is that the PDF format is designed to hold more types of data, since it must be a storage format for more types of programs than just a word processor. In addition, it may contain system resources that may not be available on other systems, such as fonts. But the basic concept is still essentially the same; define a monolithic storage format that can be used by a single, large application that is capable of understanding the entire format.

This approach has two intrinsic drawbacks, however. The first problem is that it requires any program that can read the format to understand each of the data types and be able to work with them. As the format expands to include more types of data, understanding the data types becomes increasingly complex. The program must know how to work with or display an ever increasing number of different data formats.

The second problem is that the entire format must be modified when adding new data types. This leads to a release of a new version of the Acrobat format whenever a new feature or type needs to be added. This new format in turn requires that new Acrobat browsers and creators be released. Everyone knows the frustration of trying to read a Word 5.1 document, when all you have is the program Word 4.0. The only way around this problem is to try to shoehorn a new data type into an existing data type definition, which usually results in loss of some important information.

The digiment format takes a different approach. Instead of attempting to define a single format for every type of data, it is more like an envelope into which you can place arbitrary data types while preserving relationships among them. This approach is more like the one taken by component object systems, such as OpenDoc and OLE 2.0. These schemes also use a "container" based approach; instead of defining a format that must be able to represent each of the data types within it, they define a container object that has data of arbitrary types within. Thus, an application that is very specialized can still open the container and work with only the data objects with types that it understands.

The digiment format solves the two problems that occur with all-in-one formats. First, a digiment browser does not need to understand the individual formats contained within the digiment. Since all the data types are standard, if it doesn't know how to handle a particular data type, it can pass that data object to another program that does. This only requires that each data object has an easily identifiable type associated with it, which can be accomplished by using MIME types (see "What is MIME?" on page 33). In fact, one of the advantages of having multiple representations of a document is for people who may not have the software to view one representation, but can recognize a different one.

Secondly, new types can be easily added to a digiment without requiring any changes to the standard or to the browsing software. All that is required is for the browser to add a new entry to a table containing a list of data types and actions to perform on each one. For example, if you wanted to browse a document that contained a MPEG movie, instead of having to get a version of a digiment browser that knows about MPEG movies, you could simply use any software that supports MPEG and tell the browser to use that player whenever it sees a MPEG object.

To illustrate the difference between the two, a digiment can contain a PDF object within it. This can be viewed by any client that can read the PDF format. However, the same digiment can also contain the original word processing, spreadsheet and database files that created the PDF, as well as voice annotations and a Postscript version of the PDF.

3.4 Requirements for a Digiment

There are many reasons why having the concept of a digiment as outlined above is useful. The following is a list of some of the more important reasons that were key criteria in the development of an actual digiment type.

3.4.1 Documents come in multiple formats

Digital documents tend to come in a variety of different formats. Older documents that have been scanned may be available in a high resolution greyscale or color format suitable for archiving and preserving the maximum amount of information about the original. These original scans may also be processed into high resolution monochrome images suitable for printing on laser printers. They may also be made available in lower resolution images suitable for viewing on a workstation screen. Thumbnail images of multiple pages may be available to quickly select a desired page. OCR software may be used on the images to produce a plaintext version of the document.

Even newer documents, which may already be available in electronic form, can have different formats. A Postscript document may be available in two different flavors; one that is a complete file ready to print, and another version with the fonts not included in order to reduce transmission time for those users who already have the correct fonts. Additionally, a PDF file may be available. Finally, individual page images may be generated from the Postscript original for people to view who do not want to transfer the entire Postscript file at once or who do not have a Postscript viewer.

A digiment format should be able to handle documents that have parts available in multiple representations.

3.4.2 Documents have different parts in different formats

A similar problem occurs when some parts are available in certain formats while others are not. For example, a scanned document may be available in greyscale, but with an additional color image for page 12 since that page has a color picture on it. Additionally, some parts of the document may be in formats that can not translate into alternate formats. For example, a digital document may contain a sound clip. The sound would not be available in any other format, and is not meant to be an alternative to another format, but an addition. A digiment format should allow documents to have multiple parts that may be displayed simultaneously.

3.4.3 There are relationships among different parts

Some parts in a digital document may be related to other parts. For example, a text-to-image map may relate plaintext to a specific sequence of bitmapped images. The text-to-image map needs to be able to point at both the plaintext and the specific images. A page-map may contain a mapping from a specific image to the page number that the author intended. When different formats are available, the relationship information is very important, in that it allows a client to know which parts are substitutable, or even desired, in place of part in another format. A digiment should be able to reference each part uniquely and keep track of the relationships among them.

3.4.4 Bibliographic information needs to travel with the document

Not all of the information about a document is contained within the body of the document itself. For example, the name of the author, the copyright holder, the distribution rights, and even the title of the document are important pieces of information that may not be contained within the body itself. The information found inside the cover page of a book is an example of bibliographic information about the book. Even if this information is included in the body of the document, it is usually extremely difficult to extract from Postscript, scanned images or even ASCII text, as the location is not standard, nor is the format of the information.

By defining a specific format for bibliographic information and including this as part of the document, a digiment can provide a way to extract information about a document in an architected manner.

3.4.5 We want to refer to the content of a digiment by reference only

Not everyone who requests a digital document will necessarily want to transfer the entire document in every format. For example, someone without a Postscript interpreter would have no need for a Postscript version of the document if all they wished to do was view it somehow. If a specific page is available in both greyscale and color versions, the user has no need for the greyscale version if he wishes to view the color version, and has no need for the color version if he does not have a color monitor! Therefore, instead of requiring that the digiment contain the actual data of the document, which may come in many formats and thus take an extremely large amount of bandwidth to transfer, the digiment should contain the data by reference. This allows a client to obtain everything they need to know about a document without having the incredible overhead of transferring the data itself, and choose exactly what and when it wants to transfer and display. This can allow a client to be much more efficient in terms of caching and bandwidth utilization.

3.5 How to Create a Digiment

Once we have laid out the basic requirements for a digiment, we need to define a new type of digital object, which is the actual digiment. This object should be structured in such a way that all of the relevant information about a document discussed above can be easily stored and accessed. The object should be modular, permitting easy addition of new formats and structural information. It should be easy to parse and easily recognizable as a digiment. Furthermore, it should not restrict the types of data which it can contain.

I have chosen a format for digiments which relies heavily on MIME, or Multipurpose Internet Mail Extensions. MIME provides two very useful concepts; the ability to embed sub-objects within objects, and the ability to associate a data type with each object. By creating a digiment out of multiple sub-objects, a digiment can easily be created by simply selecting a legal set of these sub-objects. New formats can be added by incorporating more sub-objects. In addition, since the digiment format uses MIME types, it is easy to add arbitrary data types, as long as they are tagged with a valid MIME type. The digiment itself is simply a new MIME type, which can be passed around like any other digital object.

The actual objects which make up a digiment are described in Chapter 4.

Chapter 4

The Digiment MIME types

This chapter outlines the function and purpose of the different MIME types which make up a digiment. A specification of the actual types can be found in Appendix A on page 81.

4.1 What is MIME?

MIME stands for Multipurpose Internet Mail Extensions, which was defined by RFC1341, and later revised in RFC1521, and was originally developed to allow Internet email messages to contain images, sounds, applications and other data types besides plain text. In addition, it was designed to work around current limitations in Internet mail gateways. Existing Internet mail gateways have been designed to deal with messages which have been formatted according to the RFC822 email specification, which limits line length to 79 characters and the character set to 7-bits. These limitations prohibit the transfer of files which may include 8-bit characters or any arbitrary binary data.

A MIME object is any collection of bits which is put together in a way which adheres to the MIME specification. Some MIME objects may contain other MIME objects nested inside. A MIME compliant agent is any program which can recognize MIME objects and perform appropriate actions on them, depending on the type of object it is.

4.1.1 MIME types

All MIME objects have a “type/subtype” pair associated with them. This allows a MIME compliant agent to determine what format the object is in and take appropriate action. For example, if an object has a type which specifies that it is a TIFF image, the agent can either display the image itself, if it is capable, or it can pass the object to another program which it knows can display TIFF images. This system of assigning types to every object allows an agent to keep a table of default actions for each MIME type which it knows about. The action can be either to handle the type internally, or to call "helper" applications, which it knows can process objects of that type.

There are currently seven MIME types registered. These are listed in Table 4.1 on page 34. The main types are intended to categorize most types of data. The application type is intended to hold subtypes which do not fit into the other classes, such as binary data or application specific data. The audio type is intended to contain any subtype which indicates audio data. The image type is intended for bitmapped image data of any type. The message type is used to indicate encapsulated messages of various types. The multipart type is a special type which allows multiple objects to be nested within it. The text type is used for data which is primarily textual. Finally, the video type is used for any time-varying-picture data, possibly with synchronized audio.

MIME Type	Used for
application	Application specific data
audio	Audio based data
image	Bitmapped images
message	Encapsulated messages
multipart	An object which contains several other objects
text	Textual data
video	A time-varying-picture image

Table 4.1: Registered MIME types

4.1.2 Subtypes and parameters

Each of these main MIME types can have an unlimited number of subtypes, which describe the specific type of object within the general class. Some example MIME type/ subtype pairs are shown in Table 4.2 on page 35.

By using the combination of a type/subtype pair, it is possible to characterize the type of data which an object contains. This specification can be further refined by the use of parameters. Parameters can be added to a type/subtype pair to specify options within that subtype. For example, a 300 DPI, 8 bit greyscale GIF image could be specified as "image/gif;dpi=300;bits=8", while a 600 DPI monochrome GIF image could be specified as "image/gif;dpi=600;bits=1".

MIME type/subtype	Meaning
image/gif	Image in CompuServe GIF format
image/tiff	Image in TIFF format
audio/basic	Audio in ISDN 8-bit 8kH format
text/plain	Plain ASCII text
application/postscript	A Postscript document

Table 4.2: Some MIME type/subtype pairs

4.1.3 The multipart MIME type

The multipart MIME type is special in that it indicates that an object with this type actually contains multiple other objects, each of which have their own MIME type. The most commonly used subtype for multipart is multipart/mixed, which indicates multiple independent objects are contained within, intended to be viewed serially. Another type, multipart/alternative has similar semantics, but indicates that the individual objects are representations of the same data in alternative formats.

It is possible to distinguish where one object in a multipart message ends and another begins because each object is preceded by a special *encapsulation boundary*, which is given as a parameter to the multipart object. The final object is then concluded by a *clos-*

ing boundary. The boundary string is specified by the required parameter “boundary” to the multipart object.

The encapsulation boundary consists of a blank line, followed by two hyphens, followed by the string given in the boundary parameter, followed by a newline. Every occurrence of this sequence in the multipart object denotes the beginning of a new sub-object. The MIME headers and body follow immediately after the encapsulation boundary. Any data which precedes the first encapsulation boundary is ignored.

The closing boundary consists of a blank line, followed by two hyphens, followed by the string given in the boundary parameter, followed by two hyphens, terminated with a newline. Any data which follows the closing boundary is ignored.

Boundary strings must be chosen so that they do not occur in the body of the multipart object. Algorithms should be used to choose a boundary string which ensures that this restriction is met. For a more thorough description of multipart and other MIME types, please refer to RFC1521.

4.2 MIME Headers

A MIME object consists of a set of headers which describe the contents of the object, followed by a blank line, followed by the contents of the object. Two headers in particular are used very often; Content-Type and Content-ID.

4.2.1 Content-Type

The way in which a MIME object specifies its type is with the header **Content-Type:**, followed by the MIME type/subtype pair. This should be the first line in the headers. A typical multipart object would be typed like this:

```
Content-Type:  multipart/mixed; boundary="=======digboundary====="
```

Every MIME object is required to have this header.

4.2.2 Content-IDs

Each MIME object can have an ID associated with it. This ID should be as world-unique as possible and can be used to refer to a particular MIME object. This is especially useful for caching and for multipart objects which want to refer to each other. The ID is specified by a line with the header **Content-ID:**, followed by the ID. Thus, a typical content ID would look like this:

```
Content-ID:    MIT-LCS//MIT/LCS/TR-341.bib
```

4.3 The multipart/digiment type

I define multipart/digiment to be the base MIME type for a digiment. When transferring a digiment, the type used should be multipart/digiment. The multipart digiment is syntactically equivalent to the multipart/mixed type, but with different semantics. The only type allowed within a multipart/digiment is application/digiment. The use of multipart/digiment instead of multipart/mixed is to differentiate a digiment from a random multipart message. This special subtype allows MIME agents to detect a multipart/digiment and pass the entire multipart object to a browser which understands digiments. If a digiment used a standard MIME multipart subtype, a MIME agent may extract the individual parts from the multipart/digiment, which are not useful by themselves. When referring to a "digiment", we are actually referring to the multipart/digiment object which contains all the other data objects; the whole is known as the digiment.

