Tape Archiving Using the Time Capsule File System

by

Brian K. Zuzga

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995

Author . . . .
Department of Electrical Engineering and Computer Science
June 30, 1995

Certified by . . . .
Harold Abelson
Class Of 1922 Professor and Macvicar Teaching Fellow
Thesis Supervisor

Accepted by . . . .
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Tape Archiving Using the Time Capsule File System

by

Brian K. Zuzga

Submitted to the Department of Electrical Engineering and Computer Science
on June 30, 1995, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

MIT has been generating data on digital media in significant quantities since the 1960's. The rate at which new bits are produced is steadily increasing, and the knowledge needed to decipher the old bits is vanishing. All of our older media are beginning to deteriorate, and valuable data is being lost every day. The Time Capsule File System (TCFS) project was developed to address the problem of preserving this information, for use now and in the future.

Specifically, TCFS is intended to be a single universal format to which we can migrate all our old files, thus simplifying the problem of dealing with a myriad specialized formats. TCFS is also designed to be simple enough to re-engineer without any previous understanding of it, which reduces the risk that data will become "stranded".

Using TCFS, Dr. Alan Bawden and I developed a framework using TCFS to preserve our archives in a durable and easy to use fashion. Pandora Berman and I showed this framework to be viable by using it to rescue data from backup tapes written on the Incompatible Timeshare System (ITS). To this end, I developed specialized tools to translate the data from these tapes into the more general TCFS format. Simultaneously, I developed other tools to categorize, index, and search the large set of TCFS archives that were created.

Thesis Supervisor: Harold Abelson
Title: Class Of 1922 Professor and Macvicar Teaching Fellow

Acknowledgments

I would like to acknowledge the advice, help, and encouragement I received from Michael Blair, Dr. Olin Shivers, Dr. Kleanthes Koniaris, Daniel Coore, Ed Moriarty, Myron Freeman, Bob Givan, Sara Culmone, Susan Lucas, and Prof. Hal Abelson.

A great deal of credit is due to the other members of the TCFS team: Dr. Alan Bawden and Pandora Berman. Without their efforts this project would still be a few scribbles on the back of a paper napkin. They helped with everything from the design and implementation of this system, to historical information, to making sure this document looked like it was written in English. I am just a newcomer to this team; they have been working to preserve our old data since the time much of it was originally written. Charles Hannum and Nicholas Papadakis also assisted us in implementing some critical tools and the TCFS team sincerely appreciates their efforts.

Special thanks go to Mary Chung's *suan la chow show* for keeping the project members happy and well fed during important "design meetings." Lastly, I'd better not forget the numbers "0" and "1", they helped quite a bit.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"History teaches everything, even the future."
–Lamartine, speech at Macon 1847.

The Time Capsule File System (TCFS) project was started by Dr. Alan Bawden to address the problem of preserving archival data for MIT's Artificial Intelligence Lab and Laboratory for Computer Science. The project's members currently consist of Dr. Bawden, Pandora Berman, and Brian Zuzga. To date, Dr. Bawden and I have established a universal framework for preserving any archival files in a platform- and medium-independent fashion and Ms. Berman and I have begun the migration of approximately 77 gigabytes of the Labs' Incompatible Timeshare System (ITS) backup tapes to the TCFS format, as proof of the concept. I have prototyped tools to help read and manipulate this body of TCFS files so users will be able manipulate and use this vast amount of data.

The motivations for this project are the extremely poor condition of the old backup tapes and the vanishing base of people who understand how to decode them. We are losing data every day, and some of this data may prove valuable for historical, legal, and personal reasons. Also, a profusion of formats exist at the Labs, making it difficult to track and maintain them all. A single universal format will reduce the chances that valuable data would become stranded in some esoteric format. We know that some of these tapes are extraordinarily valuable right now, but we do not know which ones. There are also tapes that are not valuable now, but may be in the future.

This document contains three basic sections. The first section, consisting of chapters two and three, deals with the overall TCFS framework that has been established for archiving any file worthy of preservation, regardless of its format. The second section, chapter four, addresses issues encountered specifically while processing the ITS backup tapes. The third section, chapters five and six, discusses tools to help users deal with the massive quantity of TCFS files that have been and will be created. These tools are not application-specific, but neither are they a core component of the TCFS system, since they are not held to the same standard of robustness over time.

Chapter 2

Overview—The Problem and My Solution

"History will be kind to me, for I intend to write it."
–Winston Churchill

2.1 The Problem We Are Trying to Solve

At MIT's Artificial Intelligence (AI) Lab and Laboratory for Computer Science (LCS), we have rooms full of 7-track and 9-track magnetic tapes in various states of decay. These tapes are the incremental, full, and archival dumps of all the machines used for everyday work by students, faculty, and staff in the Labs[1] from 1971 to 1990. The data on any individual tape may or may not be valuable; we are unable to tell. Much of this data was written by operating systems no longer in common use, such as ITS, TOPS-20, and Genera. An air conditioner malfunction has allowed water to leak onto some of the tapes. Some of the protective outer tape rings have grown old and brittle. Many have already failed, causing the tape to fall off the tape racks and their cases to shatter. Many tapes are being stored off-site in an environmentally controlled warehouse, but some are completely un-cataloged. Others may be sitting lost in the lab's basement somewhere. Furthermore, we are losing the knowledge required to make sense of the data on the tapes, which could be in one of a dozen formats. Even perfectly preserved tapes may thus soon be reduced to gibberish. We need to pay attention to more than just the tapes themselves; we need to rescue data from the operators who wrote these tapes and from the paper dump logs. In summary, we are losing irreplaceable data every day.

The long-term goal of this project is to provide a usable system whereby users can easily search and access all the Labs' archival data in a uniform fashion, and to preserve this data in such a manner that it can be decoded even if all the supporting documentation for it is lost. The more immediate goal is to assemble the incremental, full, and archival backups from the Incompatible Timeshare System (ITS), record them on new media in a long-term (archival) format suitable for use well into

[1] It is important to note that the AI Lab and LCS still occupy the same physical building in 1995, so a reference to "the Lab" means one of the two organizations, while "the lab" means the physical building.

Operating System	Machine	Density (bpi)	Format	Approximate Year Written	Number	Size
ITS	DM KA-10	800	9 track	1976–83	330	–
ITS	AI KA-10	800	7 track	1971–82	362	–
ITS	ML KA-10	800	7 track	1973–83	203	–
ITS	MC KL-10	800	7 track	1976–85	511	–
ITS	MC KL-10	1600	9 track	1981–86	60	–
ITS	All KS	6250	9 track	1985–90	159	–
ITS Total					1625	77GB
Tops-20	AI OZ	1600	9 track	1982–86	397	–
Tops-20	AI OZ	6250	9 track	1986–88	118	–
Tops-20	LCS XX	mixed	9 track	1976–88	423 (in storage)	–
Tops-20 Total					938	132GB
Mixed/Unspecified		1600	9 track	–	757	30GB
Mixed/Unspecified		6250	9 track	–	436	63GB
Grand Total					**3,756**	**302GB**

Table 2.1: Inventory of Known Tapes with Valuable Archival Data

the future, and provide tools to help search and manage this large data set in an effective manner. The entire collection of readable ITS backup tapes contains approximately 77 gigabytes of data, and constitutes our smallest collection of tapes. Table 2.1 provides a brief breakdown of the tapes that have been accounted for and are believed to contain useful data. There are other tapes around the lab which are unlabeled, and there are many tapes known to be uninteresting "scratch" tapes.[2]

Starting in 1968, ITS was the workhorse of Project MAC, which later divided into the Laboratory for Computer Science (LCS) and the Artificial Intelligence (AI) Lab. ITS ran on a two hardware platforms, the DEC PDP-6 and later the PDP-10; and was implemented entirely in MIDAS, an assembly language with macro facilities[4]. One of the primary applications for ITS was MACSYMA, a symbolic and algebraic manipulation system developed under the direction of Prof. Joel Moses starting in 1969[2, 5]. The ITS machines were used for everything from writing computer programs to reading electronic mail to formatting dissertations and technical reports. Dr. Bawden has said that three common programs to be seen running on these systems were EMACS, MACSYMA, and the MACLISP dialect of LISP.

