

The International Journal of Digital Curation

Issue 1, Volume 6 | 2011

Migrating Home Computer Audio Waveforms to Digital Objects: A Case Study on Digital Archaeology

Mark Guttenbrunner, Mihai Ghete, Annu John, Chrisanth Lederer, Andreas Rauber,
Vienna University of Technology,
Vienna, Austria

Abstract

Rescuing data from inaccessible or damaged storage media for the purpose of preserving the digital data for the long term is one of the dimensions of digital archaeology. With the current pace of technological development, any system can become obsolete in a matter of years and hence the data stored in a specific storage media might not be accessible anymore due to the unavailability of the system to access the media. In order to preserve digital records residing in such storage media, it is necessary to extract the data stored in those media by some means.

One early storage medium for home computers in the 1980s was audio tape. The first home computer systems allowed the use of standard cassette players to record and replay data. Audio cassettes are more durable than old home computers when properly stored. Devices playing this medium (i.e. tape recorders) can be found in working condition or can be repaired, as they are usually made out of standard components. By re-engineering the format of the waveform and the file formats, the data on such media can then be extracted from a digitised audio stream and migrated to a non-obsolete format.

In this paper we present a case study on extracting the data stored on an audio tape by an early home computer system, namely the Philips Videopac+ G7400. The original data formats were re-engineered and an application was written to support the migration of the data stored on tapes without using the original system. This eliminates the necessity of keeping an obsolete system alive for enabling access to the data on the storage media meant for this system. Two different methods to interpret the data and eliminate possible errors in the tape were implemented and evaluated on original tapes, which were recorded 20 years ago. Results show that with some error correction methods, parts of the tapes are still readable even without the original system. It also implies that it is easier to build solutions while original systems are still available in a working condition.¹

¹ This paper is based on the paper given by the authors at the 6th International Conference on Preservation of Digital Objects (iPres 2009), October 2009; received January 2010, published March 2011. The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. ISSN: 1746-8256 The IJDC is published by UKOLN at the University of Bath and is a publication of the Digital Curation Centre.



Introduction

With storage media technology constantly changing, devices for reading certain media become obsolete. However, not only the devices, but also specimens of the computer systems needed to access the data stop working every day. Accessing data on an 8-inch floppy disc requires not only a disc drive, but also a system to connect it to, together with the right software to interpret the data on a logical level. This is all necessary to access the information contained on the floppy disc.

Estates submitted to archives now and in the future not only contain information on paper but also include digital data (e.g., digital diaries) on various storage media. As we have to deal with this data at the time it is ingested in the archive, we have to find methods to extract data from already obsolete storage media and make sure we are able to migrate it to a format that can be used in the digital archive.


A popular storage medium for home computers in the early 1980s was audio tape, also called a compact cassette². As devices that could read and write audio tapes were readily available in most households for recording and playing music, some home computer systems featured connectors for the headphone and microphone jacks of such audio systems that could be used to store and retrieve data. The data was first converted into an analogue waveform by the computer system and then recorded to a standard magnetic audio tape through the microphone input of the audio device. To retrieve the data, the tape was played back on the audio device and the audio waveform was read by the computer system through the headphone connector. High amplitudes in the waveform were recognised and the resulting data decoded into a binary stream that was then usable by the original system.

As there was no standardised format across the different home computer platforms, tapes are usually only readable by the system that was used to write them. To retrieve the data from such storage media today we therefore need access to the original system, a tape drive and a person with the knowledge of handling the original system. As audio tape playing devices are still readily available and made of standard components, it is fairly easy to track down a device able to replay tapes. Getting a working specimen of the original system, as well as a person to handle the system, can be a more difficult task.

In this paper we present a methodology and an application we developed that allows us to retrieve data written by the Philips G7400 home computer system without access to the original system. By using a tape drive and a common personal computer with a sound card, we record the audio output of the tape drive and decode the waveform. The resulting bit-stream is then interpreted and the data is migrated to a non-obsolete format that can be ingested into an archival system. Our application may also be used for carrier refreshment - storing the original data again onto an audio tape, correcting errors introduced by the decay of the waveform on the original tape.

We first present related work on the topic and present the original system this work is based on. In the following section we explain how the physical format was reengineered. After that, we show how the different types of data are stored logically.

² Compact Cassette – Wikipedia: http://en.wikipedia.org/wiki/Compact_Cassette.



Next, we present an application we developed that allows us to migrate data from waveforms, together with the two methods we used to convert the recorded waveform into binary data. We show the results of using each method on original tapes, including how much of the data we were able to recover. Finally we present our conclusions.

Related work

The UNESCO guidelines for the preservation of digital heritage (Webb, [2005](#)) list four layers where digital data can be threatened.

Audio tapes are magnetic tapes and are subject to various threats on the physical level, as described by Bhushan ([1992](#)). By converting the analog waveforms to digital waveforms and storing them as digital audio-files on current systems we can avert the immediate threat on the physical layer.

To prevent loss of data on a logical level it is necessary to re-engineer the encoding of digital bits in the analog audio signal. In a report by Ross & Gow ([1999](#)) an experiment with a Sinclair Spectrum is described, where audio data was migrated to a corresponding binary stream, which could then be interpreted using an emulator of the real system.

However, to separate the digital objects from their original environment, the bit-streams have to be interpreted in such a way as to extract the conceptual object from the logical bit-stream. By extracting the content and saving it to a format which is not obsolete at the time of migration, we can transform the data to a format that is accessible without the original hardware. No expert is needed to operate the original system, as is necessary with emulation as a preservation strategy.

