



Soundtrack 2

User Manual

(Version 2.0)

Alistair D N Edwards
Department of Computer Science
University of York
Heslington
York
Great Britain
YO1 5DD

Janet: ALISTAIR@UK.AC.YORK.MINSTER

Welcome to Soundtrack 2

Soundtrack has been developed and distributed as part of a research programme into the adaptation of human-computer interfaces for use by blind people. The copyright is owned by the author, but permission is granted for it to be freely copied and distributed, subject to the following conditions:

- it must be used only for non-profit-making purposes (this would include blind people using of Soundtrack as a word processor and anyone using it as part of further research);
- any copies distributed must include *all* the contents of the master disc - including documentation;
- everyone who uses Soundtrack is urged to fill in and return the questionnaire, as a contribution to the continuing research (See Appendix A).

Anyone wishing to use or copy Soundtrack for any other purposes must have the written permission of the author.

Soundtrack has been well tested and is free of any major bugs but any bugs which are discovered should be reported via the questionnaire and every effort will be made to correct them. However, Soundtrack is used at your own risk and no responsibility can be accepted for any losses caused directly or indirectly by its use.¹

¹ Sorry about that. Those disclaimers really annoy me in commercial software; who would ever sell a car with a disclaimer to protect the manufacturer against claims if it were to blow up spontaneously? But seriously, folks, you are getting this software for free.

Contents

1. Getting started
 2. Background
 3. Principles
 4. Sounds
 5. Dialogues
 6. Menus
 - 6.1 The File Menu
 - 6.2 Edit menu
 - 6.3 The Sound menu
 - 6.4 The Format menu
 7. The Document windows
 8. Errors and alerts
 9. Using Soundtrack with other adaptations
 10. Limitations
 - 10.1 Soundtrack is not an operating system
 - 10.2 Features which were too difficult to implement
 - 10.3 Soundtrack does not make use of folders.
 - 10.4 The Macintosh is just too visually orientated
- Appendix: User Questionnaire

Table of figures

- 3.1. The layout of Soundtrack auditory screen.
- 5.1. Check Boxes.
- 5.2. Radio Buttons.
- 6.1. The File Menu.
- 6.2. The **Select file** dialogue.
- 6.3. The **Save with what file name** dialogue.
- 6.4. The dialogue asking whether the user wishes to save a new document, which has not been named.
- 6.5. The **Select file** (for deletion) dialogue.
- 6.6. Your absolute last chance to change your mind about deleting a file.
- 6.7. The **Edit** menu.
- 6.8. The **Find string** dialogue.
- 6.9. The **Sound** menu.
- 6.10. The **Speech** dialogue.
- 6.11. The **Set volume** dialogue.
- 6.12. The **Select sound** dialogue.
- 6.13. The **Menu options** dialogue.
- 6.14. The **Mouse options** dialogue.
- 6.15. The **Format** menu.
- 6.16. The **Select font** dialogue.
- 6.17. The **Select size** dialogue.
- 7.1. The **Document** window.
- 7.2 Stepping back.
- 7.3 Jump controls.
- 7.4 Extending the selection with the *Shift* key.

Soundtrack 2

1. Getting started

To use Soundtrack it is firstly necessary to obtain a copy of Apple's *Macintalk* speech synthesizer software. This is obtainable through your Apple dealer. The Macintalk icon must be copied into the *System folder* of a system disc (hard disc or floppy). The *Soundtrack* file must be copied from the Soundtrack Master disc to that system disc. (Note that it is a feature of Macintalk that Soundtrack must be on the same disc as the System folder). Ideally you should have Soundtrack set as the Startup program so that if you re-boot your Macintosh and insert the Soundtrack disc, Soundtrack will automatically begin execution. That will mean that a blind user will be able to start Soundtrack without any assistance, but it will be necessary for a sighted person to make Soundtrack the startup program, since this has to be done through the (visual) Finder.

Running Soundtrack

Assuming you have Soundtrack on a floppy disc and you have a disc made up as described above, then you switch on the Macintosh and insert the disc. You will hear the disc whirring, and eventually Soundtrack will start up and – in synthetic speech – welcome you and make a long speech about itself. It will then emit a beep, and will be ready for use.

Compatibility

Unfortunately the Macintalk speech synthesizer software appears to not be compatible with the Macintosh SE/30 or the Macintosh II. As Macintalk is an unsupported Apple product, there seems little hope of this deficiency being corrected. A version of Soundtrack using a different speech synthesizer – which will run on those machines – may be produced if there is any demand.

Soundtrack 2 does run in Multifinder, but it is not possible to switch to other applications: the Apple menu is not available and the menu bar icon is not accessible (the menu bar is treated as being off Soundtrack's auditory screen). It seems unnecessary to make Soundtrack more compatible with Multifinder since a blind user of Soundtrack will not have access to any other Macintosh software to run in parallel in Multifinder

2. Background

A major aim in the development of human-computer interfaces has been to make computers easier to use. Although developments have led to different styles of interaction, one aspect has remained constant: output from computers has always been almost entirely visual. Programs have been made easier to use with the advent of displays which are very

complex visually. Any form of visual interface is obviously difficult for a person with a visual impairment to use. Nevertheless, simple visual interfaces *have* been adapted so that they can be used by people who are visually disabled, even to the point of being blind. Such adaptations become more difficult as interfaces become more complex that as interfaces have evolved which do make computers easier to use for sighted people, they have become less accessible to users who are disabled visually.

It seems certain that the trend will continue towards more visual interfaces. For instance, Zachmann (1987) states that "The future lies with a graphical windowing interface, mouse cursor control, pull-down menus, dialog [sic] boxes, and the like." and that computers based on such interfaces "are destined to take over the IBM PC and compatible world, as well.". Information technology has been exploited to improve the lives of people with disabilities. Scadden (1984) points out that many visually disabled people have been empowered by the use of information technology such that they have attained a degree of equality in job opportunities in areas involving the use of that technology, such as word processing. At the same time Scadden warns about the effect of the increasing complexity of human-computer interfaces. "If such visually-oriented computer equipment were to become dominant in the microcomputer marketplace, blind and visually impaired people would lose much of the newly acquired equality, unless...alternative approaches were available." Soundtrack was developed as part of a project attempting to provide such an alternative approach. Zachmann's premiss is accepted and it is assumed that if visually disabled people are going to continue to be able to use information technology then ways must be found of making highly visual interfaces accessible to them.

There are a variety of approaches which might be made to solving the problem. In this project a set of design principles was devised which provided the framework for one possible solution, based on the *adaptation* of the user interface. This was by no means the only potentially viable approach, it was just one possibility. Working within that framework it was possible to implement a particular piece of software, a word processor. The implementation involved making further specific decisions, but it meant that it was possible to test the design principles in a practical manner.

