Internal Use Only

TR-I-0137

The MapSignal remote speech editor

Alain de Cheveigne

(ATR自動翻訳電話研究所)
ATR Interpreting Telephony Labs.

1990 . 2

ATR 自動翻訳電話研究所
ATR Interpreting Telephony Research Laboratories

The MapSignal remote speech editor

Contents:

The MapSignal remote speech editor

MapSignal is a speech editing terminal program running on the Macintosh. With it, one can visualize and manipulate speech data that is resident on a remote host machine. Communication with the host is done via a serial line, but despite the low channel bandwidth MapSignal is fast. The protocol is based upon the host's normal shell, allowing easy interaction between user and host. Installation of MapSignal on a new host requires only the compilation of two simple commands. MapSignal is thus an interesting solution to the problem of portability.

1. Introduction

Speech is a complex and multiform object, difficult to discern in the raw numerical data that computers handle. A signal editor attempts to display data in a more meaningful way, allowing a researcher to manipulate speech, control processing, and visualize results. Signal editors are precious tools for speech science.

Unfortunately, they are usually highly machine-dependant, and difficult to port from one machine or system to another. This, and the fact that editors are rarely extensible, has resulted in a multitude of locally designed editors. Any speech engineer worth his salt has at some time or another built his own speech editor. This replication of efforts is wasteful.

The concept of user interface is evolving rapidly. A user interface can no longer be judged at the level of a single program. The integration of that program with other programs is a factor to consider. Scientists and engineers alike must produce reports and articles. The integration of text and graphic editors with research tools is an important goal.

MapSignal can be seen as:

• A signal editing tool in its own worth.
It allows basic editing operations in a very flexible and intuitive way, and offers a general mechanism to integrate new host-based programs within this interface. On a new host, MapSignal requires only the compilation of two simple commands. In this sense it is highly portable. MapSignal is integrated within the software environment of the Macintosh.

• A model for speech editor design.
We propose several new ideas that could be of use in any speech editor.

• A model for front-end and user-interface design.
The clear separation of host-resident and terminal-resident functions, the protocol, and the extension mechanism could be applied to other tools such as neural network editors or data base front-ends.

This report overviews first the issues of user-interface and portability, then outlines the solutions offered by MapSignal. The appendix contains the "MapSignal User Manual".

2. Portability

• The concept of speech workstation

Speech processing with a computer requires three elements, each of which comprises both hardware and software:
• signal input and output (AD-DA conversion),
• signal processing and storage,
• a graphic user interface.
Fast communication between these elements requires ideally that all three be within the same machine, connected by a high speed bus.

Thus has emerged the concept of speech workstation. A wide range of hardware and software has been proposed based on the integrated workstation concept (Kopec 1984, Eckel et al. 1987, Cong and Haton 1985).

While an integrated combination of the best available elements may seem optimal at any point in time, it is less appealing from an evolutionary point of view. The market for specialized machines is relatively small, and they tend to remain outside the mainstream of computer evolution. Software is necessarily more expensive and slow in evolution than for general-purpose machines, and certain tools may not be available for a given hardware configuration.

• Workstations and portability

Workstations tend to aggravate the portability problem. Graphical user interface libraries encourage the programmer to make use of them in every program, while integrated hardware fails to enforce modular program design. Where a program on a mini or mainframe had simple interfaces (stdin and stdout, with binary or ascii data), a similar program on a workstation might display results graphically, or invite the user to enter data with a mouse. Or it might be part of an integrated graphical software environment.

As is the case for hardware, the integration of software may seem optimal at any point in time, but it is less attractive from the point of view of evolution. Any change in hardware, system software or processing software requires the re-writing of parts of a large and complex program.

• Signal editors.

Portability problems are compounded in the case of signal editors, because they must deal with all three of the components mentioned above: signal IO, processing, and display.

• Graphical standards: the solution ?

As someone said, the nice thing about standards is there are so many them to choose among. Hardware technology and user-interface design evolve very quickly, and graphic and user interface standards have a rough time keeping up. Progress usually appears

outside the current standard, forcing an uncomfortable choice on the designer: go with performance and forsake the standard, or give up performance and stay with the standard.

The X-window standard is a case in point. The change from X-10 to X-11 forced a rewrite of all programs, so it is difficult to speak of a standard in this case. Yet no one denies the necessity of the improvements brought by X-11.

The X-windows library is low-level, and for practical purposes incomplete without a higher-level library (widgets). While the low-level library is covered by the standard, the high-level libraries are not: using a proprietary library forces a dependency on the maker that furnished it. An X-11 program written using proprietary widget libraries is, of course, not portable.

• MapSignal's solution to portability

MapSignal uses a separate machine (the Macintosh) for the user interface, thus challenging the assumption that signal IO and processing must be in the same machine as the user interface hardware.

The requirements made of the host or link are minimal:
- the host must offer a line-oriented interface, preferably its usual command interpreter or shell,
- this interface must be accessible via a serial line (soon: LAN),
- two simple server commands must be installed on the host.

MapSignal can work with a wide range of host hardware (PC, mini, main-frame, super-computer) and software (ordinary shells, Lisp, data-base or signal processing languages, etc.).

With separate machines for processing and user interface, both can evolve independently. Processing hard-ware that is old or specialized in a way that prevents it offering a graphic interface can be given a good user interface.

A further advantage is that the Macintosh offers a better quality interface than that offered by most workstations. It integrates good text and graphic editors, plotting programs, spread-sheets, etc., and allows high quality printing.