There has been a demonstrated need for some of the data on these tapes. We are not just trying to rescue data that will never be used. Here are some examples of requests that we have encountered:

2.1.1 "I Want My Thesis!"

Alumni have returned to the Labs and asked if they could retrieve some of their old work. They generally want the original files containing their theses and any accompanying code they may have written. Currently, it is very difficult to serve such requests. First, one must search the backup logs and determine which files need to be recovered, then the physical tape (which could be off-site) needs to be located. After that ordeal, one must find a tape drive that can read these old tapes and is attached to a system with the ability to make sense of the raw bits on them. Finally, it is a difficult

[2] We may eventually make an effort to preserve these tapes, but they have a relatively low priority.

task to find or develop the software to interpret these old formats. Most of what does exist is shoddy and fragile, so this vast number of ITS tapes are very difficult for us to use currently.

2.1.2 Historical Value

Dr. Philip Agre approached us recently and asked if it were possible to look through our old backup tapes and determine when the word "foo" started being used by our community. Currently this project would require that someone search through all the old backup tapes to find a definite answer to this question. Someday, someone may ask "When did Marvin Minsky first conceive of k-lines?" We want to ensure that future historians will be able to find a reasonable answer to that question, and maybe even make it easier for them to get the answer.

2.1.3 Intellectual Property and Patent Searches

People at MIT have worked on many research projects which have eventually proven commercially useful. Every day, companies try to patent ideas they believe to be original, but sometimes people have already researched the ideas. Such research is prior art in the field, and render the ideas ineligible for patent protection. As a hypothetical example, there are many modern mathematical packages that include algebraic manipulation systems, so it would be useful to show prior art in the MACSYMA source code. Unfortunately, there are currently no convenient ways of looking for prior art in our backup tapes.

2.2 Alternatives Which We Considered Impractical

Before we embarked on a massive project to copy old tapes onto new tapes and invested effort into developing software to translate the data into a more meaningful format, we examined some of the alternative solutions. The main requirement for any solution was that it had to work for the tapes containing data from the Incompatible Timeshare System (ITS), which is the oldest and most eccentric operating system that was in production use at the Labs. Any design that can accommodate ITS can easily be adapted for any Unix, DOS, Genera, or TOPS-20 data. These are the options that were considered before we decided to develop our own:

2.2.1 Preserving Only the Raw Byte Stream

This is the most inexpensive solution by far and minimizes development costs. One just has to copy the bits and record boundaries as they exist on one tape onto new storage media. This process may include checking the contents of the tape against the paper dump logs, but nothing more would be done. If we were particularly clever, we could copy the old tapes onto more compact media in a simple manner, saving us money on media costs. One obvious problem with this minimalist approach is that the knowledge to interpret the data is being lost much more quickly than the data on the tapes themselves — so the real danger with this approach is that in 10 years, we might have hundreds of gigabytes of tapes that we cannot make sense of.

2.2.2 Maintaining Vintage Systems

One suggestion proposed by ITS aficionados was to use the remaining parts of all the decommissioned ITS systems to maintain one working ITS installation; then anyone who needed to rescue any old data could use this setup to do it. Unfortunately, the expertise required to repair the unreliable hardware is vanishing even more rapidly than knowledge of the software. Also, this scheme would still require us to copy any old tapes that we wish to preserve, as discussed above.

2.2.3 Emulators

Recently, Kenneth Harrenstien developed a portable emulator of the PDP-10 which can run the ITS operating system. Use of this emulator would avoid the problem of maintaining the old hardware, but would still require that one know how to use the ITS operating system; we would have escaped the constraints of old hardware, only to be trapped by old software. Rather than spending the time necessary to maintain an emulator, it would be better for us to spend it translating the data to be saved into a form that can better weather the passage of time. Thus we could eliminate the need for any knowledge of the original operating system, instead of continuing to exert ourselves to port the emulator to every new "industry standard" platform. Also, if we used such emulators, we would have to maintain one for each operating system we decided was valuable enough to preserve. Emulators increase in complexity as the original hardware increases in complexity, so this problem only would get more difficult with time. In conclusion, there would be a serious continuing cost that would have to be addressed if this approach were used.

2.2.4 Maintaining Full Copies of the Operating System and Relevant Applications

We have full copies of the ITS operating system source code (with documentation and any relevant applications) already copied onto our current systems. This resource has not helped us, and we do not see it could be any more useful to anyone in the future. The only use such information could provide would be to allow someone to engineer a system that translates the data –- such as the one we eventually implemented. We believe it is important to do this now, before we lose any further knowledge about the original implementation. In addition, the cost for a human to interpret the code and documentation and develop such a system will only increase with time.

This, in essence, is the approach we have been using for some time. However, we do not want to continue using it as our sole means of preservation because it has already proven to be inadequate. Besides, this method does not solve the problem of reducing the number of formats that we need to decode in order to read all of our data.

2.2.5 Translating the Data into an Industry Standard Format

One could imagine translating all of our files into documents accessible to modern word processors, spreadsheets, etc. There is a continuous upgrade path of translators going from these formats to the next format that comes along. The advantage of this approach is that our entire data set would be usable by all the common applications of any one time. Unfortunately, this approach also requires

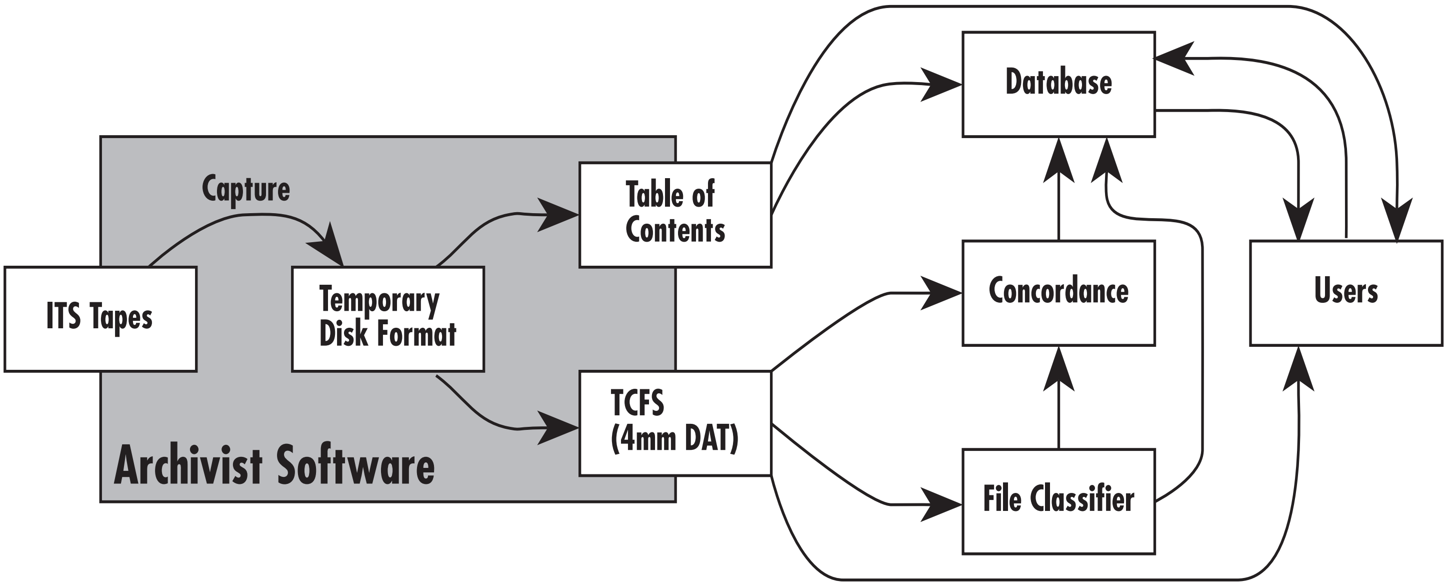

Figure 2-1: Overall View of TCFS System

that we translate our documents each time a new format comes along. Many of these translators seem to lose a small fraction of the semantics of the original document; for example, the rows in an ASCII text table may no longer line up. In passing through translator after translator, more and more formatting information may become lost or garbled, until the document becomes unreadable. Therefore, if we are going to translate our documents, it is important to maintain the original document and to choose a format that will remain stable for a long time.