Recently computer interfaces have become even more visual, with the development of the so-called window, icon, menu and pointer - or *wimp*² - style of interface. In addition to

² The word *wimp* is an acronym - a word formed from the initial letters of other words. Where such words can sensibly be pronounced phonetically they are usually treated like any other word - except that they are conventionally written in capital letters. There seems no logical reason why a word should SHOUT at the reader in this way because of its etymology. Therefore, in this thesis *wimp*

the keyboard which was previously used as the means of communicating to the computer, wimp systems have a pointing device which can be used to point at, and interact with, visual images on the computer screen. Windows, icons and menus are categories of the images used in such interfaces.

In using such systems, visual properties of the screen contents become significant in communicating to the user. Such properties include: shape, colour, size and position (sometimes in three dimensions). Printed text is still used extensively also. Remembering that it is also necessary for the user to be able to point at these images, it should be clear that a visually disabled person is seriously at a disadvantage in using such systems.

Soundtrack has been developed as part of a research programme to investigate whether wimp-style interfaces can be adapted for use by visually disabled people, and if so how best this can be done. A prototype version of *Soundtrack* was developed as the basis of research towards a PhD, and full details of its development can be found in the resultant thesis: Edwards (1987). That version contained a number of shortcomings, some of which were due to its being a prototype and some which were identified when it was evaluated. *Soundtrack 2* is intended as a production version which can be used in real applications.

Soundtrack was primarily intended as a vehicle for investigating auditory interfaces for blind users, not necessarily as a better word processor. *Soundtrack 2* is being distributed as part of the continuing research. It is hoped that people will use it in real word processing jobs and achieve a level of competence in its use. They will contribute to the research by supplying feedback on its use.

This user manual is intended to provide all the technical and practical information which will be needed by users of *Soundtrack 2* and by researchers interested in the broader implications of the *Soundtrack* project.

3. Principles

The design of *Soundtrack* is based upon two main principles. Firstly, items which users

and any other pronounceable acronyms are written using the normal rules for capitalization, while acronyms which are pronounced as initials are written in uppercase letters (e.g. HMI - human-machine interface).

of visual programs *see* are replaced with items which can be *heard*. Secondly the resultant auditory interface is constrained to make the interaction more simple and manageable. Two forms of sounds are used in Soundtrack: synthetic speech and musical tones.

The design of Soundtrack attempts to provide interactions which are as closely analogous as possible to those in a visual wimp interface. Sighted users of such programs will be familiar with the concepts involved, but for those who have not used such a program some of them will be briefly described here. Other concepts are described later on, as necessary.

The first obvious addition to a wimp-based system is a hardware pointing device. This usually takes the form of a *mouse*, although another device such as a tablet and stylus or puck, a trackball or a joystick may be used. The standard pointing device on the Macintosh is a mouse. Linked to the mouse is a pointer on the computer's screen. This is usually referred to as a *cursor*. Unfortunately this can be confused with the cursor used to mark the currently active position in a document displayed on the screen. Therefore in this manual the latter is always referred to as the *text cursor*; any reference to the *cursor* can be taken to refer to the position marker attached to the mouse. As the mouse is moved around on a flat horizontal surface its motion is detected and reflected in movements of the cursor on the screen. If the mouse is moved to the right or left, the cursor moves to the right or left respectively. If the mouse is pushed away from the user, the marker moves up the screen and it is moved down the screen by pulling the mouse towards the user.

The mouse has a button on its upper surface. Pressing this button is usually referred to as *clicking*. Many interactions are performed by moving the mouse such that the cursor points to a chosen object on the screen and then the mouse button is clicked. An alternative form of interaction is achieved by clicking the mouse twice quickly in one spot. This is referred to as *double-clicking*. A third form of interaction is the *drag*, whereby the marker is moved to point at an object, the button pressed and held down. Then, with the button still depressed, the mouse is moved to a new position.

The cursor on the screen gives the user continuous positional feedback. The basic skill required of the user in using the mouse is *hand-eye* (or *visual-motor*) coordination. As the mouse is moved, the user receives feedback about the position of the marker on the screen relative to the items displayed thereon. The positioning of the mouse works in a *relative* fashion. There is no absolute correlation between the position of a mouse on a desk and the cursor on the screen; picking the mouse up and putting it down in a different position will *not* cause the cursor to move correspondingly. This implies that the user is almost entirely dependent on the (visual) information of the position of the *cursor*. Information

about the position of the *mouse* (either visual or kinesthetic) is of less value.

It must be recognized that, although the mouse is an input device, it is intimately associated with the computer output. Contrast the mouse with a keyboard: a keyboard is an input device, and a touch-typist can use a keyboard without receiving any output feedback from the computer. (Indeed, this is how many blind typists work on conventional typewriters). However, no one can use a mouse without receiving some form of feedback (output) regarding the current state of the cursor.

The user interface to Soundtrack consists of *auditory objects* each of which has the following properties:

- a sound;
- a name;
- a spatial position;
- an action.

The sounds used by Soundtrack are basically musical tones of different pitches. However, in Soundtrack 2 a number of different tone types are available which the user can select. These are described in Section 4 below.

The auditory objects are part of Soundtrack's *auditory screen*, which is illustrated in Figure 3.1. The interface to Soundtrack is implemented entirely in an auditory form, but all actions are also reflected by a visual representation on the screen. The visual form makes Soundtrack more accessible to people with some sight. Also such *redundancy* in interfaces makes them more flexible and accessible. In fact, although Soundtrack was designed with an auditory interface for use by blind people, it could equally be used by a sighted deaf person.

One of the constraints imposed on Soundtrack's design is that auditory objects do not overlap, hence their layout is based upon grid patterns. Thus the auditory screen consists of a 4×2 grid of *auditory windows*. As the user moves the mouse around the cursor moves within the screen and tones are generated corresponding to the window which the cursor is in at the time. As mentioned above, the user can select one of a number of different sound patterns. However, they all are based on different *itches* of tones, and the general pattern is that the pitch increases from left to right and from bottom to top. The edges of the screen are marked by a different, distinctive sound - a buzz.

At any time the user can get more information about the current location of the cursor by clicking the mouse. This will cause the name of the object which the cursor is currently in to be spoken. Each of the windows has a fixed role in the program, though its contents

may change. Some windows can be empty. For instance, if no documents have been opened or created, then the Document windows will be empty.

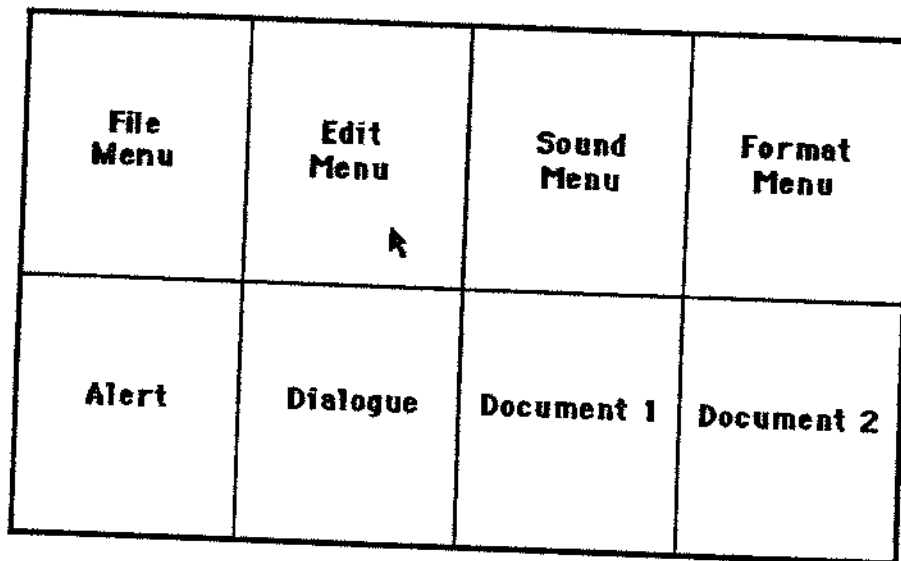


Figure 3.1. The layout of Soundtrack auditory screen, consisting of eight *auditory windows*. The cursor is currently in the Edit menu window.

