Introduction to VVencode/VVdecode

1 Introduction to VVcode

Reliable and faithful exchange of binary files between computers over networks is a well-known problem, especially if the computers use different operating systems and are connected to different networks via a gateway. Unfortunately inter-networking and electronic mail are very much children of the 1960s: they might have had to wait until the 1970s for their naissance, but their progenitors were mentally locked-in to the concept of the 7-bit ASCII code for conveying textual information. The TeX community has long been aware of this problem when trying to exchange "machine-independent" '`.dvi`' files and font-related data such as '`.tfm`' and '`.pk`' files. It has sometimes been possible to exchange this binary data by using encoding schemes that allow the data to be represented using a subset of the seven-bit ASCII character set.

Academics and authors in many fields have hitherto been able to pass '`.tex`' files back-and-forth by electronic mail—apart from a few minor quirks and blemishes, such TeX source files pass unharmed across the planet's networks. Problems are encountered when mail passes through certain gateway machines which introduce irreversible character corruptions. Particularly notorious is the Janet/Bitnet gateway which has the unfortunate habit of converting '^' to '~' and '~' to '`%`': since it leaves '`%`' itself unaffected, this makes recovery of the original file a non-trivial exercise. It sometimes also changes the brace characters '`{}`' into odd characters above 128: this is particularly embarrassing, of course, for '`.tex`' files!

For some years many TeX users, particularly those working in languages other than English, and thus familiar with character set encodings containing other than the basic ASCII set, have been agitating for TeX to be able to handle input in their mother tongues, using their own languages' character sets. In 1989, Knuth announced TeX V3, and implementors world-wide beavered away to bring each implementation up-to-date. TeX V3 now supports eight-bit character sets and so '`.tex`' source files are now effectively 'binary' files and will therefore suffer from the same exchange problems experienced with '`.dvi`' files.

All those authors that had previously been able to cooperate, despite being separated by hundreds or thousands of miles, might once again be forced to entrust floppy disks to the vagaries of the world's postal systems (although one shouldn't underestimate the bandwidth of the Royal [or other] Mail system).

Unless or until the various e-mail protocols, networks and software are converted to support uncorrupted transmission of characters codes `0x20` ... `0x7e` and `0xa1` ... `0xfe`, it will have to become the norm for '`.tex`' sources to be encoded for transmission by e-mail.

This problem is of course well known outside the TeX community.

1.1 The Aston Archive

The author is a volunteer assistant to Peter Abbott in running the world's principal repository of TeX-related material at Aston University in Birmingham. The archive (host: `TeX.Ac.Uk`) holds several hundred megabytes of text and binary files including:

- program sources for TeX, `METAFONT`, DVI drivers and many other utilities;

- binary executables for a variety of popular operating systems (e.g. Atari, Macintosh, MS-DOS, Unix, VAX/VMS and VM/CMS);
- `METAFONT` sources for Computer Modern and other fonts;
- binary font files (mainly '`.tfm`' and '`.pk`') for a number of different output devices;
- text macro and style files.

The archive provides access to these files via the following services:

- NIFTP[1] from Janet hosts—typically 300 megabytes of data are transferred every month; this would probably be much greater if we were not limited by the bandwidth of our 9600Bd connection to Janet.
- FTP and Telnet access from Internet hosts.
- Interactive browsing service via Janet PAD, including the facility to send files out using NIFTP (and later FTP).
- Interactive browsing service via dialup modem lines, including the facility to download files using Kermit and similar protocols.
- An e-mail file server which typically sends 150 megabytes of data per month to sites all over the world (though predominantly to EARN/Bitnet sites).
- A magnetic media distribution service via surface carriers. Copies of the entire archive have been sent to embryonic TeX communities in Czechoslovakia, Hungary and Poland.

We have experienced many problems trying to support all of these file types, operating systems and access methods. The e-mail file server clearly needs a reliable method of encoding files if its many customers are not to be denied access to the non-text files in the archive.