X-windows allows a similar separation between host and interface, but the constraints linking the two machines are stronger: they must both run the same version of X-windows, and be connected via an ethernet.

3. User interface design

There have been several attempts at user interface design here at ATR. This section is intended to give pointers to some classical information on user interfaces, and to outline a few principles.

• The best on user interfaces

The original Smalltalk manuals contain basic ideas, such as avoiding modes (Goldberg 1984, Goldberg and Robson 1989). Chapter 2 of Inside Macintosh vol. I (Apple 1985) summarizes many important notions, some of which are Macintosh-specific. A few more minor details are to be found in subsequent volumes (Apple 1986, 1988)). Another source of information is "The design of the user interface".

• A good interface is easy to use but difficult to program

The difficulty is partly technical: programs require many layers of software to provide the "look and feel" of a good interface. "Event-driven" programming takes some getting used to. Memory management is also an issue. The familiar types of memory allocation (global, stack and malloc), do not freely allow user interaction. As the user creates and discards objects, the program must allocate and free memory in a random order, and memory tends to be get fragmented. Smart memory management (garbage collecting, or Macintosh relocatable handles) avoids fragmentation, but makes programming tricky.

The difficulty is also partly human: the programmer must have a good intuition, a "feeling" for the interface, and at the same time be technically proficient and rigorous.

When programming an user interface, one sometimes feels a disproportion between the programming effort and the result, which often concerns a detail. This is the same disproportion as occurs in writing or preparing a talk.

The following are a few principles that the programmer should try to follow:

• Be responsive

Each action on the part of the user should provide an immediate response (often symbolic: a change in cursor shape, a blink, etc.).

• Be permissive

Any action that seems natural to the user should be allowed, if it is reasonable. Avoid situations where the user's wish meets failure. A different way of saying this is: if something *is* impossible, make it *seem* impossible, naturally.

• Be forgiving

Nothing makes a user more unhappy than mistakes, especially mistakes that cost a session of work. Avoid these situations: provide confirmation dialogs for dangerous actions (but don't over-do it). For simple actions offer an "Undo" mechanism.

• Be consistent within the program

Similar operations must *always* be done in the same way (sometimes the similarity is not immediately evident to the programmer). Most important, similar actions must *always* produce similar effects.

• **Be consistent with other programs**

Remember that your program is not the only one that the user uses. Standard actions (cut-and-paste, save, quit, etc.) and conventions (highlighting, blinks, etc..) must be the same across programs.

The recommendation is useless if other programs are not consistent among themselves. This is why the restrictive user interface definition of the Macintosh has an advantage over other freer toolboxes.

• **Be explicit and unambiguous.**

Each message (text or image) must be comprehensible by itself. Avoid abbreviations. Avoid ambiguity. For each message that you use, try to think "what other interpretation might this have ?".

This recommendation is in conflict with the necessity of concision. A balance must be found.

• **Advertise your features**

The first question a user asks is: "what can this program do ?". Find a way of letting him know at each step what are the options allowed. Menus are one convenient way, icons are another.

• **Be simple**

For each operation, try to imagine a more direct way of a) allowing the user to carry it out, b) implementing it. Difficult, because a major simplification can require a re-write of the program.

• **Avoid "featuritis"**

Featuritis is the disease of "too many features". Each new feature in your program is one more thing the user must learn. Try to *minimize* the commands that your program offers. This can imply *removing* features. Do so if there is another program that does the job as well or better, and if there is a convenient way of switching between it and your program.

• **Build orthogonal features**

Avoid offering different operations that do similar things. Try and make each command or operation as different as possible from all others.

• **Make the features combine well**

An example is "selection". This operation simply defines a portion of data. Combined with "cut", it removes the data, combined with "copy" it copies it to the clipboard,

combined with "paste" it replaces the data by the contents of the clipboard. Combined with "zoom" (in a graphics program) it zooms that portion of data.

• Make the feature set complete

If your program offers a feature in one context, and if that feature makes sense in another context, implement it there too. If you implement an "undo" command *somewhere*, implement it *everywhere*.

• Hide implementation details wherever you can

And don't hide them when you can't. You can hide the details of something that is 100% reliable, but if you depend on a mechanism that can fail (example: a serial link), it is better to let the user have full access to details.

• Test your interface

Ask a naive user to use your program. Watch him: the mistakes he makes are the mistakes *you* made. Ask the user for his opinion, desires and requirements. Do *not* implement them: re-think everything within the *entire* frame-work of the interface you are building.

• Take example on the Macintosh.

The Macintosh offers currently the best user interface. There is a wide range of programs (text, graphic, communication, spread-sheet, signal editors, etc.). Look there for ideas.

4. The design of MapSignal

This section tries to explain some of the design choices. Some of these ideas might be of use in other programs.

a) Architecture and protocol

• Two machines:

Avoids compounding the hardware dependencies of user interface, processing and I/O. Allows separate evolution of hardware. Allows use of old or specialized host hardware or software. Integrates user interface within personal computer software environment (more complete, better quality and cheaper than workstation).

• Simple protocol

The communication protocol was made as simple as imaginable. This simplicity has many advantages, and few evident drawbacks. A tentative conclusion is that communication protocols in general tend to be too complex.

• Synchronous master-slave communication

MapSignal (user interface) is master, the host is slave. MapSignal makes requests to the host, which the host serves, but the host cannot modify the state of MapSignal. MapSignal is under the sole control of the user.