The essential elements of the digital object can then be added on ingest in an archival system.

On the system we used for our case study, BASIC was used as a main programming language. Source code is a significant property of software and can be necessary to interpret the data stored by applications, and is also necessary if software is migrated for preservation purposes (Matthews et al., [2008](#)). As the system is used as a video game console as well, some of the programs are video games. This provides us with a situation where migration would be a possible solution to preserve some video games for the system (Guttenbrunner, Becker, & Rauber, [2009](#)).

Original System

For our case study we decided to use the Philips G7400³. Originally designed as a video game system with a keyboard, it can be extended to become a home computer with a Microsoft BASIC operating system using the C7420 expansion cartridge. This cartridge also features three connector cables for data input, data output and a remote controlling signal used to start and stop the audio tape, if supported by the tape player.

³ Philips G7400 - Wikipedia: http://en.wikipedia.org/wiki/Philips_Videopac_%2B_G7400.

The system was chosen as it is already very hard to find specimens in working condition, so there is an imminent threat of permanently losing the data saved with this system. It also met our second criterion: off-the-shelf audio recorders and tapes could be used for storage purposes.

The Philips Videopac+ G7400 Video Game Console System

In 1968, Ralph Baer created the prototype for the first home video game called Brown Box (Baer, [2005](#)). The American company Magnavox released the system to the public in 1972 as the “Magnavox Odyssey”. The system used cartridges that did not store any information but interconnected different electronic parts to create the desired built-in games. Only black/white output was created and by applying different overlays on the TV for every game, the impression of colour was created.

In 1978, the successor to the Magnavox Odyssey, the Magnavox Odyssey2, was sold in America (Herman, [2001](#)). In Europe the system was sold by Philips under the name “Videopac G7000” (Forster, [2009](#)). This system used an Intel 8048H CPU and the custom “VDC” (video display chip) Intel 8244 to display various different onscreen objects.

Magnavox also started to develop a successor to the Odyssey2, the Odyssey3. The system was equipped with a more powerful graphics chip than its predecessor, but was made backwards compatible to the Odyssey2. It was never sold to the public, even though some prototypes⁴ were found by video game collectors in yard sales in the area of Magnavox’ head quarters in Knoxville, TN, USA. In Europe, the Videopac G7000 system was more successful than the Odyssey2 in the US, so Philips released the Odyssey3 under the “Videopac+” brand as the “Philips G7400” (shown in Figure 1) in 1983. The system was able to use all the cartridges for the original system, but some additional cartridges only playable on the G7400 were also released. As home computers got more popular during that time, and the Philips Videopac systems were equipped with a keyboard all along, an additional cartridge that converted the system to a fully-fledged home computer was released.



Figure 1. Philips Videopac+ G7400 game console system.

The Philips C7420 Home Computer Module

In 1983, shortly after the release of the Philips G7400 game system, Philips released the Home Computer cartridge as an add-on to convert their console system to a fully-fledged home computer. As the built-in 8048 processor was not powerful enough for this task, the system itself was used for input and output only, and the computing was done mainly by a Zilog Z80 micro processor running at 3.754 Mhz and

⁴ Odyssey 3 Prototypes: http://www.dieterkoenig.at/ccc/po/s_po_o3.htm.



stored in an extra case connected to the main system (shown in Figure 2). The home computer module had 18 Kbytes ROM inside the cartridge. Microsoft BASIC was used as a programming language for the home computer add-on and used up 8 Kbytes of this. 16 Kbyte RAM were also integrated in the module, of which 14 Kbyte could be used for user programs.

To save and load programs to external storage, a microphone and a headphone connector were included, which allowed the storage of data and programs utilising home audio equipment and standard audio-tapes.

Besides the manuals included with the cartridge, a book teaching how to program the system and including some example programs was released in France in 1984 (Bardon & de Merly, [1984](#)).



Figure 2. Philips Videopac+ C7420 Home Computer Cartridge: The cartridge plugs into the system at the front, connecting to the main case that holds the additional CPU and memory in the back. The connectors for loading/saving data to an audio system (red, white and black cables for microphone, headphones and remote control) are attached to the main case.

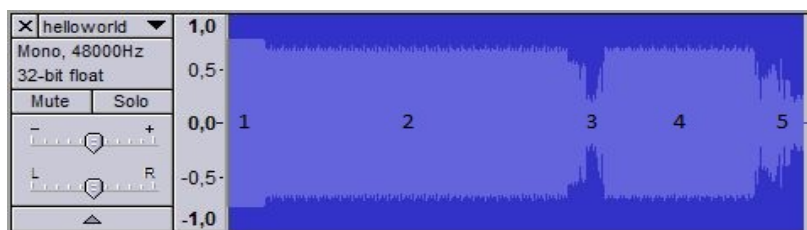


Figure 3. Waveform of “Hello World” BASIC Program (1: initial 6 kHz lead-in tone; 2: 256 x 0xFF as start of file-signature; 3: file header; 4: 128 x 0xFF as header/data separator; 5: data block).