There are two levels of interaction with the windows. To move to the second level, the user must *activate* a window. To do that they must move the cursor into the chosen window and double-click the mouse button. That will cause the window to become subdivided into a number of component objects. These follow the same rules as the windows: they are arranged in some form of grid and they have the same properties of having a sound, a name, a position and an action. For example, Figure 6.1 shows the activated File menu window.

On activating a window, the user will hear two sounds. The first is the sound for the window and the second is the one for the item within the window at which the cursor is currently pointing. If the cursor is moved out of the active window, a short buzz will be sounded as the marker crosses the edge of the window. Subsequently returning to the active window should be obvious because two sounds will be generated again, as above. At any time, at most one window can be active (there are situations in which none are active). Whenever a new window is activated the previously active window is automatically deactivated. All windows must be activated before the user can interact with them. The active window is displayed on the screen in reverse video (i.e. white on black). This is reflected in the diagrams of windows within this manual.

The mouse-button protocol is consistent throughout the operation of Soundtrack: a single-click will produce speech, naming the object currently occupied by the cursor, and a

double-click will *execute* that object. Executing an object causes its action to occur and also has a visual and auditory effect: the object on the screen flashes and its tone is sounded. In the case of windows, execution has the effect of activating the window. The effect of repeated double-clicks is obtained by holding down the mouse button after the second click. This causes the current object to be repeatedly executed until the button is released. Some objects can be *dragged*. Dragging is achieved by double-clicking on the object, holding the button down on the second click and then moving the mouse.

Objects may be *invalid*. That is to say, that in particular situations it is impossible to execute a given command, in which case it will be impossible so to do. If an object is currently invalid then clicking on it will cause its name to be spoken, preceded by a short 'buzz'. If the user nevertheless tries to execute (double-click) that object then an error will be generated (See Section 8, on Alerts).

4. Sounds

Two forms of sound are used in Soundtrack: musical tones and synthetic speech. Speech is used to communicate precise information, but is quite slow whereas tones can be used to communicate quickly, but with less precision. The pitch of the tones used gives the user spatial information as to the location of the cursor, but if the user needs to be sure which object it is pointing to, they can have the object's name spoken.

Soundtrack 2 allows the user to select the kinds of sounds used. They are all based on the same principle that *pitch* conveys *spatial* information: the pitch of the sounds increases from left to right and from bottom to top.

Chords. The default sound pattern is *Chords*. These are sine-wave tones. The component objects of an activated window have pure tones, which are separated by major thirds. The tone for an unactivated window is based upon a *root* note. That root is the same note as the bottom left-most component object. If the window is filled, the tones of all the component objects are sounded as a chord. This means that the user hears two forms of information; the pitch of the root tells them about the spatial location of the window and the number of notes in the chord gives an indication of the number of component objects in the window. Most users will be unaware of the actual number of notes, but will perceive a richer sound when more notes are present. This pattern implies that the tone for an empty window is a pure note at the pitch of the tonic. The roots of horizontally adjacent windows are separated by a whole tone, and (on the 4 x 2 screen) vertically adjacent ones are separated by a fifth.

Square waves. The chords are based upon sine wave tones, which have a very pure

sound. Square wave tones have much more of a 'buzzing' quality. In this mode each object has a single tone of a given pitch - including filled windows. Pitches increase by five semitones. This implies that two horizontally adjacent windows have pitches separated by five semitones, but vertically adjacent ones are separated by twenty semitones.

Double beeps. This mode is similar to Square waves, but for each object *two* tones are sounded. The first is the note of the lowest note in the current context and the second note is that of the current object. So, for example, entering an unactivated window will cause the tone for the **Alert** window (the bottom leftmost window) to be sounded, followed by that for the window being entered. Within an activated window the tones will be that for the bottom left-most object and that for the current object. Double beeps give the user a reference by which to gauge the current position of the cursor. Obviously, if the tones are of identical pitch then the cursor is in the bottom left-most object. Otherwise the interval between the notes indicates the distance from that root object. Double beeps take relatively long to sound, so that if the mouse is moved too quickly two distinct beeps may not be heard.

5. Dialogues

Often when a user executes an object, Soundtrack needs more information before it can complete the associated action. For example, if the user wishes to open a file from the disc, Soundtrack must be able to determine the name of the file. Such additional information is obtained via the **Dialogue** window.

Individual dialogues are described below as part of the description of the objects which cause them to be filled, but they all have certain features in common. All dialogues have a special object which identifies the dialogue. This is always the bottom object in the dialogue window. Clicking on it causes its name (and hence the identity of the dialogue) to be spoken, and double-clicking has the same effect.

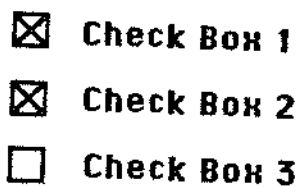


Figure 5.1. Check Boxes. Check Box 1 and Check Box 2 are 'checked' and therefore on. Check Box 3 is off.

Dialogues contain objects which are analogous to *check boxes* and *radio buttons* in visual Macintosh interfaces. Check boxes retain and display a state which can be either on or

off. For example, in Figure 5.1 Check Box 1 and Check Box 2 are on, while Check Box 3 is off.

Radio buttons also display and control on-off settings, but they are arranged in groups, and work such that only one of the group can be on at any time. Setting one on, also sets another to off. In Figure 5.2, Radio Button 2 is on. If Radio Button 3 was set on, then Radio Button 2 would go off.

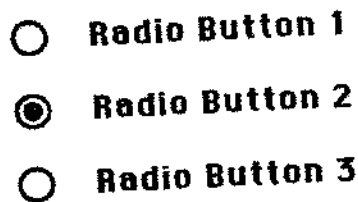


Figure 5.2. Radio Buttons. Radio Button 2 is currently on. At any time exactly one radio button in a group will be on.

Soundtrack includes objects analogous to both check boxes and radio buttons. They work in the same manner described above. Their current state (on or off) is indicated by speech when they are clicked (e.g. "Radio button 2, on"). It is also signalled visually by an on button being displayed in inverse video. Since the button must be in the activated window which is black, that implies that it will be white. Executing a check box *toggles* its state (off to on, or on to off). Executing an off radio button sets it on and the one which was on to off. Executing an on radio button has no effect.