2.3 Overview of What We Implemented

Fig. 2-1 provides a visual overview of the system as it is currently implemented. Some of the pieces can be re-used for other operating systems in their current form while others are ITS-specific.

The initial step is to read the raw data from the ITS source tapes, a process that we are calling "capture." The primary purpose of this process is to end our dependence on the fragile, aged, physical media involved. The resulting data is recorded in a system-specific format on the hard disks of our capturing system. The remainder of the archivist software then takes these temporary files and translates them into the more durable TCFS format written on 4mm DAT cassettes.[3] TCFS is designed specifically to weather the test of time better than the native file format. The archivist software also records some relevant header information, such as file name and file size, in the table of contents to assist future users with finding files in our TCFS archives. The translation process that the archivist program uses is specific to ITS, while the TCFS format, the table of contents, and the tools to manipulate both are more general.

[3]DAT was chosen quite arbitrarily; it would be trivial for us to switch to any other medium at any time. TCFS is completely insensitive to the media and system it is written on. The cost and availability of DAT were the prime factors in our decision, not any technical merits of the medium.

Another suite of system-independent tools has been developed to manipulate the two products of the archivist software. On the simplest level, there is a tool to read back files from TCFS archives into our current Unix file systems and a similar tool for reading and searching the table of contents. More advanced tools for indexing and classifying have been prototyped. One such tool is the file classifier, which takes as input an arbitrary TCFS file and produces its best guess for the file's type. This information is then used by another tool known as the "concordance," which attempts to extract relevant keywords from the body of a TCFS file and store them in an index. Finally, all of the above information useful for searching can be stored in a database, allowing users to make requests that rely on information from each of the components. The database can also have provisions for users' annotations to the material, since their interpretation of the content is valuable as well.

The work done with ITS demonstrates the feasibility of a more general framework for migrating files from all of our obsolete systems into a more durable form. By using TCFS, we can reduce the myriad of formats that are currently being used for archival storage to a single format that is much easier to reverse-engineer from scratch. We hope that current users will benefit from the ability to retrieve their archival data stranded on old tapes; we also hope that future generations can benefit from the historical knowledge that this project will make available.

Chapter 3

TCFS—The Digital Rosetta Stone

"Cleaning up the past will always clear up the future!
Lucky Numbers: 6, 7, 31, 36, 49, 53" –A Fortune Cookie

The disadvantages of keeping old file data in its raw form, as discussed in the previous chapter, forced us to choose a more amenable format for this mass of data. The ideal archival format would be portable across media and platforms and would be easy to reverse-engineer. We first searched for any existing solution we could adapt to this problem.

The need for a data format that is platform- and medium-independent has been recognized in industry for some time, as demonstrated by the System Independent Data Format[14] developed by the SIDF Association and Legato's NetWorker[9]. To be fair to these vendors, their goal was not a long-term archival format. (One would be surprised at how easily supposedly short-term storage becomes long-term.) Recently, however, people have begun to focus on the issues of media obsolescence and technology preservation[13]. We must evaluate our choice of format in the context of archival storage.

A file stored in one of the above mentioned formats is utterly indecipherable to a user not versed in the format specification. It is not valid to assume that future digital historians will have a copy of these specifications, or even be able to understand them if they do. Upon study, it becomes clear that the design of a data format for long-term archival purposes is very different from the format design for most other common applications. Consequently, we decided that the existing formats that we had examined were inadequate for our purposes, and started to develop our own format.

3.1 Design Goals

The goals for the Time Capsule File System (TCFS) format, as specified in appendix A, include the necessary provisions for platform and medium independence. It can even be used to encode non-traditional files with multiple data segments –- for instance, files from the Apple Macintosh Hierarchical File System. However, TCFS does not stop there, because we know neither what computers nor what data storage will look like in the future. With TCFS, future users will easily be able to reconstruct the format from the actual archives if all the TCFS specification documents are lost.

We made as few assumptions as possible about the archive file's reader, but we were forced to make some assumptions in the format since it would have been extraordinarily difficult to assume zero knowledge. TCFS first assumes that the reader of the file has an understanding of English (or can still find someone who does understand English), since most of the content of the files is worthless if you were to decode it and not understand the language in which it is expressed. Future archivists must also have a knowledge of the ASCII character set to decode the file headers, but the body of the file has no such restrictions. (A file encoded in the EBCDIC character set would have the headers encoded in ASCII, while the data could still be in some variety of EBCDIC.) Finally, one must understand the concept of a file as a metaphor for organizing information. This is a particularly important assumption. Without this idea, the archived information is mostly gibberish, and at the rapid rate computers and software are evolving, the idea of a file may be abandoned quickly for something more versatile, like some variety of persistent objects.

3.2 The Solution

The solution to these criteria is simple and elegant. The format, as demonstrated in Fig. 3-1, consists mostly of human-readable field identifiers, each followed by a colon and then a human-readable representation for that field value. Dates are stored in a format very similar to the RFC-822 format[1] used for electronic mail. User names are stored as strings, not as user ID numbers. In this fashion, there are no complex encodings to remember or get lost over the years. There is a more thorough description of the format in Appendix A, which would be very helpful to include in all such archives, but it is not essential for decoding the format.

The format of the data segment of the file is somewhat more problematic and dependent on the "System" field. For instance, if one were to store a word processor document written in a proprietary format in a TCFS archive, the headers may be understandable, but the data would be indecipherable. Luckily, in the ITS file systems that were used as examples for this project, most of the data involved consists of ASCII text. However, in a more modern file system, some sort of conversion will have to take place to leave the data in an understandable format. There is a provision in the TCFS format for including both a translated and a raw version of a document in the same file entry.[1] For instance, if one were to archive a document written in Microsoft Word (a popular word processor in 1995), it would be ideal to include translated Rich Text Format (RTF) and plain ASCII text along with the raw data. One could also take this approach to its logical conclusion and disassemble all executable binary files into their mnemonic machine instructions, but this exercise would be of limited value if the source code were also available. The most logical format for the ITS data field preserves both the original 36-bit words and the readability of ASCII text files.

It may often be impractical to translate files into a more reasonable format because of proprietary data formats or lack of resources. However, the exercise of archiving the untranslated file in TCFS format may still prove useful, since the header information included may indicate to future digital archaeologists whether or not a particular document is worthy of more attention.

Many people don't consider the possibility that an application program will someday be completely

[1]This is the mechanism used for including the raw and translated directory listings in the ITS example.

```
[  -----------------  Begin Time Capsule File  -----------------  ]
TCF-Length:36570
TCF-Date:12 Apr 1995 19:19:12 +0000
TCF-Host:CORN-POPS.AI.MIT.EDU
TCF-User:boogles
TCF-Type:Archive
Archive-Title:AI-KS INCR202
Capture-Host:AI.AI.MIT.EDU
Capture-User:(unknown)
Capture-Date:29 Mar 1990
Pack:0
System:ITS
Tape-Info:AI 202, Type Incremental, Tape 202, Reel 0, File 54
Name:ALAN;ALAN MAIL
Written:29 Mar 1990 15:52:45 -0500
Accessed:29 Mar 1990
Author:(485) .MAIL.
Byte-Size:7
Length:36820
Data;36036:Received: from lcs.mit.edu (CHAOS 15044) by AI.AI.MIT.EDU
 .
 .
 .
TCF-Checksum:DABGBAI@
```

Figure 3-1: Example of a TCFS file

obsolete and impossible to run. However, recent history has shown time and time again that even industry standards change quite rapidly. Efforts to converge on word processor and spreadsheet formats are foiled by continuing evolution in those applications, as new features constantly break the old molds. Even if we were to converge to a standard format for the main types of documents, it is doubtful that these formats would weather the decades as ASCII text might.