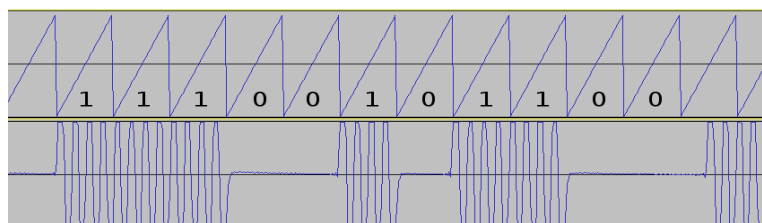


Figure 4. Representation of One Byte in the Waveform. One start bit (1), eight data bits (least significant first: 11010011b = D3h), 2.5 stop bits (0).

Re-Engineering the Waveform

Data on the system can be stored in various formats. The BASIC programming language variant that comes with the system supports saving program listings, screenshots, and storing and retrieving self-defined data (text strings and number arrays) using various forms of the “CSAVE” instruction.

In order to start re-engineering the storage encoding, the original machine’s output was connected to the input of a PC’s sound card. We started by writing some test programs on the original machine and recording the resulting audio files using Audacity⁵. One resulting waveform can be seen in Figure 3. By recording different test programs we were able to find out that there is always a 2.77 second lead-in frequency of a 6 kHz sine wave. The data block is stored in a 4.8 kHz sine wave encoding bit set (‘1’) as a tone and bit cleared (‘0’) as silence. Every byte is encoded as one start bit (tone), followed by eight data bits (storing least significant bits first) and 2.5 stop bits (silence), as in Figure 4. The data is stored at a rate of 1200 bits per second. Every file consists of the following data-bytes in the following order:

No. of Bytes	Code	Contained Information
256	0xFF	<start of file>-signature
32		file-header
128	0xFF	separate header / data
<variable size>		data-block
10	0x00	<end of file>-signature

Table 1. Data-bytes Contained in Every File.

During our online research we also found an active community⁶ that is still using and programming this system. One of its members, René van den Enden⁷, had written small programs that allowed BASIC programs to be transferred between the original system and a PC. On request, he provided us a copy of the source code of his programs, which confirmed part of our research regarding the format and provided more details we had not figured out at this point of our investigation.

Re-Engineering File Formats

To understand the logical format of the data stored in the waveforms it was necessary to find out what kind of data the C7420 can store. Using the original user manual it became apparent that the C7420 is able to store five different kinds of data with the following commands:

Object type	Command
BASIC Programs	CSAVE
Screenshots	CSAVES
Arrays	CSAVE*
Strings	CSAVEX
Memory Dumps	CSAVEM

Table 2. Commands Used to Store Different Types of Data.

By saving different kinds of test data we were able to first identify the format of the 32-byte file header, which is used for determining the format of the data block:

⁵ Audacity: <http://audacity.sourceforge.net/>.

⁶ Videopac / Odyssey2 Forum: <http://videopac.nl/forum/>.

⁷ René’s VIDEOPAC page: http://home.kpn.nl/~rene_g7400/.



- 10 bytes 0xD3;
- 1 byte determining the format of the file, usually the character after “CSAVE” (e.g., ‘S’ for screenshot, 0x20 (Space) for a BASIC program);
- 6 bytes for the program name;
- 1 byte 0x00;
- 5 bytes ASCII characters of the line number at which the execution of the program should start (for BASIC programs only);
- 3 bytes 0x00;
- 2 bytes start address in memory (Least Significant Byte (LSB) first);
- 2 bytes length of data block in bytes (excluding the first leading byte 0x00, LSB first);
- 2 bytes checksum: all data bytes added up to a 16-bit value (LSB first).

The data block (which is separated from the file header by 128 bytes 0xFF) starts with 0x00 and continues, depending on the specified format in the file header, with the following data:

Basic Program

For BASIC programs, the data block is split up into lines, which contain the following information:

- 2 bytes RAM address of the next BASIC line (LSB first);
- 2 bytes line number (LSB first);
- The actual line with the BASIC commands;
- 1 byte 0x00.

At the end of the BASIC program, a data block of 2 bytes (0x00) is written.

Every BASIC command is encoded as a byte code between 0x80 and 0xDF. The byte codes for all other characters in a BASIC command line (including white space) are stored exactly as they are input in the program.

Example BASIC line and encoding:

```
10 PRINT "HELLO"
```

Bytes	Representing
CF 88	0x88CF (address of next BASIC line in RAM)
0A 00	0x000A = 10 (line number)
94	PRINT (encoded command)
20 22	<SPACE><QUOTATION MARK>
48 45 4C 4C 4F	H E L L O
22	<QUOTATION MARK>
00	indicates end of line

Screenshot

The Philips G7400, using the C7420 Home Computer Module, can display images that are built together by using 8x10 pixel characters. 23 of the 24 40-column rows on the screen can be used for graphics. The uppermost row is used to display internal information such as cursor coordinates and cannot be accessed using the standard functions for loading and saving screenshots.

Users can change the representation (glyph) of the built-in graphics and text mode characters using the SETEG and SETET commands. Both of these commands have two parameters: the character code of the symbol to be replaced and a string consisting of twenty hexadecimal digits describing the appearance of the symbol. Each character uses an 8x10 pixel grid and is encoded as follows:

- Two hexadecimal digits (one byte) are used for each row of the grid, starting with the topmost one.
- The n -th bit of such a byte, starting with the lowest significant bit, corresponds to the n -th pixel of the row from the right.

A screenshot data block contains character and formatting data which can be used to fill 23 40-column rows on the screen, and thus is 1840 bytes long. Only the pointer to the used character and the formatting byte are stored in a screenshot file. User defined characters are lost if the program defining the characters is not stored together with the screenshot file.