Normally, having specified that an action should be carried out which causes the Dialogue window to be filled, Soundtrack cannot proceed until it has the supplementary information. In such situations the user is forced to activate the dialogue window before they can do anything else. That is to say that such dialogues are *modal* in operation. Once the Dialogue window has been filled with a modal dialogue double-clicking anywhere outside the dialogue window is an error (though single-clicks are allowed anywhere, assisting the user to locate the correct window). In this way the user is forced to activate the dialogue.

In case a modal dialogue should be opened in error they all include a Cancel object. Executing Cancel causes the original command to be canceled, and the dialogue window is emptied and deactivated.

Some dialogues are modeless. Once the dialogue window has been filled with a modeless dialogue it remains so until it is replaced by another dialogue. That means that at any time the user can move to the dialogue and interact with it, but they are never forced to use it.

6. Menus

Visual Macintosh programs make use of *pull-down menus*. The top four of the auditory windows in Soundtrack have the role of menus. A menu is activated in the manner described above (this is the analogous action to *pulling-down* a visual menu). The File menu, for example, will be transformed as shown in Figure 6.1. All menus have a similar format, in that they are one-dimensional, with all their entires arranged vertically. (See Edwards 1987, pp. 60-61, for a discussion of menu design).

Double-clicking on a menu entry will cause an action to occur. These actions are all described below. Some menu entries have *keyboard equivalents*. That is to say that those commands can be executed by pressing keys on the keyboard instead of activating the appropriate menu and then executing the corresponding entry. All keyboard equivalents on the Macintosh are *command codes*. A command code is a combination, generated by holding down the *command key* (marked \mathcal{C} on the keyboard) and pressing another key. The command key works rather like a shift key. For example $\mathcal{C}X$ is generated by holding down \mathcal{C} and pressing X. Not all menu entries have a keyboard equivalent, as is conventional in visual Macintosh programs. Depending on the speech options selected, the fact of an entry having a keyboard equivalent may be signalled when a menu entry's name is spoken. (See Section 6.3). A complete list of keyboard equivalents is given in Table 6.1.

Menu	Command	Keyboard equivalent
File	New	N
	Open	O
	Quit	Q
Edit	Cut	X
	Copy	C
	Paste	V
	Find	F
Format	Print	P

Table 6.1. Keyboard equivalent commands. They are generated by holding down the \mathcal{C} key and pressing the corresponding letter, e.g. $\mathcal{C}Q$ for Quit.

6.1 The File Menu

The File menu (Figure 6.1) controls the opening, closing and saving of document files, as well as enabling the user to quit from Soundtrack. Executing New will cause a new, empty document to be created which will fill one of the Document windows. That

document will initially be named 'Untitled', and the user can give it a proper name when they save it onto disc. To input text into that document, the user must firstly activate the appropriate window. The lower numbered Document window will be used. In other words, if neither is filled, then **Document 1** will be filled, but if **Document 1** is already filled, then **Document 2** will be used. Should both **Document** windows be filled, then **New** will be invalid.

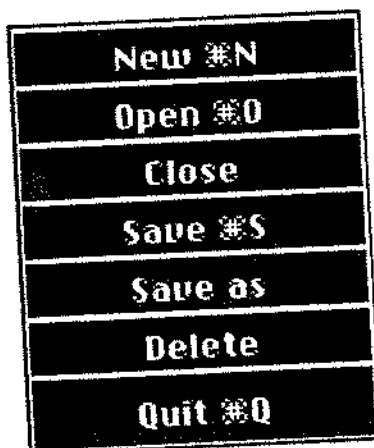


Figure 6.1. The File Menu.

Open is similar to **New** except it enables the user to open an existing file which has been saved on a disc. Having executed **Open** the user must specify the name of the file to be opened. This is done through the **Select file** dialogue (Figure 6.2).

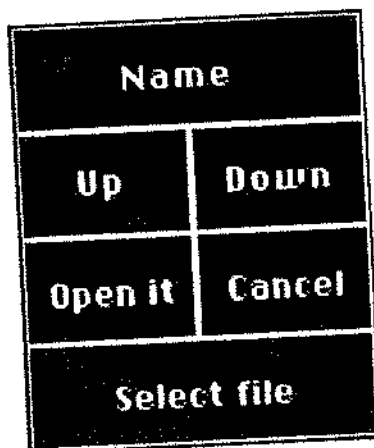


Figure 6.2. The Select file dialogue.

The names of all text files on the disc are displayed one at a time in the **Name** object. These names are arranged in alphabetical order. The user can move the list up and down, so that different ones are displayed in the **Name** object, by executing the **Up** and **Down** objects. The user can find out the name of the currently displayed file name by clicking on the **Name** object. When the required name is displayed, the user can open that file by

executing either the **Name** or the **Open** it object. Executing **Cancel** will abandon the **Open** command, and empty the **Dialogue** window.

Recall that an object can be executed repeatedly by holding down the mouse button on the second click of a double-click. Thus, if the file name that you want to open starts with a letter late in the alphabet you can quickly move down the list towards it.

Notice that Soundtrack can only edit *text* files. Formatted files which have been created with other word processors cannot be processed by Soundtrack - at least not directly. Any such files will not show up in the **Select file** dialogue's name list. Most word processors allow the user the option of saving a file in text format. That means that they are stored without formatting information, such as changes of font and paragraph layouts. Such files *can* be processed by Soundtrack, so this is an indirect way of editing files created by other word processors using Soundtrack.

Documents which are opened or created are held in the computer's memory until such time as the user specifies that they should be written to the disc. That means, for example, that if the user opens a document file which was already on the disc called *memo* then effectively two copies of the file then exist: one on disc and the other in memory. Unless and until the user explicitly saves the file back onto disc, the disc copy of the file is unchanged.

A document which has been created through the **New** command is untitled, whereas one which has been **Opened** has the same name as the disc file. Any document can be saved to disc *and* given a name through the **Save as** command. That is to say that **Save as** will give an unnamed document a name, or change the name of one which did already have a name. In either case a disc file will be created with the specified name.

Executing **Save as** causes the **Dialogue** window to be filled, as shown in Figure 6.3.

The user can then type in the name for the new file. Clicking on the **Name** object will elicit the name typed (so far). Once the name has been entered correctly, the user has several options to specify that the file should be saved with that name: double-click on the **Name** object, double-click on the **Okay** object, press the *Enter* key or press the *Return* key. Executing **Cancel** will cause the close command to be abandoned; the current document will remain unnamed and not saved on disc.

If the active document is untitled (created through **New**) and the user executes **Save** from the **File** menu, then the **Dialogue** window will be filled with **Save with what file name** dialogue, and the file can be given a name as described above. If the active

document already has a name, then executing **Save** will cause it to be written to a file with that name without further intervention.