Binary files such as '`.pk`' font files are stored in different ways to accommodate the requirements of the different operating systems supported. Currently we maintain multiple font directory trees for the Macintosh, MS-DOS, Unix and VAX/VMS with all the attendant problems of synchronization, disk space and archivists' time. We need a single storage format which allows export to all of our supported operating systems.

1.2 Specification for a Coding Scheme

In mid-1990, the archivists came to the conclusion that a universal encoding scheme was required to accommodate the many different kinds of file and file organizations that needed to be supported by the archive.

---

[1] Network Independent File Transfer Protocol — in the UK, one does not perform the pseudo-login that Internet users are accustomed to using with the FTP protocol: instead, one issues a "transfer request" for a file to be sent to or from the remote machine — the transfer itself takes place asynchronously. One nice consequence is that such transfers can be queued for overnight execution, leaving daytime bandwidth free for e-mail and true remote interactive logins.

Niel Kempson formulated the first draft of this specification in mid-1990; the requirements of the encoding scheme may be summarized as follows:

Preserving File Structure

It is insufficient, especially for an archive holding text and binary files for a variety of machine types, merely to encode data simply as a stream of bytes:

- Virtually all operating systems (except Unix) make a distinction between binary and text files, so the coding system should recognize and maintain this distinction.
- Unix and most PC-based operating systems treat files as streams of bytes with no further structure imposed. On the other hand, certain widely-used operating systems (e.g. VAX/VMS and VM/CMS) have record-oriented file systems where different types of file are stored in a format appropriate to the type of file[2].

  For these operating systems, we consider it essential that the encoding scheme should identify, preserve and record the most commonly used file organizations. The decoding program should be able to use this information to create the output file using the organization appropriate to the operating system in use. If the information is of no consequence to the receiving system, the default file structure (if any) should be created. If the encoding system does not have structure in its files, the receiving system may provide suitable defaults automatically. In all cases the programs should permit the user to override or supplement file structure information.
- Whenever possible, these details of structure should be determined automatically by the encoding program; at the very least, an indication of whether the file is text or binary shall be provided, even under an operating system such as Unix that need make no such distinction for its own use, to allow decoding to an appropriate file organization on those systems that *do* make such a distinction.

Coding Scheme

Whatever method is used must allow encoded data to be e-mailed:

- It should be possible to specify the coding table to be used to encode the data. The coding table used shall be recorded with each part of the encoded data.
- If a recorded coding table is found while decoding, it should be used to construct an appropriate decoding table. Simple one-to-one character corruptions should be corrected as long as only one of the input characters is mapped to any one output character.
- The recommended encoding uses only the following characters:

  ```
  +-0123456789
  abcdefghijklmnopqrstuvwxyz
  ABCDEFGHIJKLMNOPQRSTUVWXYZ
  ```

---

[2] It is often argued that the increase in efficiency more than offsets the increase in complexity.

Such an encoding as originally used for XXcode has been shown to pass successfully through all the gateways which are known to corrupt characters.

Integrity of Encoded Data

We want to ensure that the *whole* encoded file passes through the e-mail network.

- Encoded lines should be prefixed by an appropriate character string to distinguish them from unwanted lines such as mail headers and trailers. Whilst not essential, this feature does assist the decoding program in ignoring these spurious data.
- Lines should not end with whitespace characters as some mailers and operating systems strip off trailing whitespace.
- The encoding program should calculate parameters of the input file such as the number of bytes and CRC and record them at the end of the encoded data.

  The decoding program should calculate the same parameters from the decoded data and compare the values obtained from those recorded at the end of the encoded data.

Making Files Mailable

A mechanism is needed to overcome some gateways' refusal to handle large files.

- The encoding program should be able to split the encoded output into parts, each no larger than a maximum specified size. Splitting the output into smaller parts is useful if the encoded data is to be transmitted using electronic mail or over unreliable network links that do not stay up long enough to transmit a large file. The recommended default maximum part size is 30kB.
- The decoding program should be able to decode a multi-part encoded file very flexibly. It should *not* be necessary to:
  1. strip out mail headers and trailers;
  2. combine all of the parts into one file in the correct order;
  3. process each part of the encoded data as a separate file.
- In addition any file specifications from the operating system on which the VVE file was created must not prevent the file from being decoded.