A multipart/digiment consists of a number of application/digiment objects, which are defined in the following section. A minimal multipart/digiment consists of an application/digiment of type page-list, followed by one or more application/digiment parts of varying types. Any object which is not an application/digiment MIME type may be ignored.

4.4 The application/digiment type

The application/digiment type is the meat of the digiment. A digiment is made up of a collection of application/digiments which describe the available data formats for the digiment and the relationships among them. Each application/digiment consists of two MIME headers; a Content-Type: of application/digiment, and a Content-ID with a globally unique ID. These headers must always be present. Following these headers is a blank line and the body of the application/digiment. Thus, the beginning of an application/digiment would look like this:

```
Content-Type:  application/digiment
Content-ID:    MIT-LCS//MIT/LCS/TR-341.map

start of application/digiment body
```

There are multiple subtypes of application/digiments. Each application/digiment describes some data of the digiment, or a relationship among different data types. The specific type of the application/digiment is specified in the application/digiment headers.

4.4.1 Application/digiment headers

Each digiment-type has the same first two lines as a header. The first line must contain the tag **Version:**, followed by the version number. This document describes version 1.0 of each digiment-type. The only current legal version number is 1.0. Any future version may describe a different format and must be described in a separate document. The second line must contain the tag **Digiment-type:** followed by a legal digiment-type. The currently defined types are described below, as well as being listed in Table 4.1 on page 39. Thus, a correct part-list application/digiment would start like this (including the MIME headers):

```
Content-Type:  application/digiment
Content-ID:    MIT-LCS//MIT/LCS/TR-341.parts

Version:       1.0
Digiment-type: part-list
```

The rest of the application/digiment body follows immediately afterwards, without any blank lines.

4.4.2 Summary of digiment-types

These types are designed to cover the common cases for digiment delivery. However, they are certainly not a complete set of all possibilities for capturing all the various types of data and relationships between types which may be delivered. Therefore, it is expected that this set will be expanded by future types to enhance the flexibility of delivery options and capture more data. The following table is a complete listing of the types defined in this document.

Digiment-type:	Description
part-list	Listing of all the application/digiments in this digiment
bibliography	Bibliographic material in RFC1357 format
page-map	A mapping from VSN to page #
page-list	A list of pages available in a certain format
preferred-order	A listing of pages or body-list parts in a particular order from various page-list or body-list parts
body-list	A file which contains multiple pages

Table 4.1: Application/digiment Digiment-types

4.4.3 The part-list type

The part-list type is the only required type of application/digiment in a digiment, and must be the first part in a digiment. This type contains a listing of each of the other application/digiments in the digiment, with their Digiment-type and Content-ID. In addition, types which contain data, such as a page-list or body-list, may also have the MIME type of the data included as a hint.

The part-list type is used to describe the contents of the digiment. Each application/ digiment which is part of the digiment must be listed in the part-list, or can be ignored. By parsing a part-list, a browser can quickly determine which parts contain data, such as page-lists, and which parts specify structure, such as page-maps. The browser can then build lists of the available page-lists, body-lists and preferred-order types.

4.4.4 The bibliography type

The bibliography type is meant to contain bibliographic and other meta-information about the document which is not otherwise found in the body of the document. This can include the authors, the title, date of publication, number of pages, the abstract, copyright notices, and other types of information about the document. The information is stored in the standard bibliographic format used by the CSTR project, which is defined by RFC1357 [4]. The bibliography type allows a digiment browser to extract this information easily from the digiment. The information can then be used to help display the digiment.

A sample bibliography type can be found in section A.2.3 on page 83.

4.4.5 The page-list type

The page-list type is the main digiment-type used for specifying page-based data. Page-based data refers to any data which is accessible a single page at a time. This type was originally intended for image based data, such as a document which has been scanned, but may be used for any data type which is available a page at a time. For example, a document may be available in Postscript form, which would be specified in a body-list. Normally, a client would have to download the entire Postscript file to view a single page. However, a server may be able to serve a single page of a Postscript version of a document at a time. The entire Postscript version could then be listed as a page-list and accessed a page at a time, permitting additional information to be assigned to each page and allowing finer control of the relationships between different versions. This also has the advantage of not requiring an entire body-list part to be transferred to view a single page, making for more effective use of bandwidth.

Each distinct data format should have its own page-list, even if it does not contain an entry for each page. For example, if a digiment contained pages 1-15 of a document in greyscale GIF, but also included a separate copy of page 12 in color GIF, the digiment would have two separate page-lists. One would contain a listing for each of the pages 1-15 available in greyscale, while the other would only have an entry for page 12, which is a color GIF. The relationship between pages which contain alternate forms of the same data is maintained in a separate part called a page-map.

The page-list type consists of the MIME type of the individual pages, an page-map, which is associate with the page-list, and a list of pages that are available in the specified data type. Each page is assumed to be in the same format, which is specified by the MIME type. Separate formats should be listed in separate page-lists. Each page-list is also associated with a page-map. Page-maps, which are described in section 4.4.6 on page 42, contain various meta-information about a specific page, as well as preserve relationships between pages which contain the same data, but are in different formats.

Each page has a single entry in the page-list. An entry consists of a **Virtual Sequence Number**, or **VSN**, a **URL** which can be used to access the page data, and a **Page Description** field. The VSN uniquely identifies a page within a given page-list, and is assumed to have some absolute ordering. That is, a page with a VSN less than a second page should be displayed before the page with the higher VSN and vice-versa. VSNs do not have to be sequential, but must have some absolute relative ordering. The URL field specifies a URL which can be resolved to retrieve the page data. A page-list can have a URL-stem: header, which will be prepended to the URL, so the URL field does not need to be a valid URL by itself; however, the combination of the URL-stem: field and the URL field must create a valid URL which can be resolved to access the page data. Finally, the Page Description field is used to label the specific version of the page. This field is free text and may contain most printable characters and punctuation characters. The page description field is displayed by the browser to label a button, to indicate which version of a page a user is looking at, and which others are available. By clicking on the different buttons, users can view different versions of the same page. For example, page 12 which is in greyscale format might have a page description label of “Standard image”, while a color version of page 12 might have a label of “Color image”, while an extremely low resolution version might have a label of “Quick preview”.

A sample page-list type can be found in section A.2.4 on page 84.

4.4.6 The page-map type

The page-map type includes extra data about a specific page, and is used to relate different versions of the same page together. Multiple page-lists can share a single page-map; this is how multiple pages can indicate that they refer to the same virtual page. Page-maps are also used to describe the type of page, as well as provide the page number.

A page-map consists of a mapping for each page in the page-map. Each page has a single entry in the page-map. The entry contains a **VSN**, a **Page Type**, and a **Page Name**. This VSN corresponds to the VSN for a specific page in every page-list which points to this page-map. For example, if page-list A pointed to page-map A’, the page with VSN 12 in A would correspond to the map with VSN 12 in A’. In addition, if page-list B also pointed to page-map A’, the page with VSN 12 in A would be considered an alternate version of the page with VSN 12 in B. It is permissible for multiple page-lists which all point to a single page-map to have VSN series that do not completely overlap, as long as those VSNs that do overlap refer to the same page in alternate formats and the VSNs that do not overlap refer to pages which are only available in certain formats. This allows flexibility in specifying relationships between pages in various formats, where all the pages may not be available in each format.

The Page Type field can hold one of five values; supporting, title page, unnumbered, an integer number, or unknown. This field can be used by the browser to determine what type of logical page this is. A value of supporting means that the page is not really part of the document, but is used in processing or contains calibration or miscellaneous information. These pages are not normally displayed or printed. A value of title page indicates that the given page is the first or title page for the given digiment. A value of unnumbered means that the page has no page number, while an integer number refers to the actual page number in the document. Finally, unknown indicates that there is no logical information about this page.

Finally, the Page Name field is simply displayed by the browser to indicate which page is currently being viewed. Normally this field will contain the page number, but it may contain “Title page”, or a description of a processing control or calibration page.

A sample page-map type can be found in section A.2.5 on page 86.

4.4.7 The body-list type

The body-list type is similar to the page-list type, but it is used to specify data segments that contain multiple pages, or are not page based. For example, a Postscript version of a document would be placed in a body-list. However, sometimes a segment may contain only part of a document, such as a Postscript file for each chapter in a document. In this case, a body-list will contain an entry for each file which makes up the document.

The body-list type consists of the MIME type of the individual segments, and a list of segments which are available in the specified data type. Each segment is assumed to be in the same type, which is specified by the MIME type. Separate formats should be listed in separate body-lists.

Each data segment has a single entry in the body-list. An entry consists of a **Virtual Sequence Number**, or **VSN**, a **URL** which can be used to access the segment, and a **Segment Description** field. The VSN uniquely identifies a segment within a given body-list, and is assumed to have some absolute ordering. That is, a segment with a VSN less than a second segment should be displayed before the segment with the higher VSN and vice-versa. VSNs do not have to be sequential, but must have some absolute relative ordering. The URL field specifies a URL which can be resolved to retrieve the segment data. A body-list can have a URL-stem: header, which will be prepended to the URL, so the URL field does not need to be a valid URL by itself; however, the combination of the URL-stem: field and the URL field must create a valid URL which can be resolved to access the segment data. Finally, the Segment Description field is used to label the specific version of the segment. This field is free text and may contain most printable characters and punctuation characters. The segment description field is displayed by the browser to label a button, to indicate at which segment of a document a user is looking. For example, a segment may be named “Chapter 1” or “Chapters 4-7”.

A sample body-list type can be found in section A.2.6 on page 88.

4.4.8 The preferred-order type

The preferred-order type is used to specify a preferred sequence of data objects for the browser to display. Although page-lists and body-lists can do this, they are limited to only specifying pages or segments which are in a specific format. By using a preferred-order type, a digiment can specify a sequence of objects which are listed in any page-list or body-list in the digiment. For example, a digiment can specify that a preferred viewing order is pages 1 through 5 from page-list A, pages 6 and 7 from page-list B, segment 3 from body-list C, and pages 8 through 45 from page-list A. Digiments may contain multiple preferred-order parts. Thus, another preferred-order may be identical to the previous one described, but use page-list C instead of page-list A.

The preferred-order type consists of a listing of data objects, along with a **Preference Description**. Each object has a single entry in the preferred-order list. An entry consists of a **Preferred Sequence Number**, or **PSN**, a **Content-ID**, and a **VSN**. The PSN specifies the order in which the objects are to be displayed. The first object displayed is the object which corresponds to the lowest PSN, and objects are displayed in order of increasing PSNs. The Content-ID is the content-ID of either a part-list or body-list digiment part. The VSN specifies a specific page or segment from the part-list or body-list part. By using a VSN and Content-ID pair, it is possible to specify any object which is contained in a digiment. Thus, a preferred-order part can specify any arbitrary viewing order of objects contained within a digiment.

The preference description is a free text field which contains a short description of the preferred-order. This text is displayed by the browser to let the user choose which version of the digiment he wishes to view. For example, one preferred-order may have a preference description of “For use with color monitors”, while another preferred-order may have a description of “Suitable for high resolution printers”.

A sample preferred-order type can be found in section A.2.7 on page 89.

Chapter 5

A Prototype Digiment Generator

This chapter describes a system that is able to generate digiments on the fly for any of the MIT LCS/AI Technical Reports that are on-line as part of the Library 2000 project.

5.1 General Overview

As part of designing the digiment format, I have created a system that generates multipart/digiments for the Technical Reports (TRs) which have been placed on-line as part of the Library 2000 project. These TRs generally consist of either Postscript, which has been collected from the author as part of the submission process, or scanned images, which have been collected from our goal of scanning in all LCS and AI Lab technical reports. In some cases, both images, which usually have been generated from the Postscript, and a Postscript version exist. In addition, all the TRs on-line contain a bibliographic entry in a standard bibliographic reference form, defined by RFC1357.

5.1.1 Creating a digiment on the fly

The system that I created generates digiments "on the fly" for MIT TRs. This is accomplished by having the server run a special program, create-digiment, whenever a TR is requested in digiment form. This program determines the available formats of the TR, and generates the appropriate digiment. Create-digiment also uses a helper program, called pif2digiment, to create page-maps and page-lists. Both of these programs are written in Perl. Both programs also assume that they have access to the filesystem where the actual document images and other files are stored locally. This is currently assured by having the programs run on the server workstation.

5.1.2 How documents are stored on the server

The create-digiment program has knowledge of how the actual TRs are stored locally in order to determine what parts of the TR are available and in which formats. Each TR is stored in a specific directory on the server; the name of this directory can be uniquely determined by the ID of the TR. A special routine can take the ID of a TR and return the path name of the directory where it is stored.