This is in contrast with file transfer programs such as Kermit, or front-end programs such as the Petri-net editor AMI (Bernard et al. 1988) or the music mixing program MacMix (Freed 1986), where the state of the terminal can be modified by the host program. This is confusing when the modification is unexpected, or caused by errors or wrong settings, and it makes debugging difficult.

• The host is "stateless"

More precisely, the host always returns to the same state after each request (the state tree has only leaves). No information is kept by the host: MapSignal takes care of accumulating and organizing all the necessary information during a session. If the user aborts an operation (in case of error), MapSignal aborts the request. No information is lost beyond the parameters of that request.

• There is no "daemon"

Requests are received and serviced within the host's usual command interpreter. This is again in contrast with usual communication protocols that require a daemon to be running on the host.

A daemon is useful for accumulating state on the host. Since we avoid doing that, a daemon is not necessary.

There are several advantages in using a command interpreter:
- Many features are built-in: directory listing, file copying, etc.
- New features can be added by implementing a new command.
- The user is familiar with the command interpreter. He can diagnose the cause of communication problems. More importantly, he can interact with the command interpreter even while MapSignal is running.

• The protocol uses printing characters.

There are no 8 bit characters, no control characters. Only 6 bits out of 8 are actually used, which is inefficient, but the protocol is reliable and easy to debug.

b) User interface of MapSignal

• The user interface as an obstacle to research

A good user interface hides implementation details from the user. This becomes a problem when the *user* and *implementor* are the same person. This is the case when the user is a

researcher, and invents a new tool that did not exist when the user interface was built. This new tool should be integrated within the interface (if only for consistency), but the programming required is time-consuming, and should not be part of the researcher's role.

In this way, a user interface can actually become a hindrance. MapSignal tries to avoid this drawback in several ways.

• MapSignal avoids hiding things

All communication takes place within the host's command interpreter. The user is allowed to see the transactions and is encouraged to understand the (simple) protocol.

• MapSignal does the minimum.

MapSignal only offers capabilities that are not otherwise available (such as signal editing). It doesn't display host files as icons, or directories as windows.

• MapSignal offers a mechanism to integrate new tools.

The command editor and keyword interpreter are intended as a general mechanism to integrate new tools. It represents a compromise between "user-friendliness" and flexibility.

5. The Devil's advocate ("Throw away the workstations, buy Macs!")

This section presents the case for the use of the Macintosh as a signal processing user interface within ATR, instead of workstations. It ignores, for simplicity, the very cogent reasons why this might not be practical.

• The present state.

- Signal I/O is done with specialized hardware on VaxStations (also: MassComp).
- Processing is done on Vaxen, PMaxen, and the Alliant.
- User interaction is either via VaxStation windows, terminals or terminal emulators (PCs and Macs).
Symbolics Lisp machines are also in use for processing and interface.
- Manuscript preparation is done on the Xerox J-Stars.

• Drawbacks.

Manuscript preparation is not integrated within the user interface used for research. No easy way to cut and paste signals or graphs into documents.

X-11 is the standard window interface, but:
a) some machines still have X-10,
b) some programs only work with X-10,
c) X-window programs can no longer be run from the 8600 and 8800 because libraries haven't been updated,

d) X-window programs can't be run from the Alliant,
e) X-11 is slow, libraries and binaries are very large, and programs require much memory,

X-windows is a standard, but the interface toolboxes (for widgets, etc.) are not. Most programs will want to use these toolboxes and therefore will not be portable. This creates a dependency on the hardware maker, and also prevents programs developed at ATR from being distributed elsewhere.

• An alternative (Devil speaking):

Replace the workstations by Macintoshes. Connect them to the ethernet. Use telnet for multiple session terminal emulation, and an X-window server program to run existing X-window software. Use MapSignal for signal editing. Encourage Macintosh programming of user interfaces according to the guidelines suggested above.

• Advantages

Macintosh hardware is (relatively) cheap. Software is cheap, abundant, and of good quality. The user interface is excellent, and integrates text and graphic editors, as well as terminal emulators, spreadsheets, Mathematica, etc.. Printing is good. Mac software is fast and relatively small (in terms of disk space and memory requirements). It is also very stable.

• Disadvantages

Macintosh programming is difficult and unforgiving (a buggy program is dangerous to use, and can't be tolerated). The learning curve is steep.

Macintosh printer allows paper no larger than A4.

Conclusion

This report presented the remote signal editor "MapSignal", and discussed in detail some of the design decisions. MapSignal is not, in its current form, sufficiently complete or general for many purposes. It is hoped that it will prove to be nevertheless of some use, and that a few of the ideas presented here will find their way into the design of future, better software.

Acknowledgements

Part of this work was carried out while the author was supported by an STP fellowship awarded by the Commission of European Communities. The author wishes to thank both the Centre National de la Recherche Scientifique (CNRS, France) and Professor Culioli, Linguistics Department, Université Paris 7, for leave of absence, and ATR for its kind hospitality.

References:

Apple Inc. (1985). "Inside Macintosh", vol. I, (Addison-Wesley, New York), 550p.

Apple Inc. (1986). "Inside Macintosh", vol. IV, (Addison-Wesley, New York), 326p.

Apple Inc. (1988). "Inside Macintosh", vol. V, (Addison-Wesley, New York), 623p.

Apple Inc. (1987) " Apple Human Interface Guidelines: The Apple Desktop Interface", (Addison-Wesley, Reading, MA), 144p.