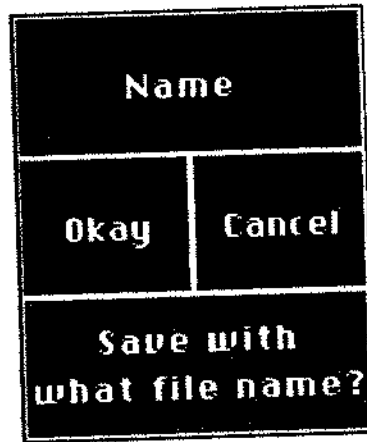


Figure 6.3. The **Save with what file name** dialogue.

Close closes a document, emptying its window. If the document has not been altered at all since it was opened (i.e. no editing operations apart from **Copy** and no typing has been performed on it), then it will be closed and the active document window emptied. However, if any changes have been made the user is given the opportunity to save its contents on disc. The **Dialogue** window will be filled with a dialogue which asks the user whether they want the modified document to be written to the disc. If the document is one which was created, using the **New** command, and it has not previously been saved, then the dialogue will appear as in Figure 6.4, asking whether to save the *unnamed* document.

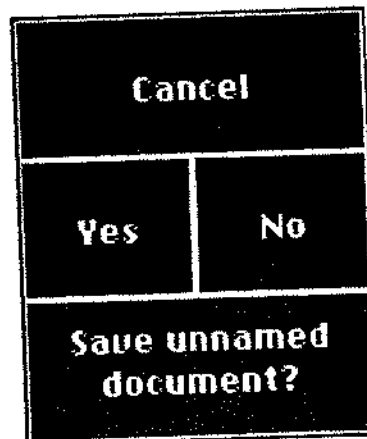


Figure 6.4. The dialogue asking whether the user wishes to save a new document, which has not been named.

If the user executes **Yes**, they will have to go through the same procedure described above when executing **Save as**. The **Dialogue** window will be re-filled with the dialogue illustrated in Figure 6.3, and it must be reactivated. The the user can type in the

file name, as described above. Executing **Cancel** will cause the close command to be abandoned: the current document will remain open and untitled.

Files can be deleted from the disc, using the **Delete** command. Executing **Delete** causes the **Dialogue** window to be filled with a **Select file** dialogue. As shown in Figure 6.5, this is similar to the **Select file** dialogue already seen (Figure 6.2), except that it contains an object named **Delete it** instead of **Open it**.

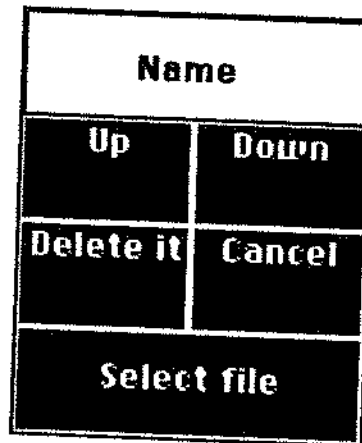


Figure 6.5. The **Select file** (for deletion) dialogue.

The file with the appropriate name can be selected as before. Assuming the chosen victim was called *Name*, the **Dialogue** window would be re-filled with another dialogue, which would have to be activated, and then would appear as in Figure 6.6.

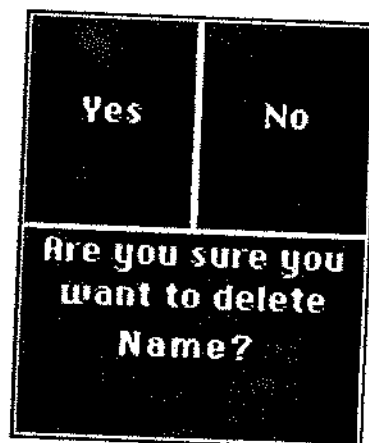


Figure 6.6. Your absolute last chance to change your mind about deleting a file.

Double-clicking **Yes** will delete the file from disc, but if you have changed your mind or made an error, then executing **No** will cause the deletion to be abandoned.

If there are any documents open when **Quit** is executed, they are closed before

Soundtrack terminates. That is to say that the user must go through the procedures outlined above, as if they had executed **Close** for each of the open documents. Thereafter the program will close down.

6.2 Edit menu

The edit menu is shown in Figure 6.7. All editing operations act on the current selection. (See Section 7). **Copy** does not alter the active document or its contents, but it makes a copy of the selected text in memory. Executing **Paste** will cause that held text to be inserted into a document. So, for example, the user might copy a sentence and move the selection to another point in the document. Executing **Paste** will cause a copy of the sentence to be inserted at that point. **Cut** also causes a copy of the selected text to be held in memory, but also deletes the selection. Both **Copy** and **Cut** are invalid if the selection is a point, and **Paste** is invalid if nothing has been cut or copied.

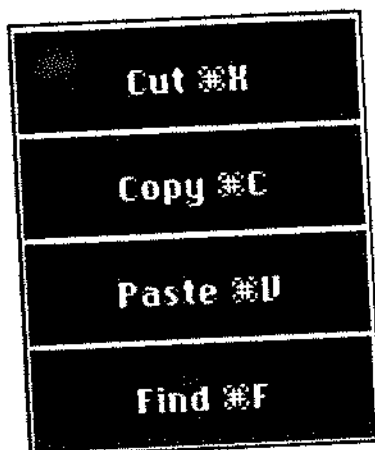


Figure 6.7. The Edit menu.

The **Find** object enables the user to search for a string in the current document. Executing it causes the **Dialogue** window to be filled with the dialogue shown in Figure 6.8.

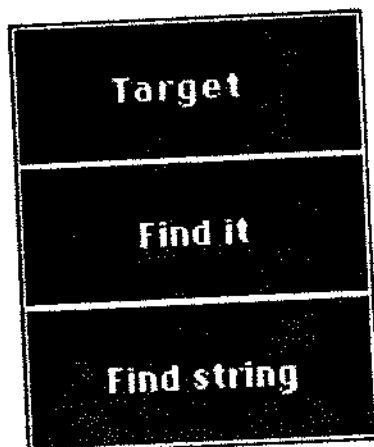


Figure 6.8. The Find string dialogue.

When this dialogue has been activated the user can type in the string to be found. Clicking on the **Target** object will cause the string typed (so far) to be spoken, and double-clicking **Find it** or the **Target** object will cause the target to be sought in the current document. As with the **Save as** dialogue, the *Enter* or *Return* key can also be used to make the find commence. The **Find string** dialogue is modeless, so the following scenario is possible. The user executes a find, as described above and then re-activates the document to see whether the correct occurrence of the string has been located. On finding that it is *not* the required occurrence, the user can reactivate the **Find string** dialogue (that is, without executing **Find** from the **Edit** menu again), and the same target string will be still be current, so that executing **Find it** will cause the next occurrence to be sought. Alternatively, any typing in the re-activated **Find string** dialogue will cause the target to be replaced by whatever is typed.

6.3 The Sound menu

The user can customize the auditory interface - to a limited extent - through the Sound menu (Figure 6.9).