Within each document directory, there are several items. Almost every TR available has an RFC1357 bibliographic record stored there. If a Postscript version is available, it too will be stored in this directory. If the document was scanned, a copy of the scanned document record will be available here. Finally, a subdirectory exists for each different format for which images are available. By looking at the contents of this directory, it is possible to determine what formats a TR is available in and to access each of these parts.

5.1.3 Parts of the digiment generator system

The process of generating a digiment can be broken down into several stages. The first step is acquiring the documents. This was already being done by the Library 2000 project. Documents are either scanned in from originals, or Postscript is acquired as part of the author's submission process. If the document has been scanned, the scanning station operator creates a scanned document record for the document.

Next, the documents are processed and placed on-line. Postscript versions are compressed and put into the proper directory. However, scanned documents are first processed into different resolutions and then stored in the proper directory. This process also generates a PIF, described below, for each resolution. Although the image processing was already part of the Library 2000 system, the PIFs were created specifically to support generation of digiments.

Finally, when a digiment is requested, the server runs the create-digiment program, which generates a proper digiment and returns it. The rest of this chapter describes the functioning of this program.

5.2 PIF Files

Part of the image processing process for scanned documents includes generating a Page Information File, or a PIF. The PIF file stores valuable information about an image, such as the format it is in, the page number of the image as the author intended, and a brief description of the contents of the page. These files are only generated for documents that have been scanned as part of the Library 2000 project.

5.2.1 Scanned document records

When each document is scanned, a human operator creates a scanned document record for the document. This record contains information about the document being scanned, the hardware and software used, and the name of the operator. Additionally, it contains a map of each page that has been scanned. For each file in a set of scanned images for a document, the scan record lists the name of the file, the size and checksum of the file, the contents of the file as determined by the human operator and any comments on that page that may be appropriate. This page mapping is used in the creation of PIFs. The scanned document record is more completely described in Appendix C.

5.2.2 Processing the scanned images

The original documents are scanned at 400 DPI, 8 bit greyscale, and stored as TIFF images. However, as each of these images takes approximately 14.5 MB of storage space, we process these images into versions more suitable for distribution. Currently, each image is converted into a 600 DPI monochrome TIFF image, suitable for printing, and a 100 DPI, 5 bit greyscale GIF image, which is designed for on-line browsing. This processing takes place automatically via a series of programs that read the scanned document record and convert each page to the proper formats.

5.2.3 Creating a PIF file

During this processing step a PIF is created for each of the different image formats. A PIF contains a listing of each of the scanned images as they were processed, which is generally the order in which they were scanned. For each image, the PIF lists several pieces of information. First is the image number, which is the number used to access the image from the server. Next is the format of the image. Third is the page number; this is either a number if the image is a numbered page, or the value "unnumbered" if the page has no page number or is an image that is not directly part of the document, such as a calibration sheet. This field is extracted from the content field of the scanned document record. Finally, the last field contains a description of the image; this is basically the content and the comment fields of the scanned document record concatenated together. Each field is separated by a tab; this allows spaces to be used within the fields. The first few lines of a typical PIF look like this:

```
Version:    1.0
Processing-comments: Tue Apr 11 09:57:37 1995
Image:   1  image/gif;dpi=100;bits=5   unnumbered  doccontrol
Image:   2  image/gif;dpi=100;bits=5   unnumbered  doccontrol
Image:   3  image/gif;dpi=100;bits=5   unnumbered  unnumbered;title page
Image:   4  image/gif;dpi=100;bits=5   2           numbered 2
Image:   5  image/gif;dpi=100;bits=5   3           numbered 3
```

A similar PIF would be created for the 600 DPI TIFF images.

5.3 The Create-digiment Program

The actual digiment is created by the create-digiment program. This program looks at the contents of a document's directory and any available PIF files to generate a proper digiment. This program is run whenever a client requests a document in digiment form from the server. Thus, any newly-added formats for a document will show up immediately when the next client requests the digiment.

The create-digiment program generates page-lists, page-maps, and body-lists, as well as a part-list and a bibliography part.

5.4 Generating Page-lists and Page-maps

The create-digiment program creates page-list and page-map digiment-types for each digiment for which PIF files exist. The create-digiment program calls a separate program, pif2digiment, which generates a page-map and page-list for each image format that exists for a particular TR. The pif2digiment program takes a PIF file, a MIME multipart boundary string and the ID of the document as arguments, are returns a page-list and a page-map application/digiment, separated by the multipart boundary. This program is run for each PIF associated with a document, creating a page-list and page-map for each image format.

The pif2digiment program scans through the entire PIF, generating a page-list and a page-map. Although these two parts are described separately, the actual processing takes place simultaneously so that the program does not need to scan through the PIF twice. When the program has scanned the entire PIF file, it prints out the page-map part, and then the page-list part, separated by an encapsulation boundary.

5.4.1 Generating page-lists

The page-list is created by first generating the standard MIME and digiment headers. Then the program generates a URL-stem that can be used to access the document. Since all of the documents are being served using the Dienst protocol, the URL-stem created uses this format. The following general URL can be used to access individual pages of a particular document that is stored on a Dienst server:

`http://`*server.name.here*`/Server/TR/`*technical-report-ID*`/Page/`*#*`?type=`*data/type*

The values in italics are replaced by the values as explained in Table 5.1 on page 52.

Value name	Meaning
server.name.here	The name of the Dienst server that stores the document
technical-report-ID	The CSTR ID for a technical report that is stored on the server
#	The image number requested
data/type	The format in which the image is requested

Table 5.1: Values for Dienst URL to access a page

The portion of the URL that is identical for all images, and therefore can be placed in the URL-stem header is everything up to the page number. So a typical URL-stem for a MIT TR would look like this:

```
URL-stem:   http://cstr-http.lcs.mit.edu/Server/TR/AI-TM-1066/Page/
```

Next, the content-type header is created. Since all the images are assumed to be of the same type for this program, the content-type from the first entry in the PIF is extracted and used for this field. The last header, page-map, contains the Content-ID of the page-map that is simultaneously generated from the same PIF file.

The rest of the page-list contains an entry for each image as specified in section 4.4.5 on page 40. The entry is constructed from the information in the PIF. The image number from the PIF is used as the VSN for the page. The remainder of the URL is constructed by concatenating the image number, the string "?type=" and the format type to complete a Dienst URL as described above. The page description field is created by looking at the image format; if it is a 600 DPI image, the field is entered with "Printable image". With 100 DPI images, the field contains "Viewable image".

5.4.2 Generating page-maps

Generating page-maps is very similar to generating page-lists. As the pif2digiment program is generating a line of the page-list, it generates the corresponding line of the page-map. Both lines are based on the same line of the PIF.

The first field in a Map: entry is the same VSN that is used in the page-list, and also comes from the PIF image number. The next two fields are the page content and page description fields. These fields are created by looking at the page number field of the PIF. If the page is a numbered page, the page content field is filled with the page number, as is the page description field. If the page number is unnumbered, then the entry depends on the description field of the PIF. If the description field contains “title page”, then the page number field becomes “title page”, and the page description field becomes "Title page”. If the description field contains “blank”, then the page number field becomes “unnumbered”, as does the page description field. Any other value in the description field causes the page content field to become “supporting”, while the page description field gets the value to the PIF description field. A summary of this is presented in Table 5.2 on page 53.

Coming from PIF		Entered into page-map	
Page number	Description	Page content	Page description
any number	anything	Page number field	Description field
unnumbered	title page	title page	Title page
unnumbered	blank	unnumbered	unnumbered
unnumbered	anything else	supporting	Description field

Table 5.2: Mapping from PIF to page-map fields

5.5 Generating the Other Digiment Parts

The create-digiment program also generates a bibliography part, a part-list part and a body-list part (if Postscript is available).

5.5.1 Generating the bibliography digiment-type

The bibliography application/digiment part is very simple to generate. All of the technical reports on-line have RFC1357 style bibliographic records stored in the TR's directory. The create-digiment program simply creates the standard MIME and application/digiment headers, and then appends the contents of the RFC1357 bibliographic file.

5.5.2 Generating the body-list digiment-type

Currently, a body-list part is only generated for those TRs that have Postscript versions available. If a Postscript version exists in the TR's directory, the create-digiment program first generates the standard MIME and application/digiment headers. Then the program generates a URL-stem that can be used to access the document. Since all of the documents are being served using the Dienst protocol, the URL-stem created uses this format. The following general URL can be used to access the entire body of a particular document that is stored on a Dienst server:

`http://`*server.name.here*`/Server/TR/`*technical-report-ID*`/Body?type=`*data/type*

The values in italics are replaced by the values as explained in Table 5.1 on page 54.

Value name	Meaning
server.name.here	The name of the Dienst server that stores the document
technical-report-ID	The CSTR ID for a technical report that is stored on the server
data/type	The format in which the body is requested

Table 5.1: Values for Dienst URL to access a page

The portion of the URL that is identical for all body parts, and therefore can be placed in the URL-stem header is everything up to the question mark. So a typical URL-stem for a MIT TR would look like this:

```
URL-stem:   http://cstr-http.lcs.mit.edu/Server/TR/AI-TM-1066/Body
```

Next, the content-type header is filled in. The current version of the program only allows a value of “application/postscript”. Finally, a single Body: line is created with a VSN of 1, a URL value of “?type=application/postscript”, and a Page Description field of “Postscript document”. Since the URL-stem line uniquely identifies the TR, this line is the same for all TRs and looks like this:

```
Body:    1     ?type=application/postscript     Postscript document
```

5.5.3 Generating the part-list digiment-type

The part-list digiment type is generated by the create-digiment program. After all of the other application/digiments that are part of the digiment have been generated, the program scans each digiment part and extracts the digiment-type and content-ID from each one. In the case of a page-list or body-list part, it also extracts the content-type. The program then creates a part-list by generating the standard MIME and application/digiment headers, then creating a Part: line for each application/digiment, consisting of the digiment type and the content-ID, followed by the content-type if applicable.

5.6 Putting It All Together

The digiment creation process is completely automated, and works for all MIT TRs which have bibliographic, Postscript or scanned images available on-line. The following is an outline of the complete process, from request to delivery of a digiment.

Since digiments are created on the fly, the process does not start until a digiment has been requested. This is accomplished by requesting the body of a document from the server with a type of multipart/digiment, the base MIME type for a digiment. Thus, a typical URL requesting a digiment might look like this:

```
http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Body?type=multipart/digiment
```

When the server is processing the request, it notices that the type requested is multipart/digiment. At this point, the server calls the program create-digiment, with the ID of the TR as an argument.

The create-digiment program first resolves the ID into the directory path where the document is stored locally. The program then checks the directory to make sure that it is actually a valid directory which contains a document. If not, it returns an error. If the directory does contain a document, the program continues by generating a bibliography part. This part is then put into a list of application/digiment parts.

The create-digiment program then looks at each of the subdirectories of the document directory to see if they contain a PIF. If a PIF exists for that subdirectory, the program inserts it into a list of PIFs. Then, the program calls pif2digiment for each PIF that was found. Each call returns a page-list and a page-map which was generated from that PIF. Each of these parts are put into the list of application/digiments.

Next, the program checks to see if a Postscript file exists in the document directory. If it does, the program generates a body-list part for the Postscript.

Finally, the program generates a part-list from all of the other application/digiment parts that have been created. It then creates a MIME header for the multipart/digiment document and returns this followed by each of the application/digiment parts, separated by the appropriate encapsulation boundaries. The server takes this return value, which is a legal multipart/digiment, and returns it to the client. The actual response time varies depending on the number of pages and formats of the document and the load of the server, but is typically on the order of a few seconds.

The entire process is summarized briefly below:

1. Client requests digiment from server.
2. Server calls create-digiment with ID of document.
3. Create-digiment validates ID and local storage of document.
4. Create-digiment gets a list of PIFs which are available.
5. Create-digiment generates a bibliography part.
6. Create-digiment generates a page-list and page-map for each PIF by calling pif2digiment.
7. Create-digiment generates a body-list if Postscript is available.
8. Create-digiment generates a part-list.
9. Create-digiment returns the application/digiments in a multipart/digiment format.
10. Server returns the multipart/digiment to the client.

Chapter 6

A Prototype Digiment Browser

This chapter describes a browser that is able to read digiments and turn them into a form that is able to be viewed by any WWW capable browser.

6.1 Basic Requirements for a Browser

Any digiment browser must have several basic requirements. Any browser that wants to display digiments in a useful way should implement these features. Although some are more important than others, these are the features that make use of the extended information available in a digiment.

6.1.1 A browser must understand the digiment format

Any browser must be able to parse the digiment format. This means being able to interpret the six application/digiment types defined in this document and extract the relevant information from them.

6.1.2 A browser should be able to use helper applications

A browser does not need to know how to use or display every possible format that may be contained within a digiment; this would be nearly impossible, as a digiment can contain data in arbitrary formats. However, it should be able to recognize the MIME type of a data object and pass the object to a user assignable application that does know how to display that data type. This is much like how most WWW browsers work today.