Bernard, J.M., Mounier, J.L., Beldiceanu, N., Addad, S. (1988). "AMI, An extensible Petri-net interactive workshop", 9th European workshop on applications and theory of Petri nets, Venice, June 1988.

Eckel, G., Rodet, X., and Potard, Y. (1987). "A Sun-mercury music workstation", Proc. Int. Computer Music Conference.

Freed, A, (1986). "MacMix 3.0: Mixing music with a mouse", Manual of MacMix software (IRCAM, Paris).

Goldberg, A. (1984). "Smalltalk 80 — The interactive programming environment", (Addison-Wesley, Reading, MA).

Goldbert, A, and Robson, D. (1989). "Smalltalk 80 — The language and its implementation", (Addison-Wesley, Reading, MA).

Johnson, D.H. (1984) "Signal processing software tools", IEEE ICASSP 84, 8/6/1-3.

Kopec, G., (1984). "The integrated signal processing system ISP", IEEE Trans. ASSP., 842-851.

de Cheveigné, A., Abe, M., Doshita, S. (1985). "The human interface of a speech workstation", Studia phonologica XIX, 18-26.

Schneiderman, B. (1987). "Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addison Wesley (Reading, MA), 488p.

Tesler, L. (1981) "The Smalltalk environment", Byte vol. 6-8, 90-147.

Zue, V. W., Cyphers, D.S., Kassel, R. H., Kaufman, D. H., Leung, H.C., Randolph, M., Seneff, S., Unverferth, J.E., III, Wilson, T. (1986). "The development of the MIT Lisp-machine based speech research workstation", IEEE ICASSP 86, 329-332.

Appendix: the MapSignal Manual.

MapSignal User Manual.

contents:

Alain de Cheveigne
January 14, 1990.

MapSignal User Manual.

I. What is MapSignal ?

MapSignal is an editor for speech or other signals. It works on the Macintosh as a terminal, connected to another machine (mini, mainframe or PC). The signals being edited reside on this other machine, the user interface is on the Mac.

MapSignal is a specialized program. It may be really useful for some people, but it's probably useless for most anybody else. Read on.

• What does MapSignal require ?

You must have a Mac and a host computer, connected by a serial line. The host may be a PC, or anything else that offers a command interpreter (MS-DOS, Unix shell, Lisp, RT-11, TSL, etc...) accessible by a serial line (this rules out the Mac itself).

The host is presumably equiped for signal processing, data storage and input/output. It presumably does *not* offer a nice user interface. If it does, you have no need for MapSignal.

You must install two commands on your host: "getsiz" and "getmap". These are simple commands (not server daemons), that execute from the command interpreter and return immediately. The sources are included with MapSignal.

MapSignal works on the Mac Plus and II and IIx. It has not been tested on other machines. It uses hierarchical menus, and so requires system 4.1 or later. It works fine under Multifinder.

• Who is MapSignal for?

The target user is a scientist or engineer doing research or development with speech or other signals. By definition, someone who creates new tools.

MapSignal avoids getting in his way by offering an interactive graphic interface for signal handling, but no more. MapSignal does not try to do anything you could do in a more traditional way. You can "see" and handle your signal files, but your directories won't look like icons.

• What can MapSignal do?

MapSignal is a communication program. You can hook up to a host computer via the modem port, and communicate at rates of 1200 to 19200 baud. Flow control is XOn/XOff (no DTR).

The terminal window has a buffer of 2000 lines or 20000 characters, whichever is smaller.

You can scroll, make selections, copy text, and paste from the clipboard to the host.

In addition, MapSignal allows you to display graphically on the Macintosh any signal file on the host, whatever its size. The data format can be short or long integer, float, or ascii (either all values in the file, or a specific column). You can zoom any part of the file to any scale.

Each new display has its own window, you can shuffle between the views. There is no limit (except memory) to the number of windows you can open, or to the number of files that can be mapped simultaneously.

At 9600 or 19200 baud, MapSignal is fast. The protocol is simple and robust. It operates within the host's usual command interpreter, and does not prevent normal (line-oriented) interaction.

MapSignal offers a menu of editable commands. When a command is selected, the corresponding string is sent to the host. The string can be any length (up to 32 k), and can contain any sort of character (including newlines).

When strings are sent to the host, they are scanned for "keywords" (words between backslashes, ex: \mouse\), that have a special meaning. When a keyword is encountered, it is interpreted and the *result* is sent to the host.

The keyword mechanism allows MapSignal to send important local information to the host. Examples are the name of the file currently displayed, or the place in the file where the user has clicked. This mechanism offers a powerful way to extend the basic functionality of MapSignal: new functions implemented on the host can be called from the menu, with parameters filled in at calling time.

MapSignal is *portable* (if we extend the meaning of the term to include carrying around a Macintosh). The two remote commands it depends on are simple and easy to install. MapSignal therefore offers a graphical user interface that works with a wide range of hosts.

• What it can't it do:

- You cannot edit signal files on the mac itself. There is no provision for file transfer.
- No spectrograms.
- MapSignal has no built-in commands to *modify* files (but you can easily add them with the editable command mechanism).
- No VT-100, no tektro emulation.
- No hardware handshaking: you may have trouble using a modem.
- MapSignal does not support communication via LocalTalk or Ethernet.
- No graphic editing, no printing. To print, you must do a screen dump and print that from a graphic editor.
- Graduations and units are very restricted. No markers. No color.
- No smooth scrolling (you need to open a new window).
- Provision for coordinating the mapping of two different files is limited.