The formatting of every two bytes of data for each screen position is described below, as well as how the formatting data for a byte influences the rendering of a character on the screen. An example image loaded in the migration tool can be seen in Figure 5.

Formatting

Each of the 40x23 characters is encoded using two bytes: one byte containing the character code, followed by a byte containing formatting data. A character is displayed either in text mode or graphics mode - this is stored as part of the formatting byte associated with it and determines the graphical representation (glyph) used, as well as the meaning of the remaining formatting data, as shown below:

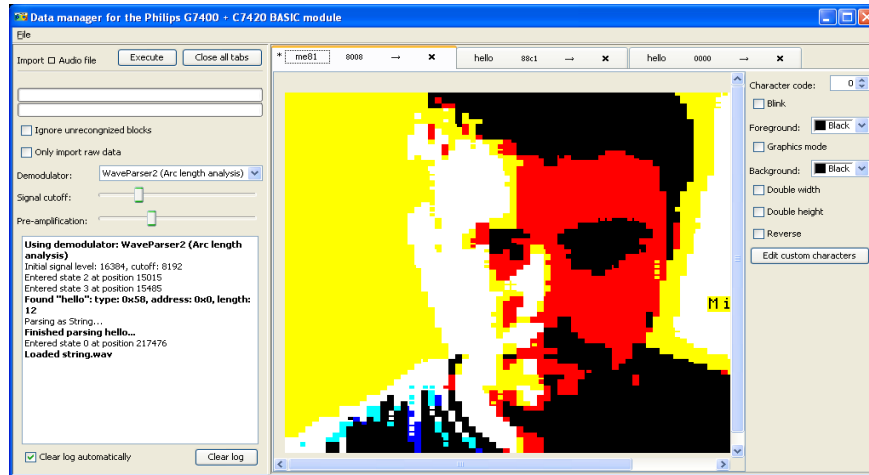


Figure 5. Migration Tool with Image Loaded from Wav File.

Format Byte Bit Usage in Text Mode:

```

7 6 5 4 3 2 1 0 (lsb)
| | | | | | | |
| | | | | +--+---- foreground colour
| | | | +----- !blink
| | | +----- double height
| | +----- double width
| +----- reverse
+----- 0 (controls graphics mode)
    
```

Format Byte Bit Usage in Graphics Mode:

```

7 6 5 4 3 2 1 0 (lsb)
| | | | | | | |
| | | | | +--+---- foreground colour
| | | | +----- !blink
| +--+----- background colour
+----- 1 (controls graphics mode)
    
```

Foreground and Background Colours

There are eight possible colours, each being a combination of red, green and blue. In the three bit representation used by the device, the first (least significant bit) determines the amount of red, the second bit determines the amount of green and the third bit determines the amount of blue. The resulting colours, ordered from 0 to 7 are: black, red, green, yellow, blue, magenta, cyan and white.

In graphics mode, each character has its own foreground and background colour as specified by the character’s formatting byte.

In text mode, each character has its own foreground colour encoded in the formatting byte. The background colour is “inherited” from the last character previously encountered on the row that was either a graphics mode character or a text

mode character with a code greater or equal to 128. In the second case, the background colour is taken from the same bits as the graphics character (bits 4, 5 and 6). This method is generally used for setting the background colour at the beginning of a row and can be seen in screenshots originating from the device - the first column contains characters with code 128 and a formatting byte with both hexadecimal digits set to the desired background colour for each line.

Double Width and Height

Text mode characters can be displayed in double width or height. Since glyph sizes are fixed, two consecutive grid cells/glyphs are required to fully display one double width or double height character, and a total of four cells/glyphs are required for a double width, double height character. Setting the double width / double height attribute for a single character results in only half of it being shown (or a quarter, if both attributes are set).

The same character code / format pair must yield two or four different glyphs, depending on which part of the character needs to be drawn. The rules used to determine which part to draw are as follows:

Double width: Each occurrence of a double width character after a single width character (or after a complete double width character) uses the first glyph (left part of the character). A double width character directly following the first glyph uses the second glyph.

Double height: Each line is assigned either top glyphs or bottom glyphs. A line containing double height characters that comes after a line containing no such characters uses top glyphs, consecutive lines are assigned bottom and top glyphs in an alternating fashion.

Blink and Reverse

The blink attribute makes a character blink on the screen - it is shown for one second, then hidden for one second, then shown again, and so on.

The reverse attribute reverses the background colour and foreground colour of a character. It also reverses the blinking phase for that character.

Array

The first byte of an array encodes the number of dimensions of the array. For each dimension, two subsequent bytes encode the number of fields in the dimension (LSB first). Finally, for every entry in the array, four bytes are used to express the value in different formats, depending on whether the array contains strings or numbers.

String Array

- 1 byte length of the string in bytes;
- 1 unused byte;
- 2 bytes address of the string in memory.

Note that the actual string data is not saved in the array but the strings have to be saved and loaded separately using the string save command “CSAVEX”.