6.1.3 A browser should understand relationships between parts

A browser should be able to understand the relationships between parts. This means that it must be able to keep track of Content-IDs and VSNs which one part of a digiment uses to refer to another. In addition, it must be able to resolve those relationships, such as knowing which pages are substitutable for the same page in another format.

6.1.4 A browser should show relevant bibliographic material and meta-information

A browser should show appropriate bibliographic material to the user. This may include the copyright, title, and abstract. In addition, the browser should be able to show meta-information about the document, such as page numbers, page descriptions, processing comments, etc.

6.1.5 A browser should let you select between different formats

If a digiment has data objects that are in multiple formats, the browser may choose a default type to use for browsing. However, it should be able to let the user choose between all available versions at any time.

6.1.6 A browser should let you choose which page to view

A browser that is browsing a page-list should allow the user to choose from any page in the page list and go to it directly.

6.1.7 A browser should let you browse a document in the correct order

A browser should be able to understand the intended order of page-list, body-list and preferred-order parts. It should then be able to provide previous and next buttons to allow the user to browse the document in the correct order.

6.2 Creating a WWW Based Browser

The browser that I created to view digiments does not display the digiment directly. Instead, it is simply a program that reads in a digiment and creates HTML documents that can be viewed with any WWW browser. There are several advantages to this scheme, as well as several drawbacks, however.

6.2.1 Advantages to using a WWW based browser

There are two primary advantages to using a WWW based browser. The first is that by using the WWW browser as the display engine, I do not have to write another front end.

Netscape and Mosaic can both handle GIF images in-line, and Netscape can also display JPEG images in-line. In addition, both can display formatted text via HTML. Furthermore, they can also display buttons and text entry fields, also via HTML. This frees the browser from having to include code for displaying text, images and user interface objects.

Secondly, the browser can be used on any platform that has a WWW browser ported to it. This includes almost every type of workstation and PC used in the world. Since the actual browser code does not have to run on the machine that the user is using, the only program the user has to run locally is a WWW browser. This enables a much wider audience than if the browser had been written for a single platform.

Another advantage is that the browser itself can run on any machine with a HTTP server. In fact, the browser code itself currently runs on the same server that the documents are stored on.

6.2.2 Disadvantages to using a WWW based browser

There are a few disadvantages to using a WWW based browser, however. One of the biggest is the lack of control over the interface. Most browsers are only able to in-line GIF images. In addition, HTML does not allow for the automatic loading of non-in-line data types. This means that anything that the browser cannot display internally, such as TIFF images, Postscript files or audio clips cannot automatically be displayed with the rest of the page. Thus, the browser can only present a page with another button for the user to click in order to actually view the data.

6.3 How the browser works

CGI, or Common Gateway Interface, is a standard for interfacing external programs with HTTP and other information servers. The browser itself is actually a CGI program that is invoked by passing a special URL to the server that is running the browser. This URL contains all the arguments necessary for the browser to determine its current state and what the user has requested. The browser interprets this URL, generates the correct HTML document that displays the requested information, and returns that to the client browser to display.

6.3.1 How to access the browser via URL

The browser is invoked by passing URLs to the server that runs it. These URLs use the CGI convention for passing parameters to a program. The way to do this is to embed a "?" in the URL; everything after that is interpreted as parameters to the program being run. The parameters take the form of "**parameter**=*value*" and are separated from each other by a "+" sign. It is valid for a parameter to not have a value. Thus, a sample call to a CGI program would look like this:

`http://`*server*`/`*path*`/`*program*`?param1=foo+param2=+param3=100`

Where *server* is the name of the server, *path* is the path to the CGI program and *program* is the name of the CGI program itself. Following the "?" are three parameters; param1, param2, and param3 with values of foo, the empty string, and 100, respectively. When the server receives a URL of this form, it calls the program with the parameter string following the "?". A URL of this form can either be constructed by another program, that knows how to form CGI style URLs, or it can be generated by a browser that knows how to use HTML forms.

The browser described in this document takes six different parameters to specify the what state the browser should be in. Since the browser is executed from the beginning each time the user access it, it has no way of keeping track of the user's current state other than embedding this information in the URL. Once the user has initially accessed the browser via the standard entry point, any URL generated by the browser for the user to use to navigate will contain values for these parameters that allow the browser to determine what its current state should be. A summary of these parameters can be found in Table 6.1 on page 64.[1]

The first parameter is **getmethod**. This parameter is used to determine where to obtain the digiment being browsed, and can have one of three different values; id, url, or local. If the value is id, then the browser uses a RFC1357 style id to obtain a digiment from a Dienst server. If the value is url, the browser uses an arbitrary url to obtain a digiment. If the value is local, the browser obtains the digiment from a local cache.

The second parameter is **id**. This parameter contains the RFC1357 style id that is used to obtain a digiment if the getmethod parameter is set to id.

The next parameter is **url**. This parameter contains a url that will return a valid digiment when referenced. This value is used when getmethod is set to url.

The fourth parameter is **pid**. This is a local reference number used when caching digiments that have already been obtained via one of the other methods. This value is used when getmethod is set to local.

The fifth parameter is **vsn**. Its value is the VSN of the digiment part that the user wishes to view.

The final parameter is **cid**. This parameter contains the Content-ID of the current digiment part that is being browsed. This part can be either a part-list, a body-list or a preferred-order. The combination of this and the vsn parameter can uniquely identify the data object that is being requested.

1. There is actually another parameter, debug, which turns on debugging when set. This parameter is only designed to be used for testing purposes, and is not available from the standard interface.

URL parameter	Parameter meaning
getmethod	Which method to use to obtain digiment
id	ID of the digiment
url	URL of the digiment
pid	Reference ID for local caching of digiment
vsn	VSN requested
cid	Content-ID of part being browsed

Table 6.1: Parameters to the digiment browser

6.3.2 Obtaining a digiment and local caching

A digiment is obtained by the browser in one of three ways. If getmethod is set to id, the browser contacts the Dienst server that serves the document and requests the document in digiment form. If getmethod is set to url, the browser resolves the url, which is expected to return a digiment. Finally, if getmethod is set to local, the browser attempts to read the digiment from a local cache.

Whenever the browser obtains a digiment from an ID or URL, it also caches the digiment locally in the /tmp directory with a name based on its current process ID, or pid. This pid is the same value used in the pid parameter in the URL. When the browser is called with getmethod set to local, it looks at the pid parameter and checks the /tmp directory to see if the digiment corresponding to that pid is cached there. If it is, it reads the digiment from the cached file. If not, the browser resorts back to the id and url methods respectively. This can improve performance significantly since the digiments are created on the fly. This means that every call to the browser would result in a digiment being created again if the browser requested the digiment from the server each time. By caching the digiment locally, the browser can eliminate this overhead.

6.4 Parsing the Digiment

Once the browser has obtained a digiment, it parses it and looks at the parameters to determine what to display.

First the browser checks that the data it has obtained is a digiment with a valid MIME Content-Type field of multipart/digiment. If not, the browser returns an error. Otherwise, the browser extracts the boundary parameter from the multipart/digiment identifier.

Next, the browser splits the multipart/digiment into its component application/digiments by using the boundary string that it previously extracted. Each of these digiment parts is then placed into an associative array, keyed by its Content-ID. This allows the browser to look up any digiment part based on its Content-ID.

The browser then parses the bibliography part. This basically involves looking for the tags for the title, date of publication, organization, notes, abstract, and authors, and extracting the data from each field. This data is then stored in global variables so all parts of the program can access them for display purposes.

The browser also parses the part-list. This involves scanning each line of the part-list and extracting the digiment-type and content-ID from it. The content-IDs for each part-list are stored in an array, as are the content-IDs for each body-list and for each preferred-order. This allows the browser to quickly access each of the part-lists, body-lists or preferred-orders for a digiment.

6.5 Displaying the Digiment

Depending on the state of the browser, it will display one of two types of interfaces. If browser parameters do not specify a specific VSN to view, which is the default when first accessing the browser for a specific document, the browser will display the digiment summary screen. If the parameters specify a VSN, then the browser displays the page or body object associated with that VSN and the content-ID from the parameter list.

6.5.1 Displaying the summary screen

The summary screen is the main entry point to the browser. This screen shows the bibliography information that was extracted from the digiment. This consists of the document's title, its authors, the publishing organization, the abstract, and the contents of the notes field.

The browser then displays a list of options for the user to choose from. First, it will display any preferred-order parts that exist, instructing the user that these are the preferred ways to browse the document. Then it will display any body-list parts that exist, and finally it will display any page-list parts.

Each of these parts is actually a hypertext URL that encodes the correct parameters to instruct the browser to display the correct digiment part. This is done by setting the cid parameter to the content-ID of the part that the user selected, and the vsn parameter to the first VSN in the part.

One exception to this is with page-lists. In this case, the vsn parameter is not set to the first VSN in the list, but actually is set by looking at the page-map associated with that page-list and selecting the VSN of the first page that is not of type "supporting". This allows the user to skip over the supporting pages, which are usually not of interest.

6.5.2 Displaying part of the digiment

If the vsn parameter contains a value, the browser displays the data object that is referenced by the combination of the content-ID from the cid parameter and the VSN from the vsn parameter.

The browser first displays a row of buttons that allow the user to navigate through the digiment. The first four buttons will take the user to the title page (or first page if there is no title page), the previous page, the next page, and the last page, respectively. If the browser is displaying data from a body-list instead of a page-list, then it refers to sections instead of pages. The next button takes the user back to the summary page. The browser then displays a pop-up list that allows the user to select any page or section available. The browser also displays the name of the current page, and its page or body description.

Once again, the buttons actually represent URLs that encode the VSN of the correct page or body section to display. In the case of page-list types, any page that is labeled as a supporting page in the page map is excluded from the title, previous, next, and last buttons. However, these pages can be viewed by using the pop-up menu to go to them directly.

If the type of data being displayed is a GIF image, the browser inserts the image into the HTML document, where it will be displayed by the WWW browser. Any other data type cannot be displayed directly, however. In this case, the browser generates a button that the user can click on to load the specified image or body section.

Finally, the browser generates another set of buttons below the image or link that are identical to those at the top of the page. This is useful when the user is viewing images that require him to scroll to the bottom of the page.

6.6 Limitations of the Current Browser

The current version of the digiment browser has some limitations, which we hope can be fixed in a later revision. Some of these limitations are inherent to using the WWW as an interface for the browser, however. The most notable of these limitations in the inability to create complete pages for any type of data other than text or GIF images. Any other type of image or data, such as Postscript, must be loaded by a separate action by the user. This is a limitation of the current HTML 2.0 specification and/or the currently available WWW browsers, depending on your point of view.

The browser also does not support preferred-order digiment parts, although it would not be difficult to modify it to do so. The browser also does not do extended error checking and recovery, which would be a essential if the browser were to be used in a service environment.

Chapter 7

Future Directions

This chapter describes improvements that could be made to the current prototypes of the digiment generator and browser, as well as extensions and modifications that could be made to the digiment format.

7.1 Improving the Digiment Generator

The current prototype of the digiment generator has some shortcomings, and could be improved in a number of ways. Implementing some of these improvements and features would make the generator much more practical for other institutions and third parties. Although the generator was designed to work specifically with TRs from the MIT Library 2000 project, it was also built with modularity and expansion in mind.

7.1.1 Reducing dependence on MIT server structure

The biggest obstacle to making this software available is reducing its dependence on the MIT TR structure. The digiment generator relies on knowledge of the local filesystem storage structure, as well as file naming conventions in order to determine the types of data objects available for a digiment. The generator also requires PIF files to be available for any image formats. By abstracting out these dependencies into modules, this software could work with any type of storage structure with the creation of an appropriate module. The current system already has some abstractions, but no modules have been created for any environment other than MIT.

7.1.2 Recognizing other body-list types besides Postscript

The current digiment generator only creates body-list parts for Postscript versions of a document. Since the MIT TRs that are on-line only contain Postscript and/or images, this was not a problem for the initial version of the generator. However, once we start doing

OCR on the scanned images, we will have text versions of the documents available, and possibly word positioning information as well, depending on the type of OCR used.

Furthermore, other sites may make use of PDF, XDOD (Xerox Document On Demand), or other formats. The generator should be able to be easily modified to recognize the existence of any available format and generate the appropriate body-lists.

7.1.3 More intelligent merging of page-maps

The current digiment generator only does a very simple merging of page-maps. If two page-maps are identical, it will merge them into a single page-map and update any page-lists that referenced them to point to the new page-map. This method works well for scanned images where all the available formats are derived from the same original images and therefore have the same page-mapping structure. However, this system breaks down when trying to relate a single color image to a list of greyscale images. The page-map for the color image will consist of a single entry with the VSN and page number of the color page. The greyscale page-map will consist of entries for each page in the page-list. Since the two page-maps will not be identical, they will not be merged.