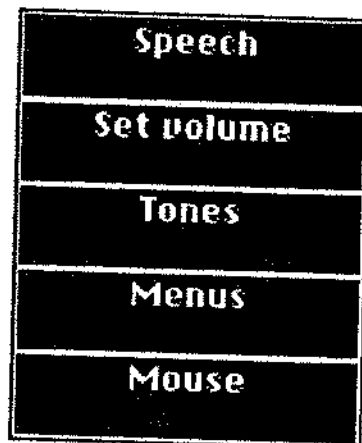


Figure 6.9. The Sound menu.

Executing **Speech** causes the **Dialogue** window to be filled with the **Speech** dialogue shown in Figure 6.10, which is modeless. Check boxes and radio buttons which are **on** are signalled visually by being displayed in reverse video. Since an activated window is shown as white-on-black, any such **on** buttons are seen as black-on-white.

The **Say chars** and **Say words** objects are radio buttons. The default setting is for **Say words** to be **on**, in which case all typing, and text within documents will be spoken word-by-word. So, for example, if the user is typing into a document their keystrokes are not echoed until they complete a word (type any non-letter character). Selections within a document will also be spoken as words, rather than being spelled out (unless the selection level of the document is **character** - see Section 7). If **Say chars** is switched on, then

speech becomes character-oriented.

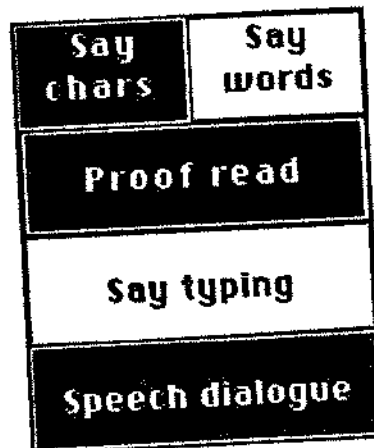


Figure 6.10. The Speech dialogue. **Say chars** and **Say words** are a pair of radio buttons, and **Say words** is currently on. Similarly **Say typing** is a check box which is on.

The **Proof read** and **Say typing** objects are check boxes. If **Proof read** is on then all punctuation is spoken and capital letters are signalled. Notice that these settings affect speaking of text in alerts and dialogues as well as that in documents.

As an example of proof reading, the following sentence:

Well, this sentence contains (quite) a lot of punctuation!

would be read (assuming **Say words** is on) as:

"Well comma space this space sentence space contains space open-bracket quite close-bracket space a space lot space of space punctuation exclamation newline"

By default **Proof read** is off which means that speech is spoken as it would be read, which is a much less wearing way of listening to text.

Users should be aware of some anomalies which will become apparent when proof reading, which arise from the fact that the same characters on the keyboard are used for more than one symbol. The minus sign (-) is also used as a dash in prose. Soundtrack does not distinguish these uses and always pronounces it as *minus*. Similarly the single quote mark (') is also used as the apostrophe. It is always pronounced as *quote* and is treated as marking the end of a word. For example, assuming **Proof read** and **Say words** are on, the word *it's* would be pronounced as *"it quote S"*.

While on the subject of errors Soundtrack makes in speaking characters, it should be said

that Soundtrack recognizes only the standard printable Ascii characters. The Macintosh has a larger character set, whereby extra characters can be typed through use of the Option key. Since the extra characters available varies from font to font, Soundtrack makes no attempt to pronounce these. It will simply speak their Ascii code value (e.g. "Ascii 128").

The user can adjust the volume of the auditory output from Soundtrack by executing the **Set volume** entry in the **Sound** menu. This will cause the dialogue window to be filled with the dialogue shown in Figure 6.11. The bar in this dialogue can be moved as a slider up and down the dialogue, adjusting the volume accordingly. To do this the user must locate the bar, double-click on it and hold the mouse button down while they move it vertically (i.e. drag it).

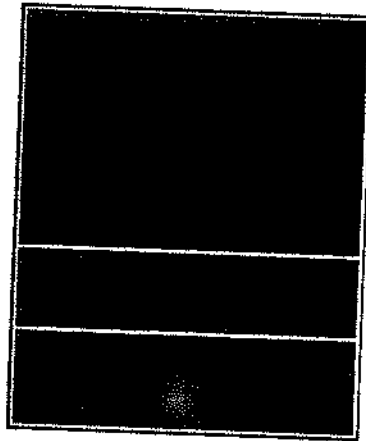


Figure 6.11. The Set volume dialogue.

Users may find that even at maximum volume the tones are not loud enough. The answer may be to employ an external speaker, which can be connected to the Macintosh by way of its jack plug socket. Some users may prefer to use headphones. These can be connected through the same socket. User should beware that if they directly connect personal-stereo-style headphones the volume is likely to be unpleasantly high, even at its lowest level. This is due to the fact that the level of output on the Macintosh jack plug is greater than that on standard personal stereo players (though this appears to vary between models).

The different types of tone which can be used in Soundtrack were explained above, in Section 4. They are selected through a the **Select sounds** dialogue (Figure 6.12) which is accessed via the **Tones** entry in the **Sound** menu (Figure 6.9). Note that **Square wave**, **Sine wave chords** and **Double beeps** form a set of radio buttons.

Variations in the way auditory menus are presented can be controlled via the **Menus** entry of the **Sound** menu (Figure 6.9). Executing it causes the **Dialogue** window to be filled

with the Menu options dialogue (Figure 6.13).

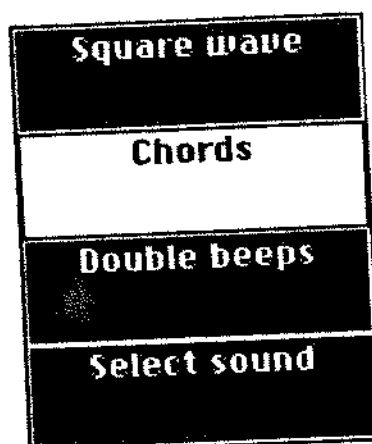


Figure 6.12. The Select sound dialogue.

As explained earlier, some menu entries have keyboard equivalents. **Say command**, **Signal command** and **Silent command** is a set of radio buttons which controls whether or not the fact that an entry has a keyboard equivalent is signalled when the entry's name is spoken. For example, if **Say command** is on and the user clicks on the **Open** entry in the **File** menu (Figure 6.1), Soundtrack will say "*Open, command O*" – because **Open**'s keyboard equivalent is **Command-O**. **Signal command** also causes the keyboard equivalent to be spoken, but instead of the letter being preceded by the word "command" it is preceded by a high-pitched tone. This is more cryptic, but quicker. By default **Silent command** is on, which means that the keyboard equivalent is not spoken, so that novice users are not distracted by this mechanism. The more experienced user may switch **Say command** or **Signal command** on to learn the keyboard equivalents. Once the user knows which entries have equivalents and which letters are used, they may switch it off again, speeding up operation.



Figure 6.13. The Menu options dialogue.

Announce dialogues is a check box. If it is on (the default setting) then whenever the **Dialogue** window is filled the identity of the dialogue is spoken. This alerts the user to this occurrence, which is particularly important if the dialogue is a modal one.