Number Array

By saving number arrays on the original system, examining the resulting byte stream, changing values and re-loading the array onto the original system we were able to find out that a floating point format is used to store numbers. The encoding is similar to, but does not follow, the IEEE 754 floating point standard (IEEE, 1987), as that was released two years later than the C7420 cartridge. With further testing, the bits for mantissa, sign and exponent were determined. Four bytes are used to encode the number as a 32-bit floating point value (LSB first), with the following meaning of the bits:

bit 25-32	bit 24	bit 1-23
exponent (exponent bias = 129)	sign (1 = negative)	mantissa

Table 3. Meaning of the Bits in a 32-bit Floating Point Value.

So any number can be calculated using equation 1:

$$\text{number} = \text{sign} * \text{mantissa} * 2^{\text{exponent}} \tag{1}$$

where,

$$\begin{aligned} \text{sign} &= (-1)^{\langle \text{bit } 24 \rangle} \\ \text{mantissa} &= 1 + (\langle \text{bit } 1-23 \rangle / 2^{23}) \\ \text{exponent} &= \langle \text{bit } 25-32 \rangle - 129 \end{aligned}$$

String

Strings are stored as a stream of bytes using the ASCII encoding (number of bytes according to the file header information).

Memory Dump

Memory dumps are stored as byte values (number of bytes according to the file header information).

Converting Waveform to Bit-Stream

In order to write a tool that is able to convert the waveform into usable data, we had to develop a method of interpreting the waveform programmatically and detecting the various stages in the signal.

In our tests the signal was sampled as a 48 kHz, 16 bit, mono signal. As the C7420 outputs the signal at a rate of 1200 bits per second, we can calculate the number of samples per encoded bit (spb) using the following equation:

$$\text{spb} = \frac{f}{\text{bps}}$$

where,

- spb = samples per bit in the digitised audio stream;
- f = sample frequency of the waveform;
- bps = bits per second as output from the C7420.

The signal output by the C7420 is a sine wave with a frequency of 4.8 kHz, so every bit is represented by four sine periods.



We implemented two different methods of interpreting the signal. Method 1 was taken from the sample programs we got from René van den Enden. For each sample, we need to decide if it marks silence or signal. The algorithm scans the sample stream of the digitised waveform until an absolute value greater than half the maximum amplitude of the signal is found. High amplitude is interpreted as a signal, such as the start of a coded bit “1”. More samples are subsequently read and counted either as “signal” or “no signal”. If more than a certain amount of “no signal” samples are found, it is assumed that the end of a coded “1” has been reached and a coded “0” starts. For a coded “1” bit to be properly recognised, half the number of samples over the duration of four sine waves has to be interpreted as “signal”. Figure 6 shows a sample waveform and the values counted as “signal” (marked on the horizontal axis as “1”) and “no signal” (marked as “0”).

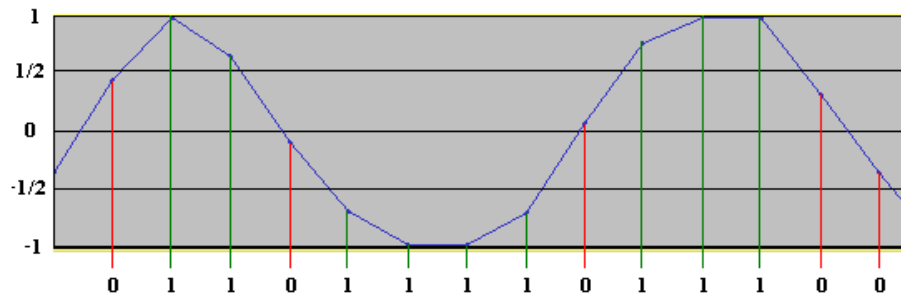


Figure 6. Interpretation of the Wave Signal Using Method 1. Vertical axis shows the strength of the amplitude; horizontal axes show the parts of the sine wave interpreted as “signal” (1) or “no signal” (0).

While we were able to read the original signal output by the console system without errors using this method, we encountered the following problems when we tried to interpret the signal stored on audio tapes:

- **Missing parts of a coded bit:** As a certain threshold of “no signal” was defined as the beginning of a coded “0”, errors were encountered while interpreting the signal if a small part of the bit had been lost due to data loss on the audio tape.
- **Noise in the signal:** Most parts of the tapes contained noise which was incorrectly interpreted as signal.
- **Differences in amplitude due to independent recordings on the same tape:** While we were able to adjust the level of the input signal using the software for recording the signal from the audio source, changes in the signal over various parts of one tape made parts of the tape unreadable.

To reduce the sensitivity of the algorithm that converts the waveform into a bit stream we implemented a second method. For Method 2 we not only looked at single samples in the waveform, but also calculated the sum of piecewise linear approximations of the amplitude, thus calculating the arc length of the sine wave for silence and signal first. Obviously, the arc length of a curve for a bit that represents “1” is longer than the arc length of a curve for a bit that represents “0”. To decide if a bit is set or cleared, a cut-off value between signal and silence wave arc length is used.

For every sample in the signal, a certain amount of samples before it are used to calculate the arc length of the sine wave up to the sample. If the arc length is above the cut-off value then the sample is recognised as “1”, otherwise it is recognised as “0”.

The algorithm is also able to adjust itself to changes in volume or noise, as the threshold which decides if a bit is set or cleared is constantly adjusted for every file in an input stream in parts of the signal which are known to be signal or silence. This way we are able to compensate for noise in the signal, as well as for changes in volume. Missing parts of a signal bit have less influence in the recognition, as not only the missing part, but also all parts before it are used to decide the state of the bit.