The appropriate action to take would be to intelligently merge the two page-maps. This would involve examining the page number from the color image and finding the entry with the same page number in the greyscale page-map. If the two VSNs are identical, then the greyscale page-map can be used for both versions. Having the extra greyscale entries in the page-map would not cause a problem for the color page-list, since any entry that is not in both the page-list and page-map is ignored.

If the two VSNs are not identical, the generator would have to resequence one or both of the VSN series in order to ensure that the color and greyscale image have the same VSN. As long as the new VSNs are in the same relative order, this does not create any problems.

7.1.4 Ability to make page-lists for images without PIFs

Another nice feature would be to simulate a PIF for sequences of images that do not have a PIF associated with them. As long as the images could be ordered in some non-arbitrary way, such as by examining the filename, a pseudo-PIF could be created for them. This PIF would be in the same format as a normal PIF, but would have the value "unknown" for every field other than the image number. This would allow the generator to construct a page-map and page-list for the images, so that they can be viewed, but without any page numbering or cross-referencing of types available.

7.2 Improving the Digiment Browser

The current prototype of the digiment browser does not take full advantage of the digiment format. In addition, using the WWW as a front end has several inherent limitations, most notable the lack of control over the interface. By improving the browser and integrating it with its own interface system, it could be a useful tool for viewing digital documents.

The single most useful enhancement would be to convert the browser from a WWW based system to a stand alone system. This would permit the browser to have complete control over the interface system, instead of relying on HTML and a WWW browser. Most notably, the digiment browser could incorporate code to allow it to handle multiple image types as well as other types of data internally. This would allow the browser to display these pages simultaneously with the rest of the interface, instead of the current system that requires the user click on an additional button to display the data for a specific page.

In addition, a stand alone browser could support multiple windows that it could update at will. Besides the window with the actual document data in it, possible other interface windows could include a window with a list of available pages, a window with bibliographic information for the current document, a window with a list of all available data parts, a window with a list of alternative formats for the current part, a list of recently viewed digiments, or simply multiple data windows at the same time. Since the browser controls the windows directly, it could update the information in them when the user changed pages, data types or digiments.

Trying to fit all of these possible interface options into the single window provided by a WWW based browser would result in a very cluttered interface that would be confusing to the user.

A stand alone browser would also have the feature that it could be used as a WWW helper application, allowing it to fit seamlessly into the existing WWW structure. When a user who is using a WWW browser clicks on a link that points to a digiment, the WWW browser can recognize that the returned object is a digiment and automatically launch the digiment browser. Thus, a user could search for a particular article on-line, click on one of the results, and have the digiment pop up in a new window on his workstation.

7.3 New Extensions and Enhancements to the Application/digiment Type

The digiment specification outlined in this document provides for the basic set of digiment-types that are necessary to display most forms of data, as well as carry bibliographic information and relationships between parts. However, this is not an exhaustive set of types and new digiment-types are expected and welcomed. Some examples of possible new digiment-types are listed below.

7.3.1 A pricing digiment-type

A pricing digiment-type could contain retrieval pricing information for each data object contained within a digiment. This would allow a digiment server to charge for the information it serves, as well as providing real time pricing information to clients. A possible format for this type would be to simply list each data object by content-ID and VSN, along with the cost for retrieval. Users could then choose which version of a digiment to view depending on their preferences for how much they wish to pay. For example, a plain text version and a Postscript version could be charged at different rates.

This would also allow users to request the same digiment from multiple servers to compare pricing structures versus the formats offered and the speed of delivery.

7.3.2 A distribution-rights digiment-type

A similar digiment-type could be created that specified distribution rights or copyrights for each data type. Two different versions of the same information may not have the same copyright or distribution rights. For example, a text version of a document may have unlimited distribution rights while the Postscript formatted version may be restricted. Or a PDF version of the document which is view-only may have different distribution rights than a fully editable version. In addition, this type could be combined with the pricing type to allow companies to charge different amounts for copies of the same document with different distribution rights.

7.3.3 A digiment-structure digiment-type

This data type could contain information specifying the structural layout of a digiment. This could entail containing a list of different document structural parts, such as chapters, appendixes and multimedia annotations, associated with the correct items from the part-lists and body-lists. A part like this would allow a browser to construct a Table of Contents for a digiment, which the user can use to view the digiment.

7.3.4 Enhancing the preferred-order type

Currently, the only description of what a preferred-order type contains is in a human-readable string. It may be useful to add another, structured, label to the preferred-order type, which can be parsed by a program. This would allow a program to automatically scan through the available preferred-order types and pick the most suitable one to use. However, this would require enumerating all possible ways in which preferred-order types could differ, so that a program can determine which one is best suited to its needs. This is not an easy task.

7.4 An Alternative Form for the Page-list Digiment-type

One of the problems with the current form of the page-list is that it does not actually contain the data, it simply references it. While this is desirable for many applications, such as a user desiring simply to browse a document that is stored centrally, it is not desirable for a electronic document delivery systems, which want to deliver a completely self contained document in a single transaction. In addition, although the MIME specification allows the embedding of either a data object, or a reference to a data object stored elsewhere, the current page-list definition uses its own method of referencing external objects, so MIME compliant agents cannot automatically extract the pages. This is because I determined the MIME version of referencing to be too inefficient and too complex for this application. However, it is possible to build page-lists that completely conform to the MIME specification and allow the option of either referencing or embedding the data object. A program that could build page-lists of this form was created along with the standard page-list generator described earlier, but is not being used.

This can be done by storing a page-list as a multipart MIME object, with each page in the list stored as a separate object. Each object has a standard MIME Content-Type and Content-ID. The value of the Content-ID header is used in place of a VSN. If the object is embedded into the digiment, the Content-Type headers contains the MIME type for the data object. If the object is stored externally, a special MIME type of "message/external-body" is used, which is defined by the MIME standard to reference externally stored data objects. Any client that is MIME compliant can resolve this reference and obtain the data object itself.

Using a page-list of this type, a digiment could either reference the data objects that are stored on the server, or embed the data objects within the digiment in order to transfer the entire document in a single transaction. However, a simple digiment containing 50 images in only two different formats would take approximately 30 kilobytes, as opposed to only 5 kilobytes with the current page-list.

This system could also be modified to work with body-lists as well as page-lists.

7.5 The Digiment as Interchange Format

One of the most useful properties of a digiment is the ability to represent an entire document, including all of its representations and variations. The digiment representation is independent of the actual formats used to store the document on a particular server. By promoting this format as an interchange standard, it would be possible to use the digiment format to transfer entire documents from one machine to another. This transfer may be accomplished in a number of ways. One way would be to transfer first the digiment object, then each of the files which it references, by traditional protocols, such as FTP, HTTP or even email.

Another way would be to create a new page-list type, such as the one described in section 7.4 on page 74, which embeds the actual data in the digiment. The digiment could then by transferred by traditional protocols. The advantage of this scheme is that the entire document could be transferred at one time, rather than transferring each of the data files separately.

However, the digiment format could also be built into existing protocols. For example, FTP has an ASCII mode, which reads text in from the host system, converts it into netascii format, and then converts it back into the proper ASCII format on the client system. This conversion to an intermediate format and back is necessary to properly deal with the way different systems signify end of lines. A digiment mode could be added to FTP, which would automatically convert a document into digiment format, transfer it to a client system, and convert it into a format suitable for storage on the client system.

A digiment transfer mechanism should also let the client specify a subset of a digiment. For example, a client should be able to request a digiment, but only the 100 DPI, 5 bit images. The mechanism should then only include the data in the page list which corresponds to the requested type. In addition, a client should be able to request a digiment in a specific preferred-order. In this case, the digiment would only include those subsets of the page-lists and body-lists which are actually referenced by the preferred-order.

A digiment can be made even more portable by using MIME encoding methods. These methods are defined in the MIME standard to permit the transfer of arbitrary binary data between different systems and gateways. By encoding and decoding each of the data parts appropriately, a program can transfer a digiment from one system to another without having to worry about the specific storage formats on either system, or what gateways the data had to pass through.

By either writing programs that use traditional protocols, creating new protocols, or adding new types to existing protocols, it would be possible to transfer any document to another system in the digiment format. Such a document interchange format would be invaluable for transferring documents between repositories or for transferring a new document from a publisher into a repository.

Chapter 8

Insights and Lessons Learned

This chapter describes the insights gained by building a model digiment browsing system.

8.1 General Conclusions

The digiment differs from other digital document systems in two key areas; making the distinction between the semantic content of a document and its representation, and creating a compound object that contains structural elements specifically designed for documents. Although some other systems implement one or both of these ideas to some extent, they focus on one more than the other, and in all cases neither of these ideas is central to the system.

In addition, several other key concepts are valuable when making a digital document system. The idea of a document ID, or handle, which does not point to a specific representation of a document, but instead references a collection of representation, is essential to deal with a document which may come in multiple forms. The use of MIME types allows easy integration into existing standards, such as the World-Wide Web. Finally, the ability of a digiment to include the content by reference instead of embedding it allows all the necessary information to browse a digiment to be sent with a very low overhead. The browser can then choose which content parts it wishes to transfer at its convenience.

Although the digiment of the future may be based on other standards, such as OpenDoc, the key concepts and ideas that went into this system will be valuable in creating a generic, expandable system for transferring and viewing digital documents.

8.2 The Distinction Between the Semantics of a Document and its Representation

One of the most important ideas that was examined is the distinction between the semantic content of a document and its representation. Most systems that deal with "documents" either explicitly or implicitly define the document in terms of its representation. This leads to the problem of defining a GIF image version of a document and a TIFF image version to be two separate documents. However, this thesis explores the idea that the true definition of a document is should not be linked to a specific representation, but is rather the abstract information that the document is conveying. This idea does not just apply to different image formats, or Postscript versus text, but can be expanded to include any notion of "sameness". Thus, both English and French language versions of a particular text may be considered the same "document" in terms of the information that it is trying to convey.

Once we have separated the semantics of a document from its representations, we can talk about a document without having to choose a specific representation. However, to manipulate these new types of documents on-line, we need to define a new type of digital object. This new object is a digiment. The digiment contains all of the semantic information about a document, such as bibliographic information, as well as all of the representations of the document that are available. By referring to digiments, people can pass around the abstract notion of a document without having to be concerned with the representation, which is irrelevant to the information that the document is presenting.

8.3 A Compound Digital Object Specifically Designed for Electronic Distribution and Browsing

The other important idea that has come out of this work is the usefulness of a compound digital object that is specifically designed for electronic distribution and browsing. Systems such as OpenDoc and OLE allow the creation of compound objects that contain arbitrary data types. An OpenDoc program can then work with these objects by manipulating the data types that it recognizes, while letting other OpenDoc programs manipulate the types that it doesn't. The digiment improves on this concept by defining an object that is structured in such a way that it preserves the types of relationships among parts that are necessary for browsing a document. In fact, digiments could conceivably be stored as OpenDoc documents with the appropriate additional information stored along with the different representations.

A true document does not just consist of arbitrary data types which can be viewed in any order. Documents have notions such as “next” and “previous”, which determine the order in which information should be presented. In addition, documents have the notion of “pages” and “sections”. The digiment allows its contents to be structured into pages and sections, as well as preserve the “next” and “previous” relationships between the pages and sections. In addition, the digiment extends this structure to alternate versions of the same data. For example, page 12 may have greyscale and color images, while page 13 may have greyscale and color images, and a Postscript version of that page.

Furthermore, documents contain relationships between objects and sub-objects. Page 12 may have a figure embedded in it, which is also available in a separate, expanded image. A Table of Figures part could list all the figures available.

By defining structural elements that are specifically designed for documents, programs can edit and view digiments while preserving the actual structure of the document.

Appendix A

The multipart/digiment and application/digiment MIME Types

A.1 The Multipart/digiment Type

The multipart/digiment is the base MIME type for a digiment. When transferring a digiment, the type used should be multipart/digiment. The multipart digiment is syntactically equivalent to the multipart/mixed type, but with different semantics. The only type allowed within a multipart/digiment is application/digiment. The use of multipart/digiment instead of multipart/mixed is to differentiate a digiment from a random multipart message. This allows MIME agents to detect a multipart/digiment and invoke the proper action to process it. When referring to a "digiment", we are actually referring to the multipart/digiment object which contains all the other data objects; the whole is known as the digiment.

A multipart/digiment consists of a number of application/digiment objects, which are defined in the following section. A minimal multipart/digiment consists of a application/digiment of type page-list, followed by one or more application/digiment parts of varying types. Any object which is not of application/digiment MIME type may be ignored.