Users can elicit auditory feedback whenever they click the mouse. This is controlled through the **Mouse** entry in the **Sound** menu (Figure 6.14). Executing it fills the **Dialogue** window with the **Mouse option** dialogue. If **Echo clicks** is *on* clicks of the mouse button are accompanied by a sound. This facility may be useful to some users confirmation that they have indeed clicked the button. Soundtrack will start with the **Echo clicks** option on if the *Caps lock* key is on when Soundtrack is opened.

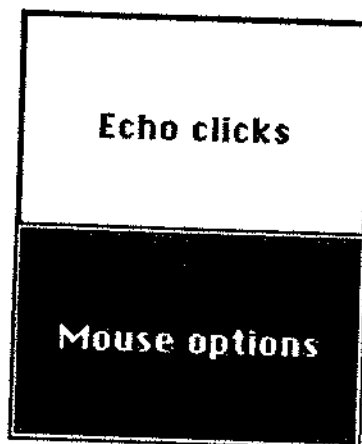


Figure 6.14. The Mouse options dialogue.

6.4 The Format menu

The **Format** menu (Figure 6.15) is used to control the format of document - both as they appear on the screen and if they are printed. In fact the command to print a document is also in this menu.

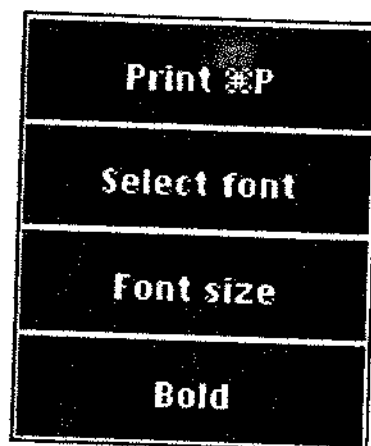


Figure 6.15. The Format menu.

Formatting options in Soundtrack are restricted to selection of the font (or *typeface*), the

size of type and whether the type is emboldened. (See Section 7).

Executing the **Select font** entry causes the **Dialogue** window to be filled with a dialogue through which the user can select the font in which documents will be printed - and displayed on the screen (Figure 6.16).

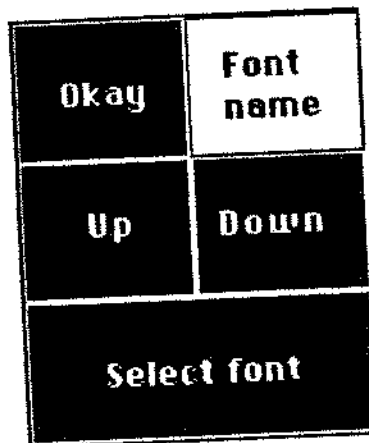


Figure 6.16. The **Select font** dialogue.

Which fonts are available depend on the configuration of the system disc. The names of the fonts available are displayed one at a time in the **Font name** object. The user can scroll up and down the list using the **Up** and **Down** buttons. When the required font is displayed, the user selects it by either double-clicking on the **Font name** object or the **Okay** object.

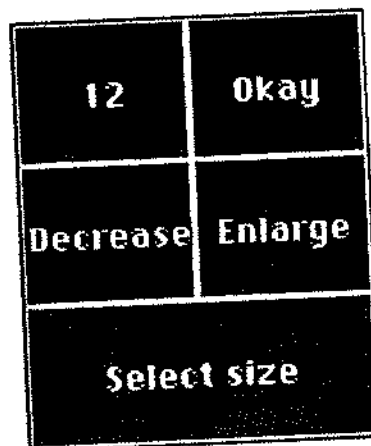


Figure 6.17. The **Select size** dialogue.

The font selection dialogue is modeless. So the user is not forced to interact with it immediately after executing **Select font**, and it remains in the **Dialogue** window unless and until the window is filled with another dialogue. That means that the user can switch back and forth between the dialogue and a document, experimenting with different fonts.

The size of the font can be selected in a similar manner. Executing the **Font size** entry (Figure 6.1) causes the **Dialogue** window to be filled with the dialogue shown in Figure 6.17.

Executing **Print** causes the current document to be printed according to the current settings of the options of font etc.

7. The Document windows

If a **Document** window is empty, clicking in the window will elicit the speech, "*Document one*", or "*Document two*", as appropriate. However, if the window has been filled, then clicking on it will cause the name of the document to be spoken. Newly created documents have no name, and so are described as, "Unnamed document".

When a document has been opened (via **Open** or **New**) the appropriate **Document** window will be filled and then can be activated. The user can now type into the document. By default the typing will be spoken as it is entered (see Section 6.3). Also the typing will be displayed on the screen. In other words, the layout of the screen is overlaid with a window showing the text in the document. Any movement of the mouse causes the text window to be hidden and the auditory screen is re-displayed.

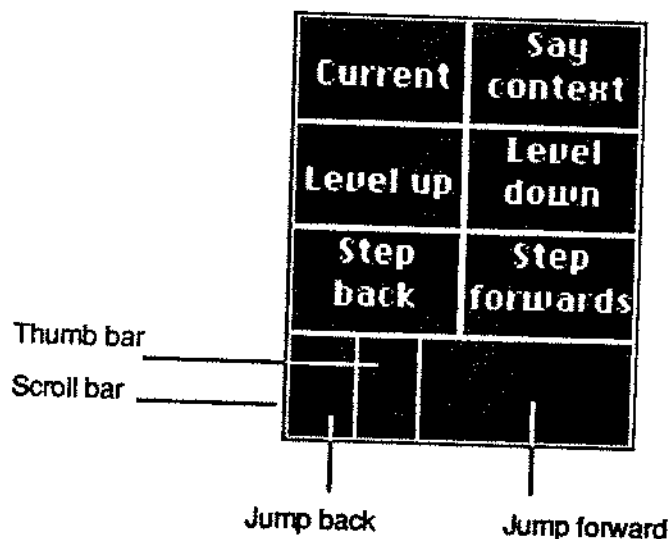


Figure 7.1. The **Document** window. The *Scroll bar* is divided into two parts, **Jump back** and **Jump forward**, separated by the **Thumb bar**, which slides along the scroll bar, changing the relative sizes of the **Jump** controls.

The document window has two properties associated with it: a *selection* and a *level*, and the (activated) **Document** window consists of objects which control these, as shown in Figure 7.1. The *selection* is the active portion of the document; all operations affect the

text which constitutes the current selection. In visual word processors the part of the text which is selected is signalled to the user by it being displayed in a highlighted manner (usually in inverse video: white letters on a black background on the Macintosh). In Soundtrack, the selection is signalled by its being spoken. The selection is defined by a start and an end position. The start and end positions may be coincident, in which case the selection will be a point. In visual word processors (and on the visual display of Soundtrack) a point selection is displayed as a *text cursor* - normally a vertical bar. For instance, in the following word the selection is the point between the *r* and the *s*.

cur|sor

The text selected is communicated to the user by its being spoken. That is to say that every time the selection is changed the new selection is spoken. The exception is when the selection is a point, in which case nothing is spoken when the selection changes.³ There are a variety of modes in which it can be spoken, which are explained below within the description of the **Sound** menu. (See Section 6.3).