A final formatting problem that concerned us from the inception of this project is that we are never sure if the data read from the old tapes is correct and not corrupted. Most of the backup formats involved have no method of error detection built into them. For this reason, we included a 32-bit CRC checksum in the TCFS format, so users will be able to verify whether the new files we write have been corrupted. Of course, this error-checking will only catch errors that occur after we have converted the data into TCFS format. Future archivists are not required to verify this CRC: it is just an added benefit. We chose a simple CRC over a more complex error correcting scheme for its readability and the fact it can be ignored if one can't decode it. A remaining major problem is that we simply trust that our source data is not corrupted.

Chapter 4

The Archivist—Translating Greek into English

The oldest data set, the ITS backup tapes, was chosen for the first migration to TCFS format; it includes tapes written between 1971 and 1990. This chapter discusses many of the specifics of translating and archiving the ITS operating system. Most of the ITS tapes were written on DEC PDP-10's, which used 36-bit words rather than 8-bit bytes. This data set therefore provides a prime example of how platform-independent TCFS is.

4.1 Capture

The initial step was to transfer the raw bits from the 1600bpi 9-track magnetic tapes on which the data originally resided. There is a group of tapes in a 7-track format that we neglected, since the hardware and software interface required to perform this task was beyond the scope of this example.[1]

4.1.1 Design

The capture software is simple and robust. No complex translation or data manipulation operations are attempted at this stage, since they could only introduce more modes of failure. We assumed that some of the tapes would only be readable once, so any failure on the part of our software would be unacceptable. For this reason, we made a conscious design decision to postpone any translation until all the data from a given tape was safely on disk.

One of the most frustrating obstacles we encountered while writing this software involved the operating system's tape driver. Upon reaching a media error, the two commercial tape drivers we tested automatically retry the read operation several times before the software gives up and moves on. This process can include as many as 30 rewind and re-read cycles, which may lead to undue

[1] One day, Dr. Thomas Knight may take on the task of assembling the necessary hardware to read these 7-track tapes.

wear on the tape or even catastrophic media failure before we can read the entire tape, resulting in data loss. Unfortunately, we eventually decided that we would live with this possibility and continue to use the tape drivers shipped with our operating system, as a lack of resources prevented us from developing our own driver software.

4.1.2 Degradation of Media

The capture process proved to be a challenging endeavor since some of the tapes involved were over 20 years old. The degradation of the media varied widely depending on the previous usage of the tape. We had initially expected that many of the tapes would have degraded to the point that they would only make it through the tape mechanism once before they either broke or had so much metal oxide flake off that they became unreadable. We also expected to find a great number of hard media errors, which the software would have to handle gracefully and efficiently, or we would never make any progress.

Much to our surprise, we have encountered far fewer of the failure modes than we had expected. Instead, our most common failure was much more counter-intuitive. We expected that the incremental dump tapes would be in the worst condition, since they had been overwritten repeatedly. In contrast to the incremental tapes, the full dumps were performed on brand new media and not accessed as frequently. It *was* the case that the incremental tapes had the highest incidence of read errors, but much to our surprise, the quality of the tapes *improved* after we ran through the entire read process a few times. We speculate that we were removing loose oxide from the back side of the tape, which had adhered to it from the next turn on the tape spool. Initially, this loose oxide interferes with reading, but after we removed it, the tape became more readable.

4.1.3 Capturing Data from the Operator

The capture program does more than simply capture the data from the old magnetic media. We also want to capture any information that the operator can provide regarding the old tapes and any information from the paper dump logs. The capture program has a mechanism for collecting some of that sort of data, such as the operator who performed the original dump and the dump title. We are fortunate to have Ms. Berman, who wrote many of these tapes, doing the labor to convert the tapes to TCFS format. We don't want to miss any opportunity to capture additional information by paying attention only to the media without regard for the human element. We may eventually have a better idea of what additional information to capture from the operator as she continues to use the system.

After the raw bits are read from the backup tapes, they are written in a temporary format onto a hard disk, which acts as a buffer for the next stage.

4.2 Translating

The second stage of the migration prepares the raw data for output in the TCFS format. This step includes interpreting the directory listings and decoding the user ID numbers to user name mappings. More importantly, we had to find some way to take the 36-bit words from the ITS file system and translate them into something coherent that can be stored in 8-bit bytes on the systems we are currently using for data storage. ITS predates the presently ubiquitous 8-bit byte, so we faced some unusual challenges maintaining the semantics of the original data.

4.2.1 Word File Format – ASCII is ASCII is ASCII

One of the primary goals of this project was to design a format for storing archival data that assumed as little knowledge as possible on the part of a future archaeologist reading the data. So, when encoding ASCII text files stored in 36-bit words, it is desirable for the ASCII text to still be readable as ASCII text in 8-bit bytes. This would make no further assumptions about what the reader knew.

Fortunately, electronic mail, program source code, and much of the system documentation on ITS was stored as ASCII text files, as it is on many Unix systems today. So, a great deal of the useful information in an ITS file system is just plain ASCII text.

To solve the problem of storing 36-bit words in 8-bit file systems, Dr. Bawden came up with the word file format. He designed it some time ago for the original problem of migrating "important" files between ITS and TOPS-20 machines, during the time when they coexisted in this building. The primary goal of the word file format is to preserve the readability of ASCII text. This translation is not one-way, so the 36-bit words can be reassembled by reversing the process. Therefore, we decided to continue using these word files for storing ITS data in our TCFS files.

This format preserves the readability of ASCII and enables us to reassemble the 36-bit words from the 8-bit bytes. However, the readability constraint prevents us from using the most trivial encoding of 36-bit words, which would split the word across 5 8-bit bytes and leave 4 bits empty. The word file format is similar to the trivial encoding, except that it converts 36-bit words that look like 5 7-bit ASCII characters to the appropriate 8-bit ASCII representation. As a result, it is more difficult to reverse-engineer a translator to reassemble the 36-bit words than in the trivial case. However, we have not complicated the encoding very much, but we have made it much simpler for files containing solely ASCII text.

4.2.2 MFD—Master File Directory

The Master File Directory (MFD) was an essential data structure on any ITS file system, so any effort to preserve the rest of the data should include this structure too. The MFD is similar to the modern-day "root directory" in a hierarchical file system, except that ITS had a flat file system with only one level of directories. So, the MFD contained a listing of all of the directories on disk. Each user had his/her own directory and each directory had a unique index number associated with it. The author of a file was recorded as the index number to his directory in the MFD. In essence, the only information the MFD provides us is the user ID number to user name mapping required to determine whose files are whose.

Decoding the MFD is simple as long as one understands some of conventions used in ITS data structures. It was our hope that by translating this in a rational manner, I would be the last person required to understand the format of the MFD.

The MFD is basically an array of usernames encoded in “sixbit”; the index number is determined by the position of the name in the array. Sixbit is a method of encoding characters in 36-bit words in which each character is 6 bits long, for a total of 6 characters per word. Sixbit does not include the lower-case characters, so all user names were in capital letters. Each user name was limited to 6 characters, so it would fit in one 36-bit word. The translation from the array of sixbit user names to an array of ASCII user names was straightforward.

If the MFD was included on a backup tape, it was always the first file. Therefore, the archivist software looks for the presence of the MFD and then decodes it as mentioned above. The raw and translated forms of the MFD are then written out in TCFS format as with the rest of the files. However, the archivist also keeps a copy of the MFD in memory so it can determine the user name associated with any files it finds later on the tape.