A.2 The application/digiment type

The application/digiment type is the meat of the digiment. A digiment is made up of a collection of application/digiments which describe the available data formats for the digiment and the relationships between them. Each application/digiment consists of two MIME headers; a Content-Type: of application/digiment, and a Content-ID with a globally unique ID. These headers must always be present. Following these headers is a blank line and the body of the application/digiment. Thus, the beginning of an application/digiment would look like this:

```
Content-Type:  application/digiment
Content-ID:    MIT-LCS//MIT/LCS/TR-341.map

start of application/digiment body
```

There a multiple types of application/digiments. Each application/digiment describes some data of the digiment, or a relationship between different data types. The type of application/digiment is specified in the application/digiment headers.

A.2.1 Application/digiment headers

Each digiment-type has the same first two lines as a header. The first line must contain the tag **Version:**, followed by the version number. This document describes version 1.0 of each digiment-type. The only current legal version number is 1.0. Any future version may describe a different format and must be described in a separate document. The second line must contain the tag **Digiment-type:** followed by a legal digiment-type. The currently defined types are described below, as well as being listed in Table A.1 on page 91. Thus, a correct part-list application/digiment would start like this (including the MIME headers):

```
Content-Type:  application/digiment
Content-ID:    MIT-LCS//MIT/LCS/TR-341.parts

Version:       1.0
Digiment-type: part-list
```

The rest of the application/digiment body follows immediately afterwards, without any blank lines.

A.2.2 The part-list type

The part-list type is the only required type of application/digiment in a digiment, and must be the first part in a digiment. This type contains a listing of each of the other application/digiments in the digiment, with their Digiment-type and Content-ID. In addition, types which contain data, such as a page-list or body-list, may also have the MIME type of the data included as a hint.

The part-list type consists of the application/digiment headers followed by a line for each of the other application/digiments. Each line starts with the tag **Part:**, followed by the digiment-type of the part, followed by the Content-ID of the part. For parts that contain references to data, such as page-list or body-list, the format of the data can also be included. Each of these values is separated by a tab, and the line is terminated by a newline. Each application/digiment object contained within the digiment must have its own **Part:** line within the part-list. A part-list for a digiment with a bibliography, a single page-list and a page-map would look like this:

```
Version:          1.0
Digiment-content: part-list
Part:    bibliography   MIT-AILab//AIM-1066.bib
Part:    page-map       AIM-1066.100x5cLZW.map
Part:    page-list      AIM-1066.100x5cLZW.list image/gif;dpi=100;bits=5
```

A.2.3 The bibliography type

The bibliography type is meant to contain bibliographic and other meta-data about the document which is not otherwise found in the body of the document. It consists of the standard application/digiment headers followed by bibliographic information about the document in RFC1357 format. This format generally consists of a tag, followed by two colons (::), followed by the data for that tag. The data continues until the start of the next tag. A sample bibliography type might consist of the following:

```
Version:          1.0
Digiment-content: bibliography
 BIB-VERSION:: v2.0
          ID:: MIT-AILab//AIM-1050a
       ENTRY:: February 27, 1995
ORGANIZATION:: Massachusetts Institute of Technology, Artificial
     Intelligence Laboratory
       TITLE:: What Are Plans For?
      AUTHOR:: Agre, Philip E.
      AUTHOR:: Chapman, David
        TYPE:: Technical Memo
        DATE:: September 1988
       PAGES:: 18
       GRANT:: N00014-85-K-0124
    ABSTRACT:: What plans are like depends on how they're used. We
   contrast two views of plan use. On the plan-as-program-view, plan use
   is the execution of an effective procedure. On the plan-as-
```

```
communication view, plan use is like following natural language
instructions. We have begun work on computational models of plans-as-
communication, building on our previous work on improvised activity
and on ideas from sociology.
      END:: MIT-AILab//AIM-1050a
```

A.2.4 The page-list type

The page-list type is the main digiment-type used for specifying page-based data. Page-based data refers to any data which is accessible a single page at a time. This type may be used for any data type which is available a page at a time. For example, a document may be available in Postscript form, which would be specified in a body-list. Normally, a client would have to download the entire Postscript file to view a single page. However, a server may be able to serve a single page of a Postscript version of a document at a time. The entire Postscript version could then be listed as a page-list and accessed a page at a time, permitting additional information to be assigned to each page and allowing finer control of the relationships between different versions. This also has the advantage of not requiring an entire body-list part to be transferred to view a single page, making for more effective use of bandwidth.

Each distinct data format should have its own page-list, even if it does not contain an entry for each page. For example, if a digiment contained pages 1-15 of a document in greyscale GIF, but also included a separate copy of page 12 in color GIF, the digiment would have two separate page-lists. One would contain a listing for each of the pages 1-15 available in greyscale, while the other would only have an entry for page 12, which is a color GIF. The relationship between pages which contain alternate forms of the same data is maintained in a separate part called a page-map.

The page-list type starts with the standard application/digiment headers. These are followed by three more headers and a list of pages available in the specified data type. The first header is **URL-stem:**, followed by the stem of a URL that can be used to access a page. By placing the first part of a URL in this header, it is not necessary to repeat a part of a URL that is common to all the pages of a given type. This stem is prepended to the URL given for a specific page to form a complete URL. If this header is empty or missing, the URL used to access a given page is simply that URL which is given for that page.

The next header is **Content-Type:**, followed by a MIME content type/subtype that describes the format of the pages. Each page in the page-list must be in the format described by the Content-Type header. This header is required.

The final header is **Page-map:**, followed by the Content-ID of a application/digiment of type page-map which this page-list uses. If this header is missing, the digiment browser should assume an implied page-map with all entries as “unknown”. The page-map type, described in section A.2.5, contains various meta-data about an individual page, as well as preserves relationships between pages which contain the same data, but in different formats.

Following the headers is a list of pages. Each page has a single entry in the page-list. An entry begins with the tag **Page:**. This is followed by a **Virtual Sequence Number**, or **VSN**. The VSN uniquely identifies a page within a given page-list, and is assumed to have some in-order. That is, a page with a VSN less than a second page should be displayed before the page with the higher VSN and vice-versa. VSNs do not have to be sequential, but must be have some absolute relative ordering. Following the VSN is a **URL** which can be used to access the page data. If the URL-stem: header is present and non-empty, the contents of that header will be prepended to the URL, so the URL field does not need to be a valid URL by itself; however, the combination of the URL-stem: field and the URL field must create a valid URL which can be resolved to access the page data. The URL field is followed by a **Page Description** field. This field is free text and may contain any of the characters [a-zA-Z_-] and punctuation characters. The page description field is displayed by the browser to indicate which version of a page a user is looking at, and which others are available. For example, page 12 which is in greyscale format might have a page description field of “Standard image”, while a color version of page 12 might have a field of “Color image”, while an extremely low resolution version might have a field of “Quick preview”. Each of these fields is separated by a tab and the line is terminated by a newline.

The following sample shows a page-list which specifies 5 pages in GIF format. The URL field of each page is appended to the URL-stem: header to form a valid URL to access the specified page. This page-list points to a page-map which is also contained within the digiment; this page map contains additional information about each page. Finally, each page is labeled with a page description of “Viewable image”. This text will be shown by the browser in a button which can be used to select among different versions of the same page.

```
Version:           1.0
Digiment-content: page-list
URL-stem: http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Page/
Content-type:      image/gif;dpi=100;bits=5
Page-map:          AIM-1066.100x5cLZW.map
Page:    1         1?type=image/gif;dpi=100;bits=5     Viewable image
Page:    2         2?type=image/gif;dpi=100;bits=5     Viewable image
Page:    3         3?type=image/gif;dpi=100;bits=5     Viewable image
Page:    4         4?type=image/gif;dpi=100;bits=5     Viewable image
Page:    5         5?type=image/gif;dpi=100;bits=5     Viewable image
```

A.2.5 The page-map type

The page-map type includes extra data about a specific page, and is used to relate different versions of the same page together. Multiple page-lists can share a single page-map; this is how multiple pages can indicate that they refer to the same virtual page. Page-maps are also used to describe the type of page, as well as provide the page number.

A page-map consists of the standard application/digiment headers followed by a mapping for each page in the page-map. Each page has a single entry in the page-map. The entry begins with the tag **Map:**, which is followed by a **VSN**. This VSN corresponds to the VSN for a specific page in every page-list which points to this page-map. For example, if page-list A pointed to page-map A', the page with VSN 12 in A would correspond to the map with VSN 12 in A'. In addition, if page-list B also pointed to page-map A', the page with VSN 12 in A would be considered an alternate version of the page with VSN 12 in B. It is permissible for multiple page-lists which all point to a single page-map to have VSN series that do not completely overlap, as long as those VSNs that do overlap refer to the same page in alternate formats and the VSNs that do not overlap refer to pages which are only available in certain formats. This allows flexibility in specifying relationships between pages in various formats, where all the pages may not be available in each format.

The next field after the VSN is the **Page Type** field. This field can hold one of five values; supporting, title page, unnumbered, an integer number, or unknown. This field can be used by the browser to determine what type of logical page this is. A value of supporting means that the page is not really part of the document, but is used in processing or contains calibration or miscellaneous information. These pages are not normally displayed or printed. A value of title page indicates that the given page is the first or title page for the given digiment. A value of unnumbered means that the page has no page number, while an integer number refers to the actual page number in the document. Finally, unknown indicates that there is no logical information about this page.

The last field is the **Page Name** field. This is simply displayed by the browser to indicate which page is currently being viewed. Normally this field will contain the page number, but it may contain "Title page", or a description of a processing control or calibration page.

The following example is part of a page-map. Notice that the first page, with VSN 1, is a supporting page of type "doccontrol", while the second page, with VSN 2, is a IEEE-

167A-1987 calibration target. The actual digiment does not start until VSN 3, which is the title page.

```
Version:           1.0
Digiment-content: page-map
Map:      1        supporting     doccontrol
Map:      2        supporting     calibration IEEE-167A-1987
Map:      3        title page     Title page
Map:      4        1              1
Map:      5        2              2
Map:      6        3              3
```

A.2.6 The body-list type

The body-list type is the main digiment-type used for specifying data segments that contain multiple pages, or are not page based. For example, a Postscript version of a document would be placed in a body-list. However, sometimes a segment may contain only part of a document, such as a Postscript file for each chapter in a document. In this case, a body-list will contain an entry for each file which makes up the document.

The body-list type starts with the standard application/digiment headers. These are followed by two more headers and a list of segments that are available in the specified data type. The first header is **URL-stem:**, followed by the stem of a URL which can be used to access a segment. By placing the first part of a URL in this header, it is not necessary to repeat a part of a URL which is common to all the segments of a given type. This stem is prepended to the URL given for a specific segment to form a complete URL. If this header is empty or missing, the URL used to access a given segment is simply that URL which is given for that segment.

The second header is **Content-Type:**, followed by a MIME content type/subtype which describes the format of the segments. Each segment in the body-list must be in the format described by the Content-Type header. This header is required.

Following the headers is a list of segments. Each segment has a single entry in the body-list. An entry begins with the tag **Body:**. This is followed by a **Virtual Sequence**

Number, or **VSN**. The VSN uniquely identifies a segment within a given body-list, and is assumed to have some absolute ordering. That is, a segment with a VSN less than a second segment should be displayed before the segment with the higher VSN and vice-versa. VSNs do not have to be sequential, but must have some absolute relative ordering. Following the VSN is a **URL** which can be used to access the page data. If the URL-stem: header is present and non-empty, the contents of that header will be prepended to the URL, so the URL field does not need to be a valid URL by itself; however, the combination of the URL-stem: field and the URL field must create a valid URL which can be resolved to access the segment data. The URL field is followed by a **Segment Description** field. This field is free text and may contain any of the characters [a-zA-Z_-] and punctuation characters. The segment description field is displayed by the browser to label a button, indicate which version of a page a user is looking at. For example, a segment may be named “Chapter 1” or “Chapters 4-7”. Each of these fields is separated by a tab and the line is terminated by a newline.

The following sample shows a body-list which specifies a segment in Postscript format. The URL field of each segment is appended to the URL-stem: header to form a valid URL to access the specified segment. Finally, each segment is labeled with a segment description. This text will be shown by the browser in a button.

```
Version:          1.0
Digiment-content: body-list
URL-stem: http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Body
Content-type:     application/postscript
Body:    1        ?type=application/postscript        Pages 1-115
```

A.2.7 The preferred-order type

The preferred-order type is used to specify a preferred sequence of data objects for the browser to display. Although page-lists and body-lists can do this, they are limited to only specifying pages or segments which are in a specific format. By using a preferred-order type, a digiment can specify a sequence of objects which are listed in any page-list or body-list in the digiment. For example, a digiment can specify that a preferred viewing order is pages 1 through 5 from page-list A, pages 6 and 7 from page-list B, segment 3 from body-list C, and pages 8 through 45 from page-list A. Digiments may contain multiple preferred-order parts. Thus, another preferred-order may be identical to the previous one described, but use page-list C instead of page-list A.