Migration Tool

Using the algorithms for converting the digitised waveform to a binary stream native to the system, together with the information we gathered about file formats, we developed a tool that is able to read the data contained in the waveform. Both described methods of interpreting the waveform were implemented.

The tool is written in JAVA. By using a virtual machine as a platform, the tool is independent from actual hardware for better long term stability. The tool and demo files can be found on the project homepage⁸.

The following functions were implemented in the migration tool:

- Opening an audio stream and loading the contained files, either from an audio file (WAV or FLAC) or directly from an audio-in device;
- Opening files in the C7420-native file format (binary streams converted from WAV-file);
- Saving the opened audio stream as a C7420-native file format (binary stream);
- Saving data in a non-obsolete format (screenshots as PNG, BASIC-programs and arrays as text files, binary data as binary);
- Saving data as an audio stream, either to an audio file (WAV or FLAC) or directly onto the standard audio-out device;
- Opening and saving compressed zip-archives containing a collection of migrated files;
- Creating new files of the different formats in the application, including syntax highlighting for BASIC-programs.

All the data formats used by the C7420 as described in the Re-engineering File Formats section above are supported by the migration tool.

Every file is opened in a new tab inside the application in an editor that is linked to the file type. The information associated with the file and stored in the file header (native file name, address in memory to load to) can be edited as well. A screenshot of the migration tool can be seen in Figure 7.

⁸ Migration Tool: http://www.ifs.tuwien.ac.at/dp/hc_audio_migration.

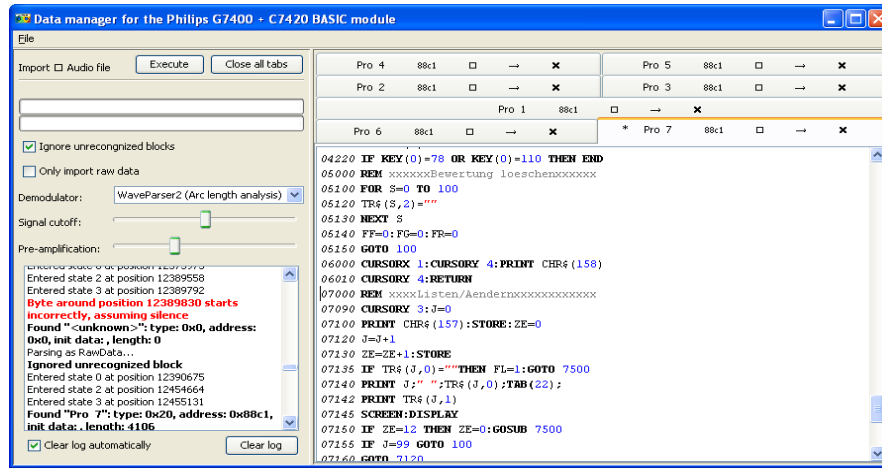


Figure 7. Screenshot of the Migration Tool GUI with Seven BASIC Programs imported from a WAV-File Recorded from an Original Tape. The import-log on the lower left shows events and errors during the import. Various import settings can be configured on the upper left and the imported programs are shown in tabs on the right.

Evaluation

To evaluate the usability of the migration tool, we recorded different programs and other data as output from the original system. The data was recorded as a waveform using Audacity and then converted to user readable data in the migration tool, using both implemented methods for converting the waveform. Then the data was loaded back into the original system, both from the recorded audio stream and from a stream re-encoded using the migration tool.

The migration tool was able to restore all the data in the waveform as output from the original machine with both methods of converting the signal. The original stream outputted by the machine and the re-encoded stream from the migration tool, both gave the same results when the data was loaded back to the original machine. For a clean signal that was not distorted due to age, the migration tool perfectly read and wrote the data from and to the original machine.



Figure 8. Tapes Used for Evaluation of Migration Tool. Upper left corner: C10 computer cassette; lower left and right Philips FE-I 60 normal position audio tapes.

Additionally, we acquired three audio tapes created with the original system approximately 20 years ago from a private archive. Two of the tapes were standard Philips FE*I 60 normal position audio tapes as used for recording music, while one was a C-10 computer cassette tape from manufacturer a11, specially manufactured for recording data (Figure 8). The source who recorded the tapes and the contents was not known before we started the experiments.

We used a standard HIFI-system as an audio player and the software Audacity to record the audio streams as 44 KHz, 16 bit mono digital signal. The audio streams were saved as uncompressed WAV-files (Petermichl, 2009) containing the pulse code modulated (PCM) (Cattermole, 1969) raw audio data as bit stream. Two of the tapes had data recorded on both sides of the tape; one had data only on side A. Five WAV-files were obtained, one per side and per tape.

Each file was then loaded using the migration tool. The resulting migrated files were stored in a zip archive. For comparison, the files were also loaded onto the original system.

A visual check for the characteristic waveform was done using Audacity to see how many files we expected the migration tool and the original system to find. A comparison between expected and loaded files can be found below (first column for each method shows recognised files, second shows unrecognised files and third shows false positives):

tape-side	visual	C7420		method 1		method 2		
C10-A	8	5	3	0	8	7	1	5
C10-B	2	1	1	0	2	2	0	0
Philips-1-A	6	0	6	0	6	6	0	3
Philips-2-A	6	0	6	0	6	6	0	2
Philips-2-B	1	0	1	0	1	1	0	0
Total	23	6	17	0	0	22	1	10

Table 4. Comparison Between Expected and Loaded Files.