Miscellaneous

Further considerations include:

- Support for character sets other than ASCII is essential if the encoding scheme is to be useful to IBM hosts. The encoding program should label the character set used by the encoded data, and both encoder and decoder should enable the conversion between the local character set and another character set. For example a user on an EBCDIC host should be able to encode text files for transmission to another EBCDIC host, or to convert them to ASCII before encoding and transmission to an ASCII host. Similarly, that user should be able to decode text files from ASCII and EBCDIC machines, creating EBCDIC output files.

- Where possible, the original file's timestamp should be encoded and used by the decoding program when recreating the file: this will permit archives to retain the originator's time of creation for files, and thus permit the users (not to mention the archivists) to identify more clearly when a new version of a file has been made available. Timezones should be supported where possible.
- The encoding and decoding schemes should be able to read and write files that are compatible with one or more of the well established coding schemes (e.g. UUcode, XXcode).
- The source code for the programs should be freely available. It should also be portable and usable with as many computers, operating systems and compilers as possible.

1.3 The Search Commences

Naturally, the first step was to examine the existing coding schemes in comparison with the above ideal specification. Such schemes fell into two broad classes: *portable schemes*, which were intended to permit the encoding of files on any computer architecture into a form that could be transmitted electronically, and decoded on the same or a different architecture; and *platform-specific schemes*, which provided rather better support for transferring files between two computers using the same architecture and operating system.

1.3.1 Portable Coding Schemes

The most commonly used coding schemes supported by a variety of platforms are:

- `BOO`
- `UU`
- `XX`

Most implementations of these schemes known to the authors are designed for use with stream file systems. These programs have no means of recording, let alone preserving, record structure and are thus unsuitable for our purposes. This is not surprising since `UUcode` and its mutation `XXcode` were developed specifically for exchanging files between Unix systems. In fairness to these schemes, they are well suited to the transmission of text files and certain unstructured binary files.

Standard `UUcode` encodes files using characters ‘ ’ …‘_’ of ASCII. This can result in one or more spaces appearing at the ends of lines: some mailers decide that this is information not worth transmitting, with consequent inability to reconstruct the original file.