The preferred-order type starts with the standard application/digiment headers. These are followed by another header and a list of data objects. The header is **Preference Description:**, followed by a short description of the preferred-order. This text may contain any of the characters [a-zA-Z_-] and punctuation characters, and is displayed by the browser to let the user choose which version of the digiment he wishes to view.

Following the headers is a list of data objects. Each object has a single entry in the preferred-order list. An entry begins with the tag **Object:**. This is followed by a **Preferred Sequence Number**, or **PSN**. The PSN specifies the order in which the objects are to be displayed. The first object displayed is the object which corresponds to the lowest PSN, and objects are displayed in order of increasing PSNs. PSNs do not have to be sequential, but must have some absolute relative ordering. Following the VSN is a **Content-ID**, which is the content-ID of either a part-list or body-list digiment part within the digiment. The Content-ID field is followed by a **VSN** field. The VSN specifies a specific page or segment from the part-list or body-list part. By using a VSN and Content-ID pair, it is possible to specify any object which is contained in a digiment. Each of these fields is separated by a tab and the line is terminated by a newline.

The following sample shows a preferred-order part with 5 objects; 3 pages and 2 body segments. The first two objects are the cover and abstract pages. The next two objects are the two Postscript files which make up the body of the document. The final object is the bibliography page. This order is preferred for using color output devices.

```
Version:          1.0
Digiment-content: preferred-order
Preference-Description: Color for high resolution devices
Object:  AIM-1066.600x5cLZW.list    1
Object:  AIM-1066.600x5cLZW.list    2
Object:  AIM-1066.postcolor.body    1
Object:  AIM-1066.postcolor.body    2
Object:  AIM-1066.600x5cLZW.list    115
```

These types are designed to cover the common cases for digiment delivery. However, they are certainly not a complete set of all possibilities for capturing all the various types of data and relationships between types which may be delivered. Therefore, it is expected that this set will be expanded by future types to enhance the flexibility of delivery options and capture more data. A complete listing of the types defined in this document is in Table A.1 on page 91

Digiment-type:	Description
part-list	Listing of all the application/digiments in this digiment
bibliography	Bibliographic material in RFC1357 format
page-map	A mapping from VSN to page #
page-list	A list of pages available in a certain format
preferred-order	A listing of pages or body-list parts in a particular order from various page-list or body-list parts
body-list	A file which contains multiple pages

Table A.1: Application/digiment Digiment-types

Appendix B

A Sample Digiment

B.1 A Digiment With Multiple Formats

```
Content-Type: multipart/digiment; boundry = "========digboundry======"

--========digboundry======
Content-Type: application/digiment
Content-ID: MIT-AILab//AIM-1066.parts

Version: 1.0
Digiment-content: part-list
Part:   bibliography  MIT-AILab//AIM-1066.bib
Part:   page-map      AIM-1066.100x5cLZW.map
Part:   page-list     AIM-1066.100x5cLZW.list   image/gif;dpi=100;bits=5
Part:   page-map      AIM-1066.600x1cG4.map
Part:   page-list     AIM-1066.600x1cG4.list    image/tiff;dpi=600;bits=1
Part:   body-list     AIM-1066.post.body        application/postscript

--========digboundry======
Content-Type: application/digiment
Content-ID: MIT-AILab//AIM-1066.bib

Version:          1.0
Digiment-content: bibliography
 BIB-VERSION:: v2.0
          ID:: MIT-AILab//AIM-1066
       ENTRY:: February 27, 1995
ORGANIZATION:: Massachusetts Institute of Technology, Artificial
     Intelligence Laboratory
       TITLE:: Analysis of Differential and Matching Methods for Optical
     Flow
      AUTHOR:: Little, James J.
      AUTHOR:: Verri, Alessandro
        TYPE:: Technical Memo
        DATE:: August 1988
       PAGES:: 27
       GRANT:: N00014-85-K-0124, DACA76-85-C-0010
       NOTES:: Keywords: optical flow, computational vision, numerical
     differentiation
     Cost: $3.00
     AD-A234424
         END:: MIT-AILab//AIM-1066

--========digboundry======
Content-Type: application/digiment
Content-ID: AIM-1066.100x5cLZW.map

Version: 1.0
```

```
Digiment-content: page-map
Map:     1     supporting        doccontrol
Map:     2     supporting        doccontrol
Map:     3     title page        Titlepage
Map:     4     1                 1
Map:     5     2                 2
Map:     6     3                 3
Map:     7     4                 4
Map:     8     5                 5
Map:     9     6                 6
Map:     10    7                 7
Map:     11    8                 8
Map:     12    9                 9
Map:     13    10                10
Map:     14    11                11
Map:     15    12                12
Map:     16    13                13
Map:     17    14                14
Map:     18    15                15
Map:     19    16                16
Map:     20    17                17
Map:     21    18                18
Map:     22    19                19
Map:     23    20                20
Map:     24    21                21
Map:     25    22                22
Map:     26    23                23
Map:     27    24                24
Map:     28    25                25
Map:     29    26                26
Map:     30    27                27
Map:     31    supporting        agent
Map:     32    supporting        scancontrol
Map:     33    supporting        calibrationIEEE-167A-1987
Map:     34    supporting        calibrationAIIM-#2


--=========digboundry======
Content-Type: application/digiment
Content-ID: AIM-1066.100x5cLZW.list

Version: 1.0
Digiment-content: page-list
URL-stem: http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Page/
Content-type: image/gif;dpi=100;bits=5
Page-map: AIM-1066.100x5cLZW.map
Page:   1       1?type=image/gif;dpi=100;bits=5          Viewable image
Page:   2       2?type=image/gif;dpi=100;bits=5          Viewable image
Page:   3       3?type=image/gif;dpi=100;bits=5          Viewable image
Page:   4       4?type=image/gif;dpi=100;bits=5          Viewable image
Page:   5       5?type=image/gif;dpi=100;bits=5          Viewable image
Page:   6       6?type=image/gif;dpi=100;bits=5          Viewable image
Page:   7       7?type=image/gif;dpi=100;bits=5          Viewable image
Page:   8       8?type=image/gif;dpi=100;bits=5          Viewable image
```

```
Page:   9       9?type=image/gif;dpi=100;bits=5          Viewable image
Page:   10      10?type=image/gif;dpi=100;bits=5         Viewable image
Page:   11      11?type=image/gif;dpi=100;bits=5         Viewable image
Page:   12      12?type=image/gif;dpi=100;bits=5         Viewable image
Page:   13      13?type=image/gif;dpi=100;bits=5         Viewable image
Page:   14      14?type=image/gif;dpi=100;bits=5         Viewable image
Page:   15      15?type=image/gif;dpi=100;bits=5         Viewable image
Page:   16      16?type=image/gif;dpi=100;bits=5         Viewable image
Page:   17      17?type=image/gif;dpi=100;bits=5         Viewable image
Page:   18      18?type=image/gif;dpi=100;bits=5         Viewable image
Page:   19      19?type=image/gif;dpi=100;bits=5         Viewable image
Page:   20      20?type=image/gif;dpi=100;bits=5         Viewable image
Page:   21      21?type=image/gif;dpi=100;bits=5         Viewable image
Page:   22      22?type=image/gif;dpi=100;bits=5         Viewable image
Page:   23      23?type=image/gif;dpi=100;bits=5         Viewable image
Page:   24      24?type=image/gif;dpi=100;bits=5         Viewable image
Page:   25      25?type=image/gif;dpi=100;bits=5         Viewable image
Page:   26      26?type=image/gif;dpi=100;bits=5         Viewable image
Page:   27      27?type=image/gif;dpi=100;bits=5         Viewable image
Page:   28      28?type=image/gif;dpi=100;bits=5         Viewable image
Page:   29      29?type=image/gif;dpi=100;bits=5         Viewable image
Page:   30      30?type=image/gif;dpi=100;bits=5         Viewable image
Page:   31      31?type=image/gif;dpi=100;bits=5         Viewable image
Page:   32      32?type=image/gif;dpi=100;bits=5         Viewable image
Page:   33      33?type=image/gif;dpi=100;bits=5         Viewable image
Page:   34      34?type=image/gif;dpi=100;bits=5         Viewable image

--========digboundry======
Content-Type: application/digiment
Content-ID: AIM-1066.600x1cG4.map

Version: 1.0
Digiment-content: page-map
Map:     1     supporting        doccontrol
Map:     2     supporting        doccontrol
Map:     3     title page        Titlepage
Map:     4     1                 1
Map:     5     2                 2
Map:     6     3                 3
Map:     7     4                 4
Map:     8     5                 5
Map:     9     6                 6
Map:     10    7                 7
Map:     11    8                 8
Map:     12    9                 9
Map:     13    10                10
Map:     14    11                11
Map:     15    12                12
Map:     16    13                13
Map:     17    14                14
Map:     18    15                15
Map:     19    16                16
Map:     20    17                17
Map:     21    18                18
```

```
Map:     22    19                 19
Map:     23    20                 20
Map:     24    21                 21
Map:     25    22                 22
Map:     26    23                 23
Map:     27    24                 24
Map:     28    25                 25
Map:     29    26                 26
Map:     30    27                 27
Map:     31    supporting         agent
Map:     32    supporting         scancontrol
Map:     33    supporting         calibrationIEEE-167A-1987
Map:     34    supporting         calibrationAIIM-#2



--=========digboundry======
Content-Type: application/digiment
Content-ID: AIM-1066.600x1cG4.list

Version: 1.0
Digiment-content: page-list
URL-stem: http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Page/
Content-type: image/tiff;dpi=600;bits=1
Page-map: AIM-1066.600x1cG4.map
Page:   1       1?type=image/tiff;dpi=600;bits=1        Printable image
Page:   2       2?type=image/tiff;dpi=600;bits=1        Printable image
Page:   3       3?type=image/tiff;dpi=600;bits=1        Printable image
Page:   4       4?type=image/tiff;dpi=600;bits=1        Printable image
Page:   5       5?type=image/tiff;dpi=600;bits=1        Printable image
Page:   6       6?type=image/tiff;dpi=600;bits=1        Printable image
Page:   7       7?type=image/tiff;dpi=600;bits=1        Printable image
Page:   8       8?type=image/tiff;dpi=600;bits=1        Printable image
Page:   9       9?type=image/tiff;dpi=600;bits=1        Printable image
Page:   10      10?type=image/tiff;dpi=600;bits=1       Printable image
Page:   11      11?type=image/tiff;dpi=600;bits=1       Printable image
Page:   12      12?type=image/tiff;dpi=600;bits=1       Printable image
Page:   13      13?type=image/tiff;dpi=600;bits=1       Printable image
Page:   14      14?type=image/tiff;dpi=600;bits=1       Printable image
Page:   15      15?type=image/tiff;dpi=600;bits=1       Printable image
Page:   16      16?type=image/tiff;dpi=600;bits=1       Printable image
Page:   17      17?type=image/tiff;dpi=600;bits=1       Printable image
Page:   18      18?type=image/tiff;dpi=600;bits=1       Printable image
Page:   19      19?type=image/tiff;dpi=600;bits=1       Printable image
Page:   20      20?type=image/tiff;dpi=600;bits=1       Printable image
Page:   21      21?type=image/tiff;dpi=600;bits=1       Printable image
Page:   22      22?type=image/tiff;dpi=600;bits=1       Printable image
Page:   23      23?type=image/tiff;dpi=600;bits=1       Printable image
Page:   24      24?type=image/tiff;dpi=600;bits=1       Printable image
Page:   25      25?type=image/tiff;dpi=600;bits=1       Printable image
Page:   26      26?type=image/tiff;dpi=600;bits=1       Printable image
Page:   27      27?type=image/tiff;dpi=600;bits=1       Printable image
Page:   28      28?type=image/tiff;dpi=600;bits=1       Printable image
Page:   29      29?type=image/tiff;dpi=600;bits=1       Printable image
```

Page: 30 30?type=image/tiff;dpi=600;bits=1 Printable image
Page: 31 31?type=image/tiff;dpi=600;bits=1 Printable image
Page: 32 32?type=image/tiff;dpi=600;bits=1 Printable image
Page: 33 33?type=image/tiff;dpi=600;bits=1 Printable image
Page: 34 34?type=image/tiff;dpi=600;bits=1 Printable image

--=========digboundry======
Content-Type: application/digiment
Content-ID: AIM-1066.post.body

Version: 1.0
Digiment-content: body-list
URL-stem: http://cstr-http.lcs.mit.edu/Server/TR/MIT-AILab:AIM-1066/Body
Content-type: application/postscript
Body: 1 ?type=application/postscript Pages 1-27

--=========digboundry======--

Appendix C

The CSTR Scanned Document Record

C.1 The CSTR Scanned Document Record, version CSTR 1.3

January 26, 1995

By Jerry Saltzer, Jack Eisan, and Mike Cook

1. OBJECTIVES

The objective of this document is to specify both the form and content of the information that must be captured when a document is scanned, as an on-line record that becomes a component of the scanned form of the document.