4.2.3 UFD—User File Directory

A crucial set of data structures that must be translated are the User File Directories (UFD’s), which are similar to the directories of today. The UFD’s are essential for preserving any file links in a directory. Furthermore, on the incremental backups, one cannot tell what files were on the original file system but were not included on that incremental without decoding this information. Therefore, it is critical to make some sense out of these data structures since we cannot depend on future archivists to decipher this raw information.

The UFD is a much more difficult structure to interpret than the MFD for two reasons. First, the UFD keeps critical information in structures that can be decoded only by interpreting PDP-10 byte pointers. Second, the UFD uses a custom method to track disk block allocation, which must be interpreted to determine file length.

The archivist software translates UFD’s into an ASCII text directory listing, and records this and the raw binary data in the TCFS output. The directory listing in Fig. 4-1 looks nothing like the one an ITS system would produce, but it is meant to include every single bit of information that was originally in the UFD. This directory listing is not particularly valuable to the average user who is searching for files in our TCFS archives: it is useful only to someone who wants to see what files existed on disk at a particular time.

4.3 Output to DAT – Rooms of Tapes to Shoe-Boxes of Tapes

Uncompressed DDS-format 4mm DAT tape was chosen for archival output. There is no particular justification for why we chose this medium over any other (although we also considered using a CD-ROM writer) and we may change if we find reason to do so. We were unable to find any studies suggesting that a particular medium was significantly more reliable or suitable to this purpose than any of the others[12]. We chose an uncompressed format because we have observed interchange problems between drive brands when using compressed formats, while the uncompressed formats have interchanged correctly.

The tapes are being written out using the TCFS standard, but a number of parameters remain unspecified in the standard. For instance, TCFS doesn’t specify a blocking factor to use for the

```
Directory Listing

Name:ALAN;AIMOVE BABYL
Pack:3
Written:11 Jul 1986 02:36:21 -0400
Block Count:4 blocks
Word Count:3300 words
Accessed:19 Dec 1989
Author:(496) ALAN
Byte Size:36
Byte Count:3300

Name:ALAN;ALAN BABYL
Pack:0
Written:28 Mar 1990 16:41:05 -0500
Block Count:74 blocks
Word Count:75705 words
Accessed:29 Mar 1990
Author:(496) ALAN
Byte Size:36
Byte Count:75705

Name:ALAN;ALAN EMACS
Pack:0
Written:12 Dec 1985 01:08:02 -0500
Link:ALAN;EINIT ^Q:EJ
Accessed:28 Mar 1990
Author:(496) ALAN
```

Figure 4-1: Example of a Translated UFD

output. We happened to arbitrarily choose 10-kilobyte blocks, since the performance concerns over blocking factor are largely irrelevant in the DDS format. We also chose to put an end-of-file (EOF) mark at the end of each old tape's worth of data. This, again, was a totally arbitrary decision; our hope is that the EOF mark will make searching the tape easier, but it is unclear that the mark will provide any performance benefits. These issues remain unspecified in the format because TCFS is completely self-delimiting and therefore insensitive to record length and EOF marks.

To reduce further data loss in the future, we are simultaneously making two copies of the tape on output. Neither of these two tapes will be publicly available, and any publicly available tape would be a third copy (once the privacy issues are settled). One set of these tapes will be stored at the lab, for any of the purposes mentioned in the introduction. Another set of these tapes will be placed in off-site environmentally controlled storage for posterity.

The original media will not be destroyed outright, but we will be less stringent in the care and handling of these "dead" tapes. They will probably be sent to off-site storage, but their eventual fate is unknown and will depend on financial constraints. There is no good reason for us to dispose of the old tapes, and no expedient way for us to recycle them. It may also be the case that any particular file that we were unable to reconstruct from these tapes during this project will be of such extraordinary interest to future generations to justify the heroic efforts required to rescue the data.

Some have suggested that a continuous path for file migration from disk to robotically-accessed media (media jukeboxes) to off-line media is the ideal[8, 11]. The TCFS format does not preclude such a structure; it just specifies a more durable format for files to withstand the ages. The tapes that we are migrating currently have been unavailable for access for years, so we are unable to estimate future patterns of access. Some files may be popular enough to require putting them on traditional hard disks or robotically-accessed media. In that event, we can think of these other storage media as a cache for the primary storage of files on off-line media.

Chapter 5

The Table of Contents—Finding a Needle in a 250 Gigabyte Haystack

As the TCFS dataset grows, we need more efficient methods of organizing and indexing the information. It is currently impractical to store the entire contents of the ITS backup tapes on-line, so we need some way to avoid having to shuffle around dozens of tapes whenever we are looking for some particular old data. The table of contents is designed to tackle just this problem.

This portion of the system isn't designed to be as robust over the ages as the TCFS format itself, and is expected to become obsolete at the same rate as any other piece of software. The design goal for this phase was to provide us with an efficient means of tracking the files in our TCFS archives now, not 30 years from now; we presume that superior means of cataloging files will surface as storage and processing power become ever cheaper. The table of contents was also designed to conserve space where it could.

5.1 How it Works

The table of contents, designed by Dr. Bawden, not only contains information about the files on the destination tape, it also contains some information regarding the source tapes and the destination tape itself. It was designed to account for the fact that many old tapes are often concatenated together across multiple new media of larger capacity.

The information stored for each file includes the file name, the creation date, and the approximate file size. It doesn't record the exact file size, but instead uses a logarithmic technique to conserve space in the table of contents. The file size is included mainly to provide a hint to a human user of this information; the exact file size can be determined by retrieving the actual file.

This format uses a delta encoding technique to reduce the amount of space used by each entry. An example of this technique is storing all the words in the dictionary in alphabetical order. If one has a sorted list of words, the difference between a given word and the next word is generally very small. Delta encoding records this difference compactly in a few bytes, which we then store in the table of contents.

For an example of how the table of contents is used, suppose a Lab member were looking for all files with the characters "EMACS" in their titles. He would first find where we stored the table of contents, which should be compact enough that it can be stored on-line in one place and won't need to be put on a tape itself. The user can then use one of our tools for decoding the table of contents format to peruse the listings. His search would be most effective using a combination of our tools and system tools for searching for the character string "EMACS". The output of this search would be a list of files, their locations, the approximate file size, and creation dates. Unfortunately, the search would be hindered by the fact that the user must decode all the table of contents files to be sure he performed his search across all our TCFS files. Due to the delta encoding scheme, it is impossible to avoid this interpretation overhead for any tapes that one searches.

5.2 What We Would Really Prefer

The table of contents format is optimized for minimizing space and for simplifying the interface as much as possible for the Archivist program, since we would prefer to keep the critical path for any tape migration as simple as possible. It is intended to provide the user with the bare minimum of information necessary to locate a file on tape without having to run through the entire stack of tapes to get to it. Unlike the TCFS format, the table of contents format is not designed to be extensible or portable. It is also not designed to be modified once it is written, since the delta encoding has no provisions for inserting and deleting entries.

Eventually it would be ideal to have all this information in a sophisticated database where complex queries are possible. This would give us the added ability to annotate the table of contents as it is being used.[1] We would also have the ability to track the status of files and tapes, so we would have an effective means of producing new copies of the physical media as the copies in use deteriorate.

One of the primary uses of a database-like interface is to record the format of a file. We have software that can guess the format of a file, but that guess may be incorrect and a static record like the table of contents format would not easily be able to accommodate that type of change. Entries can also be annotated for content, by perhaps including an abstract for any on-line papers. If two similar files exist, the database could indicate they are identical or have a human description of the differences between the two.

The primary reason, however, for moving toward a database-like interface would be to provide a system in which users can make complex queries and annotate the entries with their own data in a controlled fashion. Since, the TCFS data set is going to remain relatively static, so it will be practical to invest the time to create such an interface without fear of the accumulated annotations becoming obsolete.