Many of these limitations should disappear in future releases of MapSignal.

II. Using MapSignal.

The following supposes that: a) the serial connection has been established and debugged, b) the two commands (getsiz and getmap) that MapSignal needs have been installed on the host.

If this is not the case, refer to the section "Installing MapSignal".

Note: Practically all operations in MapSignal can be aborted by typing "CMD-." (press the "pretzel" key and type a period), "CMD-;" or "CMD-space".

1) A basic session.

• Start the session.

Start up MapSignal. The first window that appears is a terminal window. Use it to login to the host, and move to a directory containing signal files.

• Map a signal file.

Choose "Map New File" in the "File" menu. Type in the file name, and choose the data type appropriate for the data in the file. Press "OK". A long thin window appears, displaying the entire signal.

• Zoom it

Just double-click on the signal. A new window pops up, mapping a portion of signal centered on the place you clicked. You can keep on zooming, but first you might want to learn more about MapSignal. Read on.

2) Signal windows.

The signal data resides on the host. It is *never* sent to the Mac (that would take too long). What is sent is a *graphical description* of part of the signal, at an arbitrary scale and offset. That is why we speak of maps: a map is just a picture, not the real thing.

• The file window.

The long thin window that first appears when you map a file is referred to as the **file window.** It displays the entire signal, and serves as a sort of "thumb bar" to let you know where you are in the file each time you make zooms. The vertical scale of this window is automatically adjusted so the signal fits.

• Map windows

The easiest way to zoom is to just double-click somewhere. A **map window** pops up, showing the part of the signal centered on the place you clicked. The horizontal scale is multiplied by 4, and the vertical scale is adjusted so the signal fits the new window. You can keep on double-clicking to get more detail.

• Selection

Another way to zoom is to select part of the window. You just press the mouse and drag it accross the window, and release it when you're satisfied with the selection. Then choose "Map Selection" from the "File" menu.

If you aren't satisfied with the selection, and wish to adjust it, first press the shift key and then press the mouse. The corner nearest the click follows the mouse. While the shift key is down, you can also move from window to window.

The selection defines a zone *in the file*. It is not constrained to a particular window. The selection appears in inverse video in the front-most window, and as an outline in the others.

If you press the option key while selecting, the selection has the same height as the file window.

• Normalizing.

To adjust the vertical scale of the signal to fit the window, choose "Normalize" in the "Edit" menu.

• Mapping another file.

Proceed just as for the first file. Windows for several files can coexist without any problem.

• Synchronizing maps for two files.

Sometimes two files contain data that is related, so you would like operations on both files to be synchronized. MapSignal offers a solution to this problem. It is incomplete, but better than none:

To map a new file with the same scale and offset as an existing window, choose "Synchro-Map" in the "File" menu. Type in the new file name, and choose the appropriate alignment convention. "Same absolute values" means that the window will start at the same offset within the file, and will have the same horizontal scale. This is appropriate if both files contain data sampled at the same rate. "Same proportion" is usefull if the sampling rates are different.

A new window appears. The horizontal scale and offset are the same as in the model window, the vertical scale and offset are adjusted for a good fit.

3) The terminal window.

This a dumb terminal (no VT-100 or tektro emulation). It has a buffer of 2000 lines or 20000 characters (whichever is smaller). You can scroll, select, copy and paste (you can't select a word by double-click — yet). Text "pasted" to the terminal gets sent to the host.

Text typed at the keyboard is sent to the host *even when the terminal is not frontmost.* This is to allow typing while a signal is visible. This is helpful on small screens, but not user-interface-kosher, and indeed it is sometimes confusing.

4) The coordinates window.

The coordinates window displays the coordinates of the mouse point, whenever the mouse is over a signal window. The coordinates are that of the signal file (sample index and signal value).

Graduations in the signal windows themselves are rudimentary. File windows have no graduations, map windows have a horizontal axis at 0 and are graduated in samples.

5) Managing the windows.

Signal windows tend to accumulate and become invading. MapSignal incorprates several tricks to keep them at bay.

• Click anywhere in the terminal window: It will come to front and give you a refreshing view of a terminal screen. To send it behind everything else, click in its go-away box.

• The "Windows" menu contains items that refer to the terminal window, the coordinates window, and all the current file windows. If you choose a file window item, it, *and all of the map windows that map the same file* come to the front.

• You may have noticed that when a map window is frontmost, a *gray zone* appears in the file window. This gray zone indicates which part of the file is displayed in the map window. If you have several map windows for the same file, try pressing command key, option key and shift key together. The gray zones for all map windows appear. If you click on one such zone, the corresponding map window will come to the top.

• Perhaps the best way to get rid of windows is to close them:
- clicking on the "close" box of a *map window* closes that window.
- clicking on the "close" box of a *file window* closes that window, and all the map windows that depend on it.
- clicking on the "close" box of a *file window* while pressing the option key closes just the map windows. The file window itself remains open.

There is no hard limit to the number of windows that can be open at the same time, or to the number of files than can be mapped. You might however run out of memory. Also, things slow down if there are many windows.

6) The command editor

• The "Commands" menu.

The items below the dotted line (if any) in this menu correspond to strings that can be sent to the host. When an item is selected, the string associated with it is sent.

• The command editor.

The command editor allows you to edit the strings associated with command menu items. You can scroll, cut and paste. The command string can contain arbitrary text, including carriage returns, tabs, etc.. (tabs appear as spaces, control characters may not be visible at all). The size is limited to about 32 Kbytes (TextEdit limit).