The objective of the scanned document record is to capture information that is not explicit in the scanned image, yet is needed:

- to view, display, or print the image properly.
- to understand how to interpret the image.
- to meet contractual or legal requirements.

2. GENERAL APPROACH

The form of a scanned document record is a series of named fields, one per line. Each field line begins with the field name followed by a colon, then the field value. Any line of the scanned document record may have a comment at the end, preceded by a semicolon. Any program reading the file will ignore all comments. A field line may be of any length, but it may NOT include typed carriage returns. This form is intended to be easily created by a human operator using a word processor or spread sheet, but it is to be used by computer programs that browse, display, or print documents, so the contents of some of the fields must conform to specific standards.

The scanned document record does not specify the format of the image files, but it is intended for use with image files that are in the TIFF format.

The content of a scanned document record is most easily explained by first exhibiting a complete example, and then describing the requirements on field contents:

3. A COMPLETE EXAMPLE

```
Scanning record version: CSTR 1.3
Publishing department: M. I. T. Lab for Computer Science
Report label: MIT-LCS-TM-13
Source: first-generation original
Document series: TM
Image count: 30
Scanning agent: M. I. T. Document Services
Input form: single-sided
Input size: 8.5 x 11
Suggested print form: double-sided
```

```
Suggested print size: 6 x 9
Scanner used: Fujitsu 3096g ADF
Software used: Optix version 1.01
Operator: Michael Cook
Date Scanned: 9/28/1994
Resolution(dpi): 400
Greyscale depth(bits): 8
Scanner settings: default
Text quality: normal
Note:
Comment:filename              size    cksum   contents     comments
Map:  scanrec.cstr.1.3.txt   45102   67890   format
Map:  MIT-LCS-TR-13-srec.txt 113076  12345   scanrecord
Map:  MIT-LCS-TR-13-001.tif  8417048 12345   cover
Map:  MIT-LCS-TR-13-002.tif  8417048 01234   blank
Map:  MIT-LCS-TR-13-003.tif  8417048 78912   unnumbered; title page
Map:  MIT-LCS-TR-13-004.tif  8417048 56789   blank
Map:  MIT-LCS-TR-13-005.tif  8417048 23456   unnumbered; acknowledgment
Map:  MIT-LCS-TR-13-006.tif  8417048 90123   blank
Map:  MIT-LCS-TR-13-007.tif  8417048 67890   numbered 1
Map:  MIT-LCS-TR-13-008.tif  8417048 34567   numbered 2
Map:  MIT-LCS-TR-13-009.tif  8417048 01234   numbered 3
Map:  MIT-LCS-TR-13-010.tif  8417048 78901   numbered 4
Map:  MIT-LCS-TR-13-011.tif  8417048 45678   numbered 5
Map:  MIT-LCS-TR-13-012.tif  8417048 12345   numbered 6
Map:  MIT-LCS-TR-13-013.tif  8417048 89012   numbered 7
Map:  MIT-LCS-TR-13-014.tif  8417048 56789   numbered 8
Map:  MIT-LCS-TR-13-015.tif  8417048 23456   numbered 9
Map:  MIT-LCS-TR-13-016.tif  8417048 90123   numbered 10; original skewed
Map:  MIT-LCS-TR-13-017.tif  8417048 67890   numbered 11
Map:  MIT-LCS-TR-13-018.tif  8417048 34567   numbered 12
Map:  MIT-LCS-TR-13-019.tif  8417048 01234   numbered 13
Map:  MIT-LCS-TR-13-020.tif  8417048 78901   numbered 14; original says
page 16
Map:  MIT-LCS-TR-13-021.tif  8417048 45678   numbered 15
Map:  MIT-LCS-TR-13-022.tif  8417048 12345   numbered 16
Map:  MIT-LCS-TR-13-023.tif  8417048 89012   numbered 17
Map:  MIT-LCS-TR-13-024.tif  8417048 56789   spine
Map:  MIT-LCS-TR-13-025.tif  8417048 23456   supporting; instructions to
printer
Map:  MIT-LCS-TR-13-026.tif  8417048 90123   doccontrol
Map:  MIT-LCS-TR-13-027.tif  8417048 67890   calibration IEEE-167a-1987
Map:  MIT-LCS-TR-13-028.tif  8417048 34567   calibration AIIM-#2
Map:  MIT-LCS-TR-13-029.tif  8417048 01234   agent
Map:  MIT-LCS-TR-13-030.tif  8417048 78789   scancontrol
```

4. REQUIREMENTS ON THE FIELDS

Here are the requirements on the field contents that come about in order for a computer program to be able to unambiguously interpret the scanning record:

Scanning record version: must contain exactly the string shown in the example. The version number is incremented only for changes in the specification that are incompatible with the practice of the prior version. If a change in the specification does not require a change in the practice, the version number is unchanged.

Publishing department, Technical report label, Document series, Scanner used, Software used, and Operator: each can contain any string of characters. Conventionally, the report label will begin with the string “MIT-”

Source: One of the strings “First-generation original”, “Later-generation copy”, or “PostScript”

Image count: Must contain an integer. (The image count is the number of image files created for this document, including calibration, identification, and blank targets, etc. It matches the number in the image-number component of the highest-numbered Map record.)

Input form and Suggested print form: Must contain either “single-sided” or “double-sided”.

Input size and Suggested print size: Must contain two decimal numbers separated by the letter “x”. The integers are assumed to be measurements in inches.

Date scanned: Month/Day/Year, each component being an integer (leading zeros omitted) and the year being four digits.

Resolution and Greyscale depth: must be integers

Scanner settings: At this time the only allowed value for this field is “default”.

Note: an optional field containing any desired string. If this field is not empty, a properly constructed browser or print program will display its contents to the reader.

Comment: an optional field containing any desired string. It is ignored by browsers.

Map: Each file that comprises a single scanned document is represented by its own Map: field. This is the only field that may appear more than once in the scanning record. All Map fields must be together at the end of the scanning record. A Map field consists of an image file name, the file size, a five-digit checksum computed using the method of the BSD UNIX “sum” command, and an image content identifier. The checksum value that appears in the map for the scanning record file itself must be zero, and for the moment its value is ignored. The content identifier must be one of the following:

```
scanrecord      Document scanning record
format          Scan Record specification (this specification)
scancontrol     Scanning project control form
doccontrol      Government control form
calibration     Test target
agent           Logo of the scanning agent
blank           Blank page inserted to maintain duplex printing
                order
spine           Image intended for the book spine
cover           Image intended for the book cover
unnumbered      Page for which it appears that the publisher did
                not intend a page number
```

numbered	Page of text that the publisher intended to number, whether or not that number is printed on the page
supporting	Material not intended for display or printing

The content identifier “calibration” is followed by a string that identifies a calibration test target. The content identifier “numbered” is followed by a string that represents the publisher’s originally assigned page number.

The order of the Map fields is taken to be the order in which the images they describe are intended to be printed or displayed, except when the input form is double-sided. In that case all pages within the numbered region are assumed to have been scanned odd sides first, even sides last and reversed.

A browser should treat any field that it does not recognize as a comment, and ignore it unless the user has asked to be told about unrecognized fields.

5. DOCUMENT NAMING CONVENTIONS

The Map field of the document scanning record is flexible enough to allow any desired file name to be used for any component of the document. However, to minimize confusion, the following conventions are used in naming the files.

Image file names consist of five components separated by hyphens and followed by a suffix:

- The first component of all file names of documents originating at MIT is the string “MIT”.
- The second component of the file name is a string that identifies the department or laboratory that originated the document, e.g. “LCS” or “AI”.
- The third component of the file name is a string that identifies the document series, e.g. “TM”, “TR”, “AIM”, etc.
- The fourth component of the file name is the series tag of the document. This tag is usually an integer such as “417”, but it may include letters, as in “418a”, and in some cases it may be an arbitrary string of letters.
- The fifth component of an image file name is the image number, with enough leading zeros that all image file names for the document are of the same length. Image numbers start with 1 and are consecutive in the order the images were scanned.
- The file name of an image file ends with the string “.tif”.
- The file name of the scanning record ends with the string “srec.txt”.
- The complete file name of the format specification of the scanning record is simply “format.txt”.

The various images (in TIFF format), the scanning record (in text format), and the format specification of the scanning record (also in text format) are collected in a directory or folder which is named by the first four components above, separated by hyphens.

Note that upper- and lower-case letters in the file names of “Map” fields must be recorded exactly as they appear in the actual file name, because on some computer systems upper and lower-case letters are distinct.

6. OPEN QUESTIONS AND THINGS FOR FURTHER THOUGHT OR FUTURE REVISIONS

1. The “Document series” field is probably not needed, because the information is contained in the document label field.

2. The original document date isn’t captured. Should it be?

3. Need a systematic way to insert scanning agent logo on cover page.

4. The content identifiers “scancontrol” and “spine” could be handled as “supporting”, with their intended use appearing as comments.

5. There is no support for bringing the reader’s attention to the copyright notice. One suspects that some scheme will be needed for a browser to locate that notice and display it. (Since TR’s and TM’s generally don’t have copyright notices, this issue isn’t particularly pressing.)

6. It may be appropriate to more systematically record observations by the scanning operator, such as the original is skewed, off-center, or wrinkled or contains a photograph or other non-textual material.

7. The checksum of the scanning record that is recorded in the scanning record has the value of zero. The specification calls for it to be ignored because simply inserting the current checksum in the record would change its checksum. A procedure for adding a fiddle field at the end of the record whose value is chosen to force the actual checksum of the file to zero or some predictable value should be developed. Alternatively, the checksum for this file should be defined as being the checksum that is obtained when the checksum field for this file is replaced with zero.

8. It might be a good idea to add an “end:” field to make it easier to figure out that the scan record is complete.

9. The information captured about page numbers isn’t really sufficient to establish which image is “next” in cases where one page is scanned twice (once in black and white and a second time in color) or is scanned in two parts, as in an oversize foldout.

10. There isn’t enough information to figure out how to paste back together two image files to produce a single image (e.g., a separately- scanned color photo that belongs in the middle of a page of monochrome text.)

REVISION HISTORY

ACKNOWLEDGEMENT

This note is expanded from a set of ideas originally developed at a Library 2000 group meeting on March 17, 1994. Discussants: Jack Eisan, Mitchell Charity, Ali Alavi, Sally Richter, Mary Anne Ladd, Jeremy Hylton, Geoff Seyon, Eytan Adar, Greg Anderson, Jerry Saltzer. Since that time additional suggestions and ideas have come from Michael Cook, Gillian Elcock, Yoav Yerushalmi, and Andrew Kass.

For more information contact Jerry Saltzer <Saltzer@mit.edu>

References

[1] N. Borenstein and N. Freed. *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. RFC5121, Bellcore, September 1993.

[2] Andrew Braid. From Babel to EDIL: the evolution of a standard for document delivery. Computer Networks and ISDN Systems, Vol. 27 (1994), No. 3, December 1994, pages 367-374.

[3] Kraig Brockschmidt. Introducing OLE 2.0, Part I: Windows Objects and the Component Object Model. *Microsoft Systems Journal*, August 1993.

[4] D. Cohen, ed. *A Format for E-mailing Bibliographic Records*. RFC1357, ISI, July 1992.

[5] David Crocker. *Standard for the Format of ARPA Internet Text Messages*. RFC822, University of Delaware, August 1982.

[6] Melia H. Hoffman, Lawrence O'Gorman, Guy A. Story, James Q. Arnold, and Nina H. Macdonald. The RightPages Service: An Image-Based Electronic Library. *Journal of the American Society for Information Science*, Vol. 44, No. 8, September 1993, pages 446-452.

[7] Paul McJones and Andrew Birrel, Electronic-mail correspondence, December 1994-April 1995.

[8] Paul Mostert. *TULIP Specification of the Data Structure and of the Delivery Methods, Version 2.2*. Elsevier Science B.V., Amsterdam, March 1994.

[9] Kurt Piersol. A Close-Up of OpenDoc. *Byte*, March 1994.

[10] Steve Putz. *Design and Implementation of the System 33 Document Service*. Information Sciences and Technologies Laboratory, Xerox Palo Alto Research Center, 1993.

[11] Jerome H. Saltzer. Technology, Networks, and the Library of the Year 2000. *Proceedings of the International Conference on the Occasion of the 25th Anniversary of Institut National de Recherche en Informatique et Automatique*, Paris, France, December 1992, pages 51-67.

[12] Guy A. Story, Lawrence O'Gorman, David Fox, Louise L. Schaper, and H.V. Jagadish. The RightPages Image-Based Electronic Library for Alerting and Browsing. *Computer*, Vol. 25, No. 9, September 1992, pages 17-26.