Files containing characters such as ‘^’ are often irreversibly corrupted by mail gateways; this problem led to the development of `XXcode` which uses a rather more robust character set, namely:

```
+-0123456789
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The encoding table used is recorded with the encoded data to allow the detection of character corruptions, and the correction of reversible character transpositions. Whilst superficially a step forward, `XXcode` offered little more than most existing versions of `UUcode`, which already supported coding tables. Its major contribution was in formalizing the encoding table, and in particular its default table was proof against all the known gateway-induced corruptions.

1.3.2 Platform Specific Coding Schemes

Encoding schemes have been developed to support transfer of files possessing some structure which therefore cannot be reconstructed correctly when encoded by the portable schemes. When the encoding and decoding programs of such a platform specific scheme are each used on the same computer and operating system type, files may be encoded and transmitted with a great deal of confidence that the decoded file will reproduce the original's structure and attributes in their entirety.

Examples of such programs are `TELCODE` and `MFTU` for VMS, `NETDATA` for IBM mainframes, and `Stuffit` and `MacBinary` for the Macintosh. But these programs have the major disadvantage that they have each been implemented *only* on the single architecture for which they were designed: thus the only two of these schemes that could be used on the VMS-based Aston Archive would be of minimal interest elsewhere!

The Archive's content is in some respects artificially inflated by the presence of '`.hqx`' files for Macintoshes, '`.boo`' for MS-DOS, etc., which have to be held in pre-encoded form for transfer by those requiring them.

1.4 VVcode is Born

Realizing that none of the existing portable schemes were close enough to our ideal, an early version of our specification was circulated on various mailing lists by Niel Kempson towards the end of 1990. When the anticipated "nil return" was all that resulted, Brian Hamilton Kelly went ahead and created a rudimentary `VVencode` by modifying an existing VAX Pascal implementation of `uuencode`. After generating the companion `VVdecode`, he then re-implemented the programs in Turbo C under the MS-DOS operating system on the IBM-PC, and thereby was able to prove that the new scheme was both viable and sufficient.

This version didn't support file formats, time stamping, file splitting, character sets or CRC checking.

1.4.1 A Production VVcode

Following the minor feasibility study, Niel Kempson re-engineered the pair of programs from scratch (adding certain features of the evolving specification), paying particular attention to making the code portable across a wide variety of operating systems. Particular care was taken to avoid the use of supposedly "standard" C functions that experience had shown behaved differently under individual manufacturer's implementations, or were even non-existent in some. Therefore the code may sometimes appear to be performing certain operations in a very long-winded way; it's very easy to look at it and say "why didn't the author use the `foo()` function, which does this much more efficiently?", but this function may not even exist under another implementation of C, or behave in a subtly different manner.

The core functions of `VVcode` are implemented as a collection of routines written in as portable a fashion as possible, and a separate module of a few operating system specific routines for file I/O, timestamping, command-line or other interface, etc. Porting `VVcode` to a new platform should require only that this latter module be re-implemented, in most cases by adapting an existing one.

`VVcode` implements all of the features listed in the specification, apart from the ability to generate `UUcode` and `XXcode` compatible files. However, the decoding program is backwards compatible and can decode files generated by `UUcode` and `XXcode`.

1.4.2 Arguments against VVcode

When the advent of the `VVcode` system was first aired in the various electronic digests, some heated debate followed along the lines that a new encoding scheme was unnecessary, since `UUcode/XXcode` sufficed *for them*. However, all these correspondents were Unix users who had interpreted the '`VV`' as meaning "Vax-to-Vax" by analogy with '`uu`'[3] and who felt that such a scheme should be private to VAXen. The authors' reply was to the effect that the encoding scheme was intended to support the needs of archives like Aston's, and as such, had to provide

1. an automated tool (it would be somewhat difficult to expect our users to be able to tell the encoder what sort of file structure it was handling, when this concept was entirely alien to many of them);
2. facilities to encode binaries for many operating systems;
3. mail server features, such as splitting of large files;
4. operation across the widest possible combination of platforms.

The overhead of using the `VVcode` system is at most a couple of hundred bytes over using `UUcode`, and the extra functionality and *universality* with respect to `UUcode` or `XXcode` thereby comes almost for free.

1.5 Availability of VVcode

At present, the `VVcode` system is only available in C, but it has been shown to run successfully on the following combinations of hardware, operating system and compiler:

---

[3] '`V`' was chosen simply because it followed '`U`'; at one time, we'd seriously considered calling it `YAFES` — Yet Another File Encoding Scheme!

Macintosh At the time of writing (May 1991) John Rawnsley of the University of Warwick had commenced development of a Macintosh port, which will encode the resource and data forks in a manner that will permit the former to be ignored by non-Macintosh systems.

MS-DOS

- IBM PS/2, PC (and clones); MS-DOS 3.3, 4.01, 5.00; Borland Turbo C 1.5, 2.0, Borland C++ 1.0, 2.0, 3.0 and Microsoft C 5.1, 6.0

OS/2

- IBM PS/2, PC (and clones); OS/2 2.0; Microsoft C 6.0 and GNU C 2.1

Unix

- Sun 3; SunOS 3.x and 4.0.3; native C and GNU C
- Sun Sparcstation 1; SunOS 4.1; native C and GNU C
- SCO Unix V/386 v3.2.2, Microsoft C compiler

VAX/VMS

- All VAXen; VMS 5.2–5.4-1; VAX C V3.0–V3.2 and GNU C 1.40

VM/CMS

- VM/CMS; Whitesmith C compiler v1.0 (This implementation was ported by Rainer Schöpf; basing it upon the Unix implementation, this took him about one day.)

Table of Contents