Commands are kept as resources in the "Map... Commands" file. Changes are saved automatically.

7) The keyword interpreter

• Keywords.

All characers in a string associated with a command menu item are sent "as is", except when a "keyword" is encountered. A keyword is a string between backslashes (ex: \mouse\), with a special meaning. It is interpreted at the time the string is sent. Text before and after the keyword is sent "as is".

The result of the interpretation may send some information to the host (for example, the place the user last clicked), or perform some other action (ex: open a window).

• Comments.

These are the simplest form of keyword: interpretation has no effect, and nothing is sent to the host. Comments are anything between * and *\ (vaguely reminiscent of "C" language comments...). Example:

```
\* this is a comment *\
```

Comments can span several lines, but you cannot nest them.

• Some things you can do with keywords

\fName\
inserts the current file name into the string sent to the host,

\mouse\
changes the cursor shape, and waits for the user to click in a signal window,

\time\
sends the time,

\escape=+\
changes the escape character from '\' to something '+', until the end of the string.

See the appendix for a complete list of keywords.

• Example of a command using keywords.

There is no built-in command in MapSignal for modifying files. However, it is easy to implement a tool on the host, and call it from the command menu. The MapSignal distribution contains the source of a program called "xcat", that extracts a portion of a signal file and puts it to standard output. The usage of xcat is:

```
xcat filename start_index samples >stdout
```

Here's a command string to call it:

```
\* this command extracts a portion of the current file,
  defined by the current selection *\
 xcat \fName\\sBegin\\sSamples\>\askName\
```

Everything that is not a keyword is sent to the host ("xcat", the spaces, the ">" character, the carriage return). The keywords are interpreted as follows:

\fName\: name of the file displayed in the topmost window,
\sBegin\: start of the selection in that file ,
\sSamples\: number of samples in the selection,
\askName\: puts up a dialog to ask user for the name of a file to put the result.

• Pop-up keyword menu.

The appendix contains a list of keywords that you can refer to. There is also a pop-up menu in the command editor window, listing all the keywords. If you choose an item, it is automatically inserted.

If you press the shift key while choosing an item, a help window pops up with a short explanation of the item.

8) The **flow** window

The **flow** window allows you to implement a feature that no other editor I know of can offer: real time monitoring of AD input.

The **flow** window listens to the serial line, and interprets all incoming data as graphical min-max codes (see Protocol). As each pair comes in, a vertical line is drawn, and the window shifts leftward 1 pixel.

The **flow** window is meant to represent, in real time, the contents of a circular data buffer on the host. In this AD conversion scheme, conversion proceeds continuously as new data constantly overwriting old data in the circular buffer (as it were, data "flows" in at one end of the buffer, and out the other end). When the user stops the conversion, data is transfered to a file.

The **flow** window shows at each instant the envelope of the signal remaining in the buffer. This allows the user to check a) the data range (avoid saturation) and b) the timing (avoid truncation).

The **flow** window is invoked via the \flowWindow\ keyword (see Appendix). Example:

```
rt_adin -f 12 -o buffer -s 350\flowWindow\
```

The rt_adin command is the host command that handles AD conversion and data storage, and calculates the min-max codes, sending them to stdout in real time. Unfortunately, this command is extremely device dependant. The source is not included with MapSignal.

III Installing Mapsignal

• Copyright

MapSignal, getmap and getsiz are copyright Alain de Cheveigné, Centre National de la Recherche Scientifique (CNRS). In its present version (1.0b1), MapSignal can be used, modified, copied and distributed freely for research purposes.

• Installing MapSignal on the Mac.

MapSignal comes on a diskette without a sysem. Copy it to a disk with a system (version 4.1 or later). Copy the files "Map... Settings" and "Map... Commands" to the same directory. If these files are not present, they are created.

MapSignal saves changes to commands and settings automatically. Make sure the disk is not locked, and contains enough space.

MapSignal requires at least 250 Kbytes of memory to work. A bit more is better...

• Hooking it up

You need a serial line with a mini-DIN plug. Connect the mini-DIN to the modem port on the Mac. The other end must be connected to the host via a null-modem (adaptor that swaps the TD and RD lines).

MapSignal uses software handshake (XOn/XOff), so the cable only needs 4 wires: ground, signal ground, TD and RD. On a DB-25, these are pins 1,7,2 and 3. On a mini-DIN they are shield, 4, 5 and 3.

To keep the host happy you may have to make a loop-back on its connector: on DB-25 try connecting 4 with 5, and 6 with 8 and 20.

• Setting the serial parameters

If you don't know the right parameters, try various combinations. Be patient: serial links are notoriously difficult to debug. It helps if you have another terminal program you are familiar with: note the settings that work with it and use them in MapSignal. The public domain program Kermit is included on the diskette: try it.

• Installing the remote commands.

MapSignal requires two commands to be installed on the host (getsiz and getmap). The sources are provided with the MapSignal distribution, along with sample data.

First transfer the sources to the host. Use a file transfer program, for example the Kermit provided with MapSignal. Another solution might be to open an editor on the host (under unix: cat>file), and then paste the sources from a Mac text editor to the MapSignal terminal.

To transfer the binary signal examples (short, long and float formats) you must use a file transfer program.

The sources as provided work under UNIX 4.3BSD, and should compile with no problem under any other flavor. For other machines, systems or languages, you may have to modify or rewrite the sources (they are quite simple).