In the previous example, the Lab member searching for all files with the characters "EMACS" in their titles is quite limited by using just the table of contents. If we were to have the database above, he could limit the query, for instance, to files created on Thursdays and last modified in 1971. The database's tables and query engine can be optimized much more for speed, so he would not have to parse the entire table of contents. He could also use file types to differentiate between files that were EMACS startup files written in TECO and EMACS executable binary files. He could use a helpful

[1]In much the same way that librarians used to write notes on cards in the card catalogs of days gone by.

annotation from a previous user indicating that some of the files are EMACS documentation. Our fictitious Lab member might even be altruistic and indicate that some of the files were hopelessly scrambled before they were ever converted to TCFS.

Chapter 6

Future Work

We are pleased with our prototype system. It fulfills all of our primary design goals. Never the less, there is still much room for improvements and extensions. These are discussed in this chapter.

6.1 File Classifier

One of the minor projects I have prototyped is a system that will classify many of the commonly found ITS file types. This classifier would be an ideal way to annotate the table of contents with information, hopefully saving users countless hours of searching binary files that produce false leads.

6.1.1 Stage 1: Identifying Executable Files

The first stage of the classifier determines if a file is an ITS executable binary. This is a difficult task, since there is no special "execute" flag in ITS directories and the executable format varied over time. It is necessary to decode the instructions in the file and to look for a jump instruction which occurs at the end of every executable file.

To properly decode the word file format, one should start at the beginning of the file and work one's way forward. The word file format was implemented using a small finite state machine; therefore, you can never be certain if there is some small bit of state skipped over if you start in the middle. The file classifier isn't this meticulous: it tries to save some time by using a heuristic to guess if the file is an executable. The cases covered are hoped to be robust enough to catch all of the files that are executable. The test runs have proven, on verification by hand, to be 100% successful thus far.

6.1.2 Stage 2: Differentiating Text and Binary Data Files

The second stage of the file classifier determines if a file consists of plain text or contains binary information. I developed a heuristic that gives very few false answers, but also doesn't need to scan the entire file. This heuristic involves reading the first line of the file and checking whether that the line length is reasonable and whether the first line contains any "nonsense characters" that would not occur in a typical text file. Again, a few trial runs have shown this technique to be effective.

6.1.3 Stage 3: Other File Types

The third stage of the classifier uses techniques like looking at first lines of a file or looking for distinctive or characteristic headers to determine a specific file type. For instance, in identifying a mail file one would look for "`Received:`" and "`From:`" headers, while in LISP code one would look for "lots of irritatingly silly parentheses."

6.1.4 Summary of Classifier Prototype

All in all, the file classifier uses relatively simple techniques to correctly guess the types of all of the files it has run across so far. It can be argued that the data set chosen to tune the classifier is not representative, but with some further adjustments the classifier should become a useful production tool.

6.2 Concordance

This future phase would provide us with the means to do searches that include the contents of the files, not just the header information. One would be freed from the task of shuffling tens of gigabytes of media in search of a single phrase, using a compressed concordance instead. The main obstacle to overcome is building a fast system that can do searches like this and still fit on a conventional hard disk of today.

The concordance would use the information produced by the file identifier to extract useful keywords from each individual file. For instance, only the comments would be extracted from LISP source code, leaving out reserved words and variable names.[1] The subject and body of a mail message would be extracted, ignoring the mail path headers. The result would be a list of keywords and the files in which they occur.

This list of keywords would then be sorted and encoded using a delta-encoding technique like that used for the table of contents. The keyword listing could also be split into smaller chunks, so the entire file wouldn't have to be decoded to extract a single word. One could also use hashing techniques to distribute the keywords equally across many files, so they could be found quickly.

The best use of this tool would be for a project like Dr. Agre's quest mentioned in the Overview (Sec. 2.1.2). He wanted to find when the word "foo" was first being used by our community. With the concordance available, he would only need to search it for the word "foo," thereby listing the files where the word "foo" appears. Afterward, if he decided to look for the word "foobar" that

[1] Variable names may be included separately. They may be useful, but they may produce too many false leads and bloat the concordance significantly. This issue has not been studied in enough detail to make a final decision.

search would take the same short amount of time, rather than running through the entire TCFS archive again, as he would currently be required to do.

6.3 Combining All of the Above Pieces into Production Quality Software

Eventually, people may want to make a complex request like: find all of Marvin Minsky's mail files in 1976 containing the words "forward chaining reasoning".[2] This would require a smooth interface between all of the above elements. Our hope is that some day we will be able to leave a small drawer of some variety of physical media in our lab reading room, containing all our old data. People would be able to search the entire data set from their terminals, then go to the reading room and get just what they need.[3]

6.4 Authenticity

One issue not addressed in the design of the TCFS system was authenticity. If we are intending to use these tapes as prior art in patent cases, it is always possible that someone may have accidently or intentionally changed the data in our archives. The CRC built into each TCFS file is protection against media errors, but it does nothing to show that our current archives are an accurate representation of the backup tapes[4]. It is possible that we could incorporate some hashing and digital signature scheme[7] into our framework by extending the TCFS format with a new field. This was beyond the scope of my work, but our design should not preclude the use of such a system.

[2] Note that scanning someone's mail files introduces some privacy issues and we wouldn't permit such a scenario without the owner's permission.

[3] Ideally, we would want a media jukebox that could give us on-line access to all the data 24 hours a day. At present, an auto-changer system able to contain the 262 gigabytes of "important" data (from Table 2.1) would be prohibitively expensive.

[4] Furthermore, there is nothing on our backup tapes that would lead us to believe that they are an accurate representation of the file system. Our only assurance on that front is the paper dump logs.

Chapter 7

Conclusion

"We learn from history that we do not learn from history."
–Georg Friedrich Wilhelm Hegel

In summary, the general framework that TCFS creates allows us to reduce the number of data formats we must understand. Furthermore, TCFS is a format that is specifically designed for long term storage, while most other formats are designed to optimize speed of access and space. TCFS is also intended to be easily reverse-engineered, so that when a future archaeologist finds a half-mangled scrap of magnetic tape (or whatever becomes the industry standard, such as holographic paper) she will be able to make sense of the data on that scrap, without requiring any of the surrounding context.

By moving all of our files to TCFS, it will become more economical for anyone to develop tools to help search and organize this vast amount of data. I prototyped a few such tools as part of this work. It is possible to build a single unified interface for all of the Labs' archival data. This not only allows easy access to all our old files, but simplifies maintenance. As a result, we hope that none of our data is "stranded" in some forgotten format on an unlabeled tape ever again.

Appendix A

TCFS Format Specification

This is an unpublished internal group memo by Alan Bawden discussing the specification for the Time Capsule File System in further detail.

```
                      The Time Capsule File System
                      ============================


Introduction
============

The goal is to put files in a format so that they may be preserved
indefinitely.  To achieve this goal, the problems to be addressed are
actually very similar to the problems faced in designing the Internet:

In the Internet domain, the fundamental problem is how to communicate
information from place to place in a heterogeneous network environment.  To
solve this problem, the Internet Protocol (IP) specifies some simple
abstractions (IP packets, IP addresses, etc.) that can be supported on
almost any network hardware between hosts running very different operating
systems.

In designing the Time Capsule File System (TCFS), the problem is how to
communicate information through -time-, rather than through space.  This is
an even more heterogeneous environment than IP must cope with, as we can
only -guess- what kinds of equipment the people in the future will be
using.  To solve this problem, TCFS specifies some simple abstractions that
we believe will be easy to support far into the future.

The basic abstraction is the Time Capsule File (TCF).  A TCF is a sequence
of bytes that contains a file bundled together with all of the other
information relevant to that file, such as the file's name, creation date,
author, the machine it was "dumped" from, when it was dumped, etc.  TCFs
also contain their own length and a checksum so that they make minimal
demands on the storage media they may have to inhabit during their journey
into the future.

Design Goals: * easy to recognize the parts of the format
              * extensible when we wish to add new fields
```

TCFS resembles Unix "ar" and "tar" format -- except without the bugs.

You concatenate TCFs together to form time capsules.