Some files on the C10 tape were recognised by the original system, but could not be loaded due to a “Bad Label” error (with the suggestion to reposition the tape); while on the other two tapes no files were recognised at all. No files were correctly recognised using Method 1. All but one file were recognised by Method 2. Ten additional files recognised using Method 2 were false positives that were easily detectable in the user interface and recognition could even be suppressed by checking a checkbox in the migration tool.

The files that were recognised contained BASIC-programs. To check the files for validity, we loaded them onto the original system from the tape and also loaded them onto the original system as output from the migration tool.

From the 23 files on the three tapes no file was readable and usable on the C7420. All the six files that were recognised on the tapes were loaded with a “Bad File” error message and were not usable due to missing lines and incorrectly interpreted bytes. Thus, the original system could not be used to load the data from the original tapes.

The results of recognised data in the loaded files using the migration tool can be seen below:

tape-side	loaded	not recognised or wrong file format	with errors	no errors
C10-A	7	0	4	3
C10-B	2	0	2	0
Philips-1-A	6	1	5	0
Philips-2-A	6	1	5	0
Philips-2-B	1	1	0	0
Total	22	3	16	3

Table 5. Recognised Data in Loaded Files Using the Migration Tool.

Of the 22 files loaded, three files could be recognised without errors. 16 files were loaded with various warnings in the migration tool, indicating that some bytes could not be recognised or were misidentified (e.g., wrong checksum, missing bits in bytes). Three files were not recognised in the correct format and shown as binary stream only.

Without manual preprocessing of the waveform or manual post-processing of the binary stream, we were able to recover 19 files opposed to just six files loaded by the original system.

The files loaded with errors were in various states of completeness. Some files were missing various lines at the end of the file. Other program lines were erroneous due to incorrectly identified bytes (an example can be seen in Figure 9). As the original data stored on the tapes was not available for comparison, it is not possible to quantify the errors. But in general it seems that only single bytes were lost. As the data on the tapes consists of BASIC programs, it should be possible to correct the errors by re-engineering the recovered program sources and thus reconstruct most of the data on the tapes.

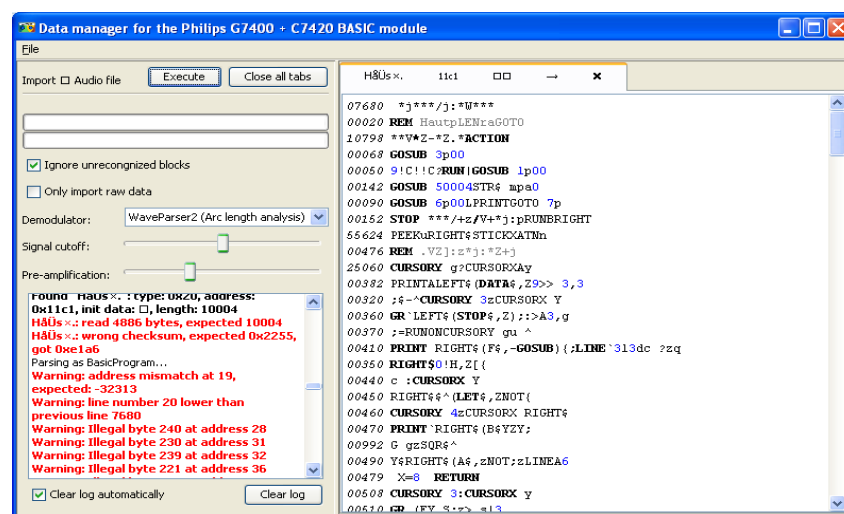


Figure 9. Screenshot of a BASIC-Program Imported with Errors from a WAV-File. In the program listing on the right side, incorrect arguments for commands and line numbers out of order can be found. The log on the left side shows error events that occurred during the import.

Conclusions

The case study performed in this work proved that it is possible to extract proprietary data from the analog audio signal stored by a system without previous knowledge of the format it is stored in. By having access to the original system to write test programs, we were able to re-engineer the audio waveform, as well as all data formats, and write a tool to migrate the data to non-obsolete formats. Archives or libraries that have or may receive audio tapes containing data for the Philips G7400 can use this tool to migrate the digital data without access to the original system or knowledge of how to handle the system.

This paper also shows that even a system with a very simple architecture, compared to today's technical standards, is rather complex. We not only had to research the physical and logical formats of storing the data, but also the interpretation of the data formats, and migration to non-obsolete formats had to be considered.

If the actions described in this paper are undertaken today, while the original systems still work, it is possible to develop tools for the migration of digital objects now. Once the original systems do not work anymore, it will not be possible to run code on the original system, thus having to re-engineer the system on a circuit-diagram level and disassembling the BIOS source code, which makes the task more difficult and time consuming.

Re-engineering of the System


While digital archaeology and re-engineering systems is seen as a rather complex task, this case study shows that the re-engineering of the format is easier while having access to the original system, as this way, test data can be produced. Interpreting the number format without seeing the effects of the changed numbers on the original machine would have been a rather difficult task. It should also be noted that non-commercial 'retro gaming' communities still working with the system can be an excellent source not only for emulation, but also for data archaeology on home computer systems.

Information Lost Due to Migration

It is not only the information stored in the files which have been migrated which has to be considered, but also how this information is rendered on the screen, e.g., for image formats. Thus it is necessary to characterise the potential objects that have to be migrated and look at their significant properties.