The remote commands are meant to work within the context of a "command interpreter". This can mean an ordinary shell. It could also be a Lisp interpreter, a data base language, a signal processing language, or any program that accepts to service the calls.

IV Using MapSignal with another terminal program.

Currently, MapSignal does not offer any VT-100 or tektro emulation. It is possible, under some conditions, to run MapSignal together with other communcation programs under MultiFinder.

Warning: Running several terminal programs at the same time on the Macintosh is not altogether safe. Please open and close the programs in the order recommended here.
Do not attempt to use ResEdit if you are not knowledgeable. Make modifications on a **copy** of the program, and re-name the copy.

1) Order of opening and closing.

a) open the other terminal program,
b) open MapSignal,
c) close MapSignal,
d) close the other terminal program.

2) What to do if the other terminal program "steals" bytes from the background

Most terminal programs listen actively to the serial port, even when they are in the background. Such a program will steal bytes from MapSignal and prevent it working correctly.

You can prevent this behavior by modifying some of the flags in the other program's SIZE resource, using ResEdit.

a) Don't do this if you are not sure what you are doing.

b) Make a copy of the program you want to modify. Give it another name (for example "Versaterm_I_steal_no_bytes") to prevent confusion with the original.

c) Open the copy with ResEdit and set the following flags to 0 in each SIZE resource: "Accept suspend events", "Can Background" and "Juggler aware".

If the program allowed background services such as file transfer, these will no longer work.

This procedure should become unnecessary in the future when MapSignal uses the new Communication Tool Box system software.

V Protocol

The communication protocol is simple and robust, and easy to debug in the event of a problem.

1) Principles

• Communication is synchronous.

• The Mac is master, the host is slave. The host cannot make requests or otherwise alter the state of MapSignal. MapSignal is under the sole control of the user.

• The host is "stateless": after serving a request, the host always returns to the same state.

• Requests are served within the host's command interpreter. There is no "daemon".

• Commands, parameters and data use printable ascii characters: no 8th bit, no control characters. Nothing is hidden. If something fails, the cause is immediately visible in the terminal window.

2) Order of transactions

• The user requests a map of a file, specifying the file name.

• MapSignal sends the **getsiz** command with the name as parameter.

• The host executes the command, answering the file size in samples, together with the min and max of the signal values, and returns to the shell.

• MapSignal calculates scale factors to map the signal file to the window, and uses them as parameters to the **getmap** command.

• The host executes the command, answering a coded graphical description of the file, and returns to the shell.

• MapSignal interprets the graphical description and draws the envelope of the signal in the window.

For subsequent requests (zooms), only the **getmap** command is used.
Outside of these transaction sequences, the user is free to interact with the host's interface via the terminal window.

3) Command format

There are two commands. Here UNIX shell syntax is assumed (Lisp, and TSL are also available).

• getsiz

Usage is:

```
getsiz -v
```

answers the version (not used by MapSignal),

```
getsiz filename format
```

answers the file size in samples, and the max and min.

Parameter <format> indicates the data type of the signal file. It can be any of:

"-s": short integer,
"-l": long integer,
"-f": floating point,
"-a": ascii.

In the case of ascii format, the parameter can be followed by a number that indicates the *column* that contains the data (ex: "-a12" is column 12). No number indicates column 1, a zero indicates that *all values* are to be used as data, regardless of column organization.

• getmap

Usage is:

```
getmap -v
```

answers the version (not used by MapSignal),

```
getmap filename block index hscale size vfactor voffset format
```

answers a graphical description.

Parameters <block> and <index> define the index in the file of the start of the zone to map (= 1024*block + index). This strange convention exists for historical reasons.
Parameter <hscale> indicates the horizontal scale, or number of samples per pixel. It can be a float. (in the case of a non-integer scale, an algorithm alternates between two integer values). It must not be less that 1 (MapSignal handles scales smaller than 1 in a special way, using for that purpose a call to getmap with <hscale> = 1).
Parameter <size> (integer) is the size in pixels of the window to draw, <vfactor> and <voffset> (float) define the vertical scaling (in order: add then multiply).
Parameters <filename> and <format> are as for getsiz.

4) Data format

The start of data is signaled by the string "_ok_" (4 chars).
The start of an error is signaled by the string "_nk_".

• file size data

Getmap answers the file size in samples (integer) followed by the min and max of the entire signal (float). For example:

```
sp-alain% getsiz speech.a -s
_ok_ 9736 2608.000000 14649.000000
```

• graphical description data

The getmap command calculates the min and max of groups of samples. These min and max are scaled and clipped to -2048 and +2047 . Then they are shifted to a 0 - 4095 range, and coded in 4 bytes as follows:

```
c1 = min/64 + 32
c2 = min%64 + 32
c3 = max/64 + 32
c4 = max%64 + 32
```

All bytes are printable ascii characters between 32 (space) and 95 (_). Any character outside this range (ex: newline) is ignored, and can be used as a separator.

Of the -2048 to +2047 range allowed by the code, only the range -512 to +511 maps to the MapSignal window. Values outside this range are clipped by the drawing routines.

Each min-max pairs is used by MapSignal to draw a vertical line in the window. MapSignal uses a joining algorithm to ensure a continuous display even when successive min-max intervals do not overlap.

When the horizontal scale (pixels per sample) is fractionary, a dithering algorithm alternates calculation between the integer scales immediately above and below. When it is less than 1, getmap is called with an <hscale> parameter of 1, and lines are drawn between successive samples instead of from min to max.