Each TCF is a self-contained entity -- if a collection of TCFs are all created at the same time to form an archive of a file system hierarchy, each individual TCF contains all the information necessary to reconstruct the context it was taken from.

We only assume that file systems of the future will be capable of storing sequences of 8-bit bytes. We are biased towards systems that store characters in those bytes using the ASCII encoding, but nothing depends on the survival of ASCII. We are also biased towards English. (Future digital historians will probably have to be familiar with ASCII and English, even if they don't use it themselves.)

If you prepare a large time capsule (by writing a tape, for example), you should include the most recent copy of this document. Including some relevant source code (which?) would be good too. Rosetta stones for ASCII and English should also be designed and included.

It is recommended that TCFs -not- be subjected to any form of data compression without taking steps to insure that the decompression algorithm will remain known. (Will future digital historians know the format used by Unix's 'compress' program?)

What happens if the concept of a "file" stops being meaningful?

Fields

Each TCF is made up of a sequence of fields. Each field contains a name and an associated value. Some fields specify some attribute of the enclosed file. (For example, the "Written" field records the date the file's contents were last modified, and the "Data" field contains the file's actual contents.) Other fields specify attributes of the TCF itself. (For example, the "TCF-Length" field records the length of the TCF, and the "TCF-User" field records the name of the person who created the TCF from the file.)

Comparison of field names, and other tokens called for in this document, is case insensitive. (Of course things like file names from case sensitive file systems must remain case sensitive.)

In the grammar that follows, all characters are explicitly taken to be ASCII. This is different from the normal grammars you read that describe programming languages or mail headers (to name two examples). In those grammars, an "A" can be an ASCII "A" or an EBCDIC "A", depending on the environment. But for the purposes of the Time Capsule File System, "A" always means ASCII "A" (101 octal). A program that runs on IBM mainframes that makes TCFs has to use ASCII for all the field names. Of course the "Name" field and the "Data" field can still -contain- pathnames and data in EBCDIC if that is appropriate.

Note that an effort has been made to keep TCFs readable text files. If someone without a clue about TCF format reads a TCF into a text editor, she should stand a pretty good chance of recovering the original data by just reading the text and guessing what everything means. In particular, if an ASCII text file is made into a TCF, then the resulting TCF will also be an

ASCII text file.

There are two forms in which a field can be written. In the first form a field consists of a field name, a colon (":"), the value of the field, and a terminating line character (-either- ^J or ^M). Note that in this case the field value cannot contain either a ^J or a ^M.

In the second form a field consists of a field name, a semicolon (";"), a sequence of decimal digits that specify the length of the field's value, a colon (":"), and finally the value of the field. Note that in this case, the field value can contain arbitrary 8-bit data, since the length is explicitly specified. In a typical TCF, only the data field is written in this form.

Some initial whitespace characters are allowed before the field name.

A complete TCF starts with a recognizable string, followed by a sequence of fields. There is no restriction on the order of the fields, except that the TCF-Length field must be the first one.

Field Syntax

```
  <Alpha> := "A" | "B" | ... | "Z" | "a" | "b" | ... | "z"
  <Digit> := "0" | "1" | ... | "9"
  <Alnum> := <Alpha> | <Digit>
  <Space> := " " | ^I | ^J | ^K | ^L | ^M
  <LineChar> := ^J | ^M
  <NameChar> := <Alnum> | "-"
  <Name> := <NameChar>+
  <Value> := ":" (<Char> - <LineChar>)* <LineChar>
           | ";" <Digit>+ ":" <Char>^n
  <Field> := <Space>* <Name> <Value>
  <TCF> := "[  -----------------------  "
           "Begin Time Capsule File"
           "  -----------------------  ]"
           <Field>*
```

Individual Fields

Note that not all of these fields need be present. This catalog is as long as it is only so that we may all agree what a field means when it is encountered.

```
TCF-Length:<Integer>    Length of the entire capsule, starting from the "["
                        in the recognition string.

                        This field must be present and must come first.

TCF-Checksum:<Chars>    Chosen so the the entire capsule (as measured by
                        the TCF-Length field) has a 32-bit CRC of 0.

                        This field must be present.  Typically, this field
                        will be last.

TCF-Date:<Date>         The date that this TCF was made.
                        The meaning of some other fields are necessarily
```

relative to this date. See Date Syntax below.

This field must be present.

TCF-Host:<Hostname> — The host where this TCF was made. <Hostname> is relative to TCF-Date (see Host Names below). Other fields are relative to TCF-Host.

This field must be present.

TCF-User:<Username> — The user who made this TCF. <Username> is relative to TCF-Date and TCF-Host (see User Names below).

TCF-Type:<Type> — What type of TCF this is.

The type "Archive" means that this TCF contains a file captured from some file system in order to preserve it. "Archive" is the only type there is, for now. Someday we might put things in TCFs other than files we want to preserve.

Programs that -read- TCFs must check the TCF-Type field to be certain that they properly understand the rest of the TCF.

This field must be present.

Capture-Date:<Date> — The date that the contents of this TCF was captured from its original location. For example, if this TCF was recovered from a dump tape, this will be the date the tape was written, while TCF-Date will be the date that the TCF was made from the tape.

Defaults to TCF-Date.

Capture-Host:<Hostname> — The host that the contents of this TCF was captured from. In the case of a dump tape, this will be the machine that wrote the tape, while TCF-Host will be the machine that read the tape and made the TCF.

Defaults to TCF-Host. <Hostname> is relative to Capture-Date (see Host Names below).

Capture-User:<Username> — The user that captured the contents of this TCF. In the case of a dump tape, this will be the hacker who wrote the tape, while TCF-User will be the hacker that read the tape and made the TCF.

Defaults to TCF-User. <Username> is relative to Capture-Date and Capture-Host (see User Names below).

Archive-Title:<Title> — If this TCF is part of a logically related set of files, the entire collection may have been given a descriptive title.

System:<Sysname> — ITS, Multics, Tops-20, Lisp-Machine, Unix, VMS, MacOS, DOS, etc.

TCF-Date relative, since the authority who assigns system names may change over time. Currently that authority is the same as the authority who assigns host names (see Host Names below).

The meaning of some fields are system dependent. In particular, the System field determines the algorithm for recovering the original data from the rest of the fields in the capsule. Typically this involves interpreting the contents of the Data field.

When the TCF-Type is Archive, this field must be present.

Tape-Info:<Info>

If the contents of this TCF are also contained on a dump tape, this field describes that tape. The format of <Info> is system dependent, since different systems have different conventions for labeling tapes.

For an ITS tape <Info> contains:

```
<Name>, <Type>, <Tape#>, <Reel#>, <File#>
```

Note that the naming scheme for labeling tapes is probably relative to the Capture-Host, and perhaps to the Capture-Date as well.

Tape-Location:<Text>

Contains information about the physical location of the dump tape in the Tape-Info field, at the time the file was copied into this TCF. For example, an ITS tape stored off-site at InStar might have a Tape-Location field of "InStar MCG06465".

Root:<Pathname>

If this TCF is part of a sub-hierarchy of a file system that was captured all at once, this is the pathname of the root of that sub-hierarchy. This must not be a relative pathname. By computing the -difference- between this pathname and the contents of the Name field, an isomorphic copy of the captured sub-hierarchy can be reconstructed later if need be.

Defaults to the root of the file system on the Capture-Host.

Name:<Pathname>

The name that the data was given at the source.

The interpretation of this field is system dependent. The pathname is given in the native pathname syntax for the given system (using native quoting conventions if needed).

This may -not- be a relative pathname. (See the Root field.)

When the TCF-Type is Archive, this field must be present.

Created:<Date>
Creation of the file itself. (The Mac file system has this.) If not present, taken to be the same as the Written date. See Date Syntax below.

Written:<Date>
Creation of the -contents-. (Unix mtime, ITS cdate. Virtually every file system in the world has this.) This is the date you -really- want. See Date Syntax below.

When the TCF-Type is Archive, this field must be present.