While most of the information that can be stored in files on the Philips Videopac+ G7400 can be migrated to non-obsolete formats, certain restrictions apply:

- **Screenshots:** The G7400 is able to render blinking information on screen. By choosing PNG as a non-obsolete (static) format, this dynamic information of the data is lost. Additionally, it is possible to define custom characters using the BASIC language. As these are not stored in the waveform with the screenshot data, a complete program with the definition of the custom characters would have to be stored and preserved to keep the information available. To correctly render the characters again on a new system either the program containing the character definitions has to be



analysed and incorporate that information as well when migrating to a new format or the program has to be executed in an emulated environment to recreate the original rendering.

- **String Arrays:** As a string array contains only the addresses of the strings stored in it and the strings themselves are each stored in separate files, the interrelation between these files is lost without the logic of the program that establishes the link between them.

If no adequate non-obsolete format is available to store the information necessary for the rendering process, alternative preservation approaches like emulation have to be considered as well.

Evaluated Tapes

Examination of the data on tapes from a private archive showed that the data was no longer readable on the original machine. Using the migration tool we were able to retrieve most of the data with small errors. The evaluation also showed that it is necessary to act now and migrate data that was stored on magnetic tapes 20 years ago, as the lifetime of magnetic tapes is expected to be a maximum of 20 years (Van Bogart, [1998](#)). Most of the data retrieved in the experiment could not be extracted without errors.

Improvement of Migration Results

As shown in the evaluation, not all of the programs stored on the tapes were read without errors. A corrupted byte does not just change one letter in the command, as every BASIC command is encoded in one byte, but results in a completely different command. Automatic correction of the files would thus be possible by checking the BASIC programs for certain rules, like commands, which allow or enforce a certain number or types of arguments and point out inconsistencies to an expert doing the migration. He or she can then correct the results manually. Possible automatic support could also be offered by showing commands with, for example, a one-bit difference in the encoded byte.

Media Refresh

Using the developed migration tool, it is possible to refresh the media (audio cassettes) by reading and decoding the content, recoding it into a waveform and recording it to the tape again without using the original system.

Using Decoded Data for Emulation

With the possibility to save the data encoded in the waveform as a system-native binary stream, files can be stored for usage in emulators. Currently no emulators for the C7420 are available, but by storing the streams in the native format the data is kept safe for emulation at a future date.

Interpreting Results for Other Media Types

As audio tapes can be read using standard non-proprietary audio equipment, access to the physical layer of data is not in immediate danger. Other magnetic media, like floppy discs, cannot be read as easily. Even with floppy drives using the same media size (8", 5¼", 3½") access to data written on non-compatible computer systems is not possible, as the physical parameters of the written data are not necessarily the same (e.g., number of tracks, block size, even recording technology).

The results of re-engineering the logical data can be used for other media as well. Re-engineering file formats can either be done using original systems or emulators, if available. Expert knowledge in handling the system has to be at hand to complete these tasks.

Acknowledgements

Part of this work was supported by the European Union in the 6th Framework Program, IST, through the PLANETS project, contract 033789.

References

- Baer, R.H. (2005). *Videogames: In the beginning*. Springfield, NJ: Rolenta Press.
- Bardon, C., & de Merly, B. (1984). *Jeux sur Philips C7420 Videopac+*. Paris: Edimicro.
- Bhushan, B. (1992). *Mechanics and reliability of flexible magnetic media*. New York: Springer.
- Cattermole, K.W. (1969). *Principles of pulse code modulation*. London: Iliffe Books.
- Forster, W. (2009). *The encyclopedia of game machines: Consoles, handhelds and home computers 1972-2009*. Utting, Germany: GAMEplan.
- Guttenbrunner, M., Becker, C., & Rauber, A. (2010). Keeping the game alive: Evaluating strategies for the preservation of console video games. *International Journal of Digital Curation*, 1, (5).
- Herman, L. (2001). *PHOENIX: The fall & rise of videogames - Third Edition*. Springfield, NJ: Rolenta Press.
- IEEE. (1987). IEEE Standard 754-1985 for Binary Floating Point Arithmetic (IEEE, 1985). Reprinted in *SIGPLAN* 22, (2).
- Matthews, B., McIlwrath, B., Giaretta, D., and Conway, E. (2008). *The significant properties of software: A study* (JISC Study). Retrieved December 11, 2009, from http://www.jisc.ac.uk/media/documents/programmes/preservation/spssoftware_report_redacted.pdf.



Petermichl, K. (2009). *Handbuch der audiotechnik: Kapitel 12: Dateiformate für audio*. Heidelberg, Germany: Springer Berlin.

Ross, S., & Gow, A. (1999). Digital archaeology: Rescuing neglected and damaged data resources. *A JISC/NPO study within the electronic libraries (eLib) programme on the preservation of electronic materials*. Retrieved December 11, 2009, from <http://eprints.erpanet.org/47/>.

Van Bogart, J. (1998). Storage media life expectancies. *Digital Archive Directions (DADs) Workshop 1998*. Retrieved December 11, 2009, from <http://nssdc.gsfc.nasa.gov/nost/isoas/dads/presentations/VanBogart/>.

Webb, C. (2005). *Guidelines for the preservation of the digital heritage*. Information Society Division: United Nations Educational, Scientific and Cultural Organization (UNESCO) – National Library of Australia. Retrieved December 11, 2009, from <http://unesdoc.unesco.org/images/0013/001300/130071e.pdf>.