The **flow** window (called from the command interpreter using the \flowWindow\ keyword) uses the same coding scheme.

VI Future enhancements

It is a design principle that MapSignal should do as little as possible. Anything that can be done in a more traditional way (such as listing directories) should be done that way.

Enhancements I'd like to make are:

- Printing,
- Cut and paste of waveforms in graphical and numerical format,
- Grow boxes on windows,
- Some form of scroll (by quarter or half window),
- Graduations, and markers,
- Various minor user interface improvements,
- Incorporate for the Communication Tool Box software (allows connections other than serial and offers VT-100 emulation),
- Local waveform editing (result sent to the host via a keyword),
- Spectrogram windows (probably implemented via keywords),
- Synchronized mapping of several windows,
- Staggered plots,
- File transfer,
- Modification of the command sring mechanism to allow redirection (command string -> local file, host -> local file, local file -> host).

These enhancements should not modify the communication protocol. As far as possible, new features will be added using the keyword mechanism.

For example, a sonagram mechanism would call a specialized FFT command on the host via an editable command string, with parameters filled in by keywords. A specialized window, invoked by a keyword, would listen to the serial port and interpret the incoming coded graphical data to display a sonagram.

The "flow" window, invoked by the \flow\ keyword, is currently used in this way.

Acknowledgements

The author is with the Centre National de la Recherche Scientifique (CNRS). The work was carried out mainly at the Laboratoire de Linguistique Formelle (LLF) at Paris 7 University, and in part at the ATR Interpreting Telephony Research Labs (Japan), while the author was supported by the Scientific Training Program (STP) of the European Communities. Thanks to all those institutions, and to the people behind the names.

Appendix

List of keywords

Keywords are used within command strings. When a string is sent to the host, any keywords encountered are interpreted. Below is a list of keywords, along with their meanings.

• mouse data

\mouse\
\vMouse\
\hMouse\
Change cursor shape to invite user to click in a signal window.
Wait until he does. The difference between the 3 keywords is the cursor shape.

\hClick\
Send the index of the sample that the user last clicked on.
You should precede \hClick\ with a call to \mouse\ or \hMouse\.

\vClick\
Send the value (in remote data file coordinates) that maps to the point that the user last clicked on.
You should precede \ vClick \ with a call to \mouse\ or \ vMouse \.

• file data

\fName\
Send the name of the current file (topmost signal window).

\fSamples\
Send the size of the current file (topmost signal window).

\fMax\
Send the maximum value in current file (topmost signal window).

\fMin\
Send the minimum value in current file (topmost signal window).

• window data

\wBegin\
Send the index of the sample that maps to the left of the current window.

\wEnd\
Send the index of the sample that maps to the right of the current window.

\wSamples\
Send the width in samples of the zone that maps to the current window.

\wTopVal\
Send the value (in remote data file coordinates) that maps to the top of current window.

\wBottomVal\
Send the value (in remote data file coordinates) that maps to the bottom of current window.

\wRange\
Send the span of values (top to bottom) mapped by current window.

\wMax\
Send the maximum value of data in current window.

\wMin\
Send the minimum value of data in current window.

• Selection data

\sBegin\
Send the index of the sample that maps to the left of the current selection.

\sEnd\
Send the index of the sample that maps to the right of the current selection.

\sSamples\
Send the width in samples of the zone that maps to the current selection.

\sTopVal\
Send the value (in remote data file coordinates) that maps to the top of current selection.

\sBottomVal\
Send the value (in remote data file coordinates) that maps to the bottom of current selection.

\sRange\
Send the span of values (top to bottom) mapped by current selection.

\sMax\
Send the maximum value of data in current selection.

\sMin\
Send the minimum value of data in current selection.

•Mapping parameters

\hScale\
Send the horizontal scale (samples per pixel) of current window.

\vFactor\
Send the vertical scale factor of current window.
This is the ratio of: a) the nominal range of -512 to +511 to: b) the range of data values actually mapped to the window.

\vOffset\
Send the vertical offset of current window.
This is the amount added to the data (before scaling) to map it to the current window.

• units (some of these are as yet meaningless).

\samplePeriod\
Send the value of sample period given for current file.

\vUnits\
Send the vertical units string given for current file.

\hUnits\
Send the horizontal units string given for current file.

• control characters

Any control character can be sent by enclosing the corresponding (non-control) character in backslashes. Example:

\C\
Send the ctrl-C control character.

• escape

\\
Send a backslash.

\escape=<new escape>\
Change the escape character to the character <new escape> (type the new character flush with the = sign)."
The change takes effect immediately and remains in effect until the end of the command.

• commands

\command <command name>\
Execute another user defined command.

• time

\ticks\
Insert time in ticks (1/60 second) since Macintosh startup.

\secs\
Insert time in seconds since Macintosh startup.

\date\
Insert date in year-month-day format.

\fDate\
Insert date in French day-month-year format. Cocorico!

\time\
Insert time.

• special

\askName\
Ask user for a string (for example a file name) and send it to the host.

\flowWindow\
Special window for real-time signal input monitoring. \flowWindow\ does:
1) send a return character to the host (to terminate current command),
2) read off the echo, then
3) put up a FLOW window.
A FLOW window is a special window that captures graphical data from the host and draws it while scrolling right to left, a pixel at a time. This window is intended to display in real time the contents of a circular signal buffer during AD-conversion on the host.