Changed:<Date>
The last time -any- aspect of the file changed. (Unix ctime.) If not present, taken to be the same as the Written date. See Date Syntax below.

Accessed:<Date>
The last time anybody looked at the file. (Unix atime, ITS rdate.) If not present, taken to be the same as the Changed date. See Date Syntax below.

Author:<Username>
The person primarily responsible for the contents of the file. (Unix owner, ITS author.) See User Names below.

Reader:<Username>
The person who last -read- this file. Tenex had this.

Group:<Groupname>
Unix group. Hopefully as a name, not a number.

Mode:<Octal>
Unix mode -- as an octal number.

Reference-Count:<Number>
The reference count of this file or directory. (Unix has this.) Defaults to 1.

Pack:<Number>
ITS disk pack. (Yes, ITS pack numbers were customarily written in decimal.)

Account:<String>
Tops-20 account.

Kept-Versions:<Number>
Tops-20 and LMFS generation retention count.

Expunge-Interval:<Seconds>
For Tops-20 and LMFS directories. The number of seconds between automatic expungings.

Expunged:<Date>
For Tops-20 and LMFS directories. The time when the last expunge took place. See Date Syntax below.

Flags:<Flags>
<Flag>, <Flag>, <Flag>, ...
Where <Flag> is one of:

Deleted
For file systems supporting soft deletion.

Dont-Delete
It was an error to delete this file.

Dont-Dump
This file was not supposed to be

dumped.

	Dont-Reap	This file was to be considered valuable (ITS "$" flag).
	Not-Dumped	This file had not been dumped (ITS "!" flag).
	Offline	This file had been moved to archival storage.
	Temporary	This file was marked as temporary.

Comments:<Text> Additional information of any form. May appear more than once.

Type:<Filetype> Directory, File, Link, Hard-Link, ... Defaults to File.

Byte-Size:<Number> The size of the bytes that make up the file (in bits). Defaults to 8.

Block-Size:<Number> The size of the blocks that make up the file (in bits).

Length:<Number> The length of the original file measured in bytes.

Blocks:<Number> The length of the original file measured in blocks. If this field is present, then Block-Size must also be present.

Data:<Data> The actual contents, encoded somehow as a sequence of bytes.

The algorithm for converting this back into its original form is system dependent (and may require data from other fields), but an effort should be made to keep the data in a form where files that were considered text at the source are encoded here as readable (or close to readable) ASCII.

Link-To:<Pathname> If this is a link rather than an ordinary file, this is the pathname that is the target of the link. This might be a relative or an absolute pathname.

This is used for both hard and soft links. In the case of a hard link, this is the name under which a previous TCF of this file was created (in the same collection).

Directory-Listing:<Listing>

If this is a directory rather than an ordinary file, this field contains an ASCII listing of the contents of the directory in some readable fashion.

The Data field may or may not also be present. The Data field, if present, contains some system-specific (perhaps binary) representation of the directory, while the Directory-Listing field contains something future historians can read without having to interpret the Data field.

Host names

The naming scheme for hosts may change over time, and so host names are always relative to some date. The current (in 1992) rules are:

1. Hosts are named using fully qualified domain names.

2. Arpanet hosts from the days before the domain name system are named using their Arpanet host names.

3. For machines that don’t have names assigned by either of the rules above, names will be assigned by a single naming authority. Currently that authority remains with the author of this document (Alan Bawden).

In the future, naming authority may be further delegated.

It is quite likely that the concept of a host will cease to be meaningful someday fairly soon. We’ll cross that bridge when we come to it.

User Names

The naming scheme for users may also change over time, and so user names are also date relative.

Currently (1992) there is no user naming system in place that is independent of host naming, so for now a user name is always relative to some host name.

Some systems assign a number (e.g. Unix’s "uid") to each user. Such numbers are allowed to appear as user names in TCFs, but this should be avoided unless it is absolutely necessary. In fifty years it will be a lot harder to figure out who "5098" was than who "alan" was.

Date Syntax

```
<Date> := <Day> " " <Time> | <Day>

<Day> := <DD> " " <Mon> " " <YYYY>
<DD> := "1" | "2" | ... | "31"
<Mon> := "Jan" | "Feb" | ... | "Dec"
<YYYY> := "1900" | "1901" | ...

<Time> := <HH> ":" <MM> ":" <SS> " " <Zone>
<HH> := "00" | "01" | ... | "23"
<MM> := "00" | "01" | ... | "59"
<SS> := "00" | "01" | ... | "59" | "60"
<Zone> := ( "+" | "-" ) <HH> <MM> | "Local"
```

This is actually more likely to be portable into the future than something like number-of-seconds-since-1-Jan-1900. Note that timezones are not named, but are always given as the offset from GMT. (Where "-0500" is EST, "-0400" is EDT, "-0800" is PST, "-0700" is PDT, "+0000" is GMT, etc.) A time zone of "Local" indicates that the time zone was unknown -- some files in some file systems may only be marked with wall clock time.

The BNF doesn't capture all the constraints. The <Day> must make sense for the <Month> and <Year>. <SS> can be larger than "60" if there is a leap second, or "59" can be skipped if there is an anti leap second.

The <Year> cannot be abbreviated (as in "92" instead of "1992").

This format has been chosen to be similar to RFC822 date format, but it is not identical.

Checksum

Each TCF is formed so that when considered as a polynomial over the integers mod two, the entire TCF is congruent to 0 modulo

$$x^{32} + x^7 + x^3 + x^2 + 1.$$

This is done in bigendian order. That is, the very first byte in the TCF is taken to be the highest order 8 terms in the polynomial, and the very last byte is lowest order 8 terms. The lowest order bit of the last byte gives the coefficient for x^0 term.

(This is a fairly standard 32-bit CRC that any textbook on error detecting and correcting codes will explain in better detail than I can do here.)

(This particular 32-bit CRC was chosen not because of its error correcting properties (which are not needed for this application), but because it is particularly fast to compute one byte at a time.)

Bibliography

[1] Crocker, D., "Standard for the Format of ARPA Internet Text Messages," Request for Comments (RFC) 822, Department of Electrical Engineering, University of Delaware, August 1982.

[2] Cuthill, E. and P. Wang, Introduction from *1979 MACSYMA Users' Conference*, 1979.

[3] Drapeau, A. and R. Katz, "Striped Tape Arrays," Technical Report UCB/CSD 93/730 Computer Science Division, University of California, Berkeley, 1993.

[4] Eastlake, D., "ITS Status Report," AI Memo 238, Massachusetts Institute of Technology, 1972.

[5] Fateman, R., Preface from *Proceedings of the 1977 MACSYMA Users' Conference*, NASA CP-2012, 1977.

[6] Ford, D., "Media Management in Database Tertiary Storage," Technical Report RJ9710, IBM Almaden Research Center, 1994.

[7] Graham, P., "Intellectual Preservation: Electronic Preservation of the Third Kind," *The Commission of Preservation and Access*, 1994.

[8] Katz, R., T. Anderson, J. Ousterhout, and D. Patterson, "Robo-line Storage: Low Latency, High Capacity Storage Systems over Geographically Distributed Networks," Technical Report UCB/CSD 91/651 Computer Science Division, University of California, Berkeley, 1991.

[9] Legato Systems, Inc, "NetWorker Data Formats," March 1992.

[10] Miller, E. and R. Katz, "An Analysis of File Migration in a Unix Supercomputing Environment," Technical Report UCB/CSD 92/712, University of California, Berkeley, 1992.

[11] Mott-Smith, J., "The Jaquith Archive Server," Technical Report UCB/CSD 92/701 Computer Science Division, University of California, Berkeley, 1992.

[12] Robertson, K., "Gigabyte Storage – Backup and Beyond," *Proceedings of the Thirteenth National Online Meeting*, Learned Information, Inc., 1992.

[13] Rothenberg, J., "Ensuring the Longevity of Digital Documents," *Scientific American*, January 1995, pp. 42-47.

[14] SIDF Association, "System Independent Data Format V1.0," 1994.