As described below, the *level* setting corresponds to a syntactic unit and specifies the unit which is the basis of any changes of the selection. The **Level up** and **Level down** objects control the setting of the level. The available levels are (in descending order):

document,
paragraph,
sentence,
word,
character,
point.

It is not possible to execute **Level up** when the level is **document**, or **Level down** when it is **point** (i.e. the corresponding controls are invalid).

Many visual word processors are designed to work on the spatial layout of a piece of text; they have commands to move the cursor up and down lines, for instance. In a sense the spatial layout of a document is an arbitrary structure, depending on visual attributes such as the lengths of words and size of page (or screen). Using the auditory interface there is

³ Users who are unfamiliar with synthetic speech may initially find it disconcerting that some of the words in a document are not pronounced very well, yet system messages are pronounced correctly (the word *window* for example). This is because Soundtrack has been explicitly programmed to speak its built-in messages.

no need to be concerned about the line structure. In Soundtrack levels and selections are defined in terms of the syntactic structure of documents. If a document is to be displayed visually, either through the visual interface to Soundtrack or when a document is printed, lines are broken automatically to fit the screen or paper. Within a paragraph no line structure is defined.

Specific definitions must be given for the units which Soundtrack treats as paragraphs, words and sentences. Obviously the beginning of the document can be the left-most delimiter of any of these units, and the end of the document can be the right-most. Generally, though, a paragraph is defined as the text contained between two newline characters, including one or more newlines which come at the end of the paragraph.

A word is defined as an unbroken string of letters and/or digits. A word will thus be delimited by two characters which are not letters or digits (i.e. usually a space or a punctuation mark). If a word is immediately followed by a space, then that space is treated as part of the word. This makes editing easier. For instance, if a word is cut from one place it can be pasted directly in another place and the correct spacing will be automatically maintained. One unfortunate deficiency in this definition of a word is caused by the ambiguous use of the single quote symbol as both a quotation mark and the apostrophe. Without a certain amount of analysis of a piece of text it is not possible to decide which role the symbol is in at any one appearance. The definition used here treats it always as a quotation mark, which means that apostrophized words are treated as two separate words.

The end of a paragraph (a newline character) may mark the end of a sentence. Otherwise, a sentence is defined as being terminated by either a full-stop, an exclamation mark or a question mark which is followed by one or more spaces which in turn is followed by a capital letter. This definition aims to reduce possible confusion which might be caused by the use of full-stops in other roles. For instance it would mean that the sentence,

An envelope marked O.H.M.S. has been delivered.

would be treated as such, and not effectively truncated after any of the full-stops in *O.H.M.S.* Of course this definition is still not clever enough to recognize,

My name is A. Edwards.

as one sentence. A sentence is defined as starting with the first capital letter following the end of the previous sentence, as defined above.

Altering the level will cause the selection to be adjusted (and hence spoken). Moving the

level up will cause the end of the selection to be extended forwards through the text to the end of the nearest unit of the new level. For example, if a character is selected and the level is moved up to **word** the end of the selection will be extended to the end of the current word. The beginning of the selection (i.e. the original character) will not be affected. Moving the level down correspondingly causes the selection to be contracted from the right. For instance, if a sentence is selected and the level is moved down to **word** the selection will become the first word of the sentence.

The selection can also be altered by moving it through the document. The *step* objects cause the selection to move in units corresponding to the current level. Suppose, for example, that the current level is **sentence** and a complete sentence is selected. If **Step back** is executed then the previous sentence will become the new selection, as illustrated in Figure 7.2.

Current	Say context
Level up	Level down
Step back	Step forwards

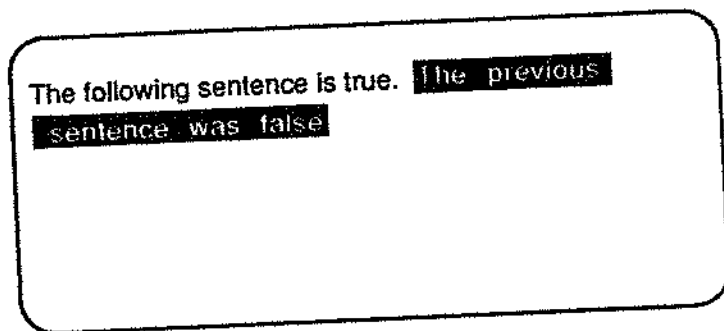



Figure 7.2a. The level is **sentence** and a sentence is selected - as shown by being displayed in inverse video.

Current	Say context
Level up	Level down
Step back 	Step forwards

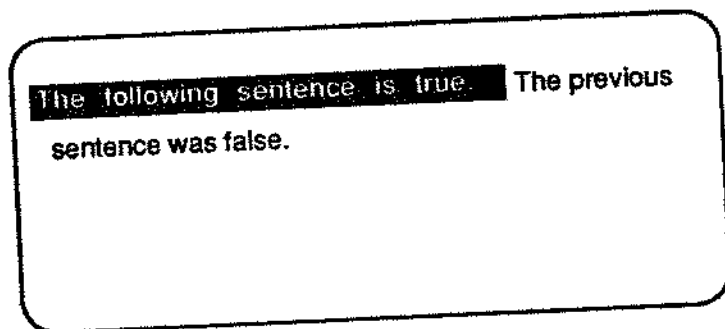


Figure 7.2b. The **Step back** control has been executed and the first sentence is now selected.

It is possible for the level and the unit currently selected to be different. For example, suppose the level is character, and one character in the middle of a word is selected:

This **statement** is false.

Then if the level is moved up to **word**, the selection will be extended to the right to the end of the word:

This **statement** is false.

So, now, although level is **word**, less than a whole word is selected.

The movement of the selection is implemented in a consistent fashion. For instance, if the selection is moved backwards then the start of the selection is moved to the left to the first starting position appropriate to the current level. The end of the selection is then adjusted - if necessary - to correspond to that new start. So, in the example, executing **Step back**, would adjust the selection to become the whole word:

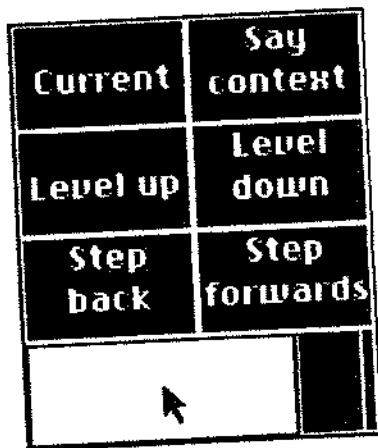
This **statement** is false.

The *jump* controls allow the selection to be moved in larger quanta. In fact the selection is moved through the distance corresponding to the unit one level up from the current level. That is to say that if the current level is **word**, for instance, and a **Jump back** is executed, the selection will move to the first word of the sentence, and so on. Figure 7.3 gives an example of this.

Current	Say context
Level up	Level down
Step back	Step forwards

The following sentence is true. The previous sentence **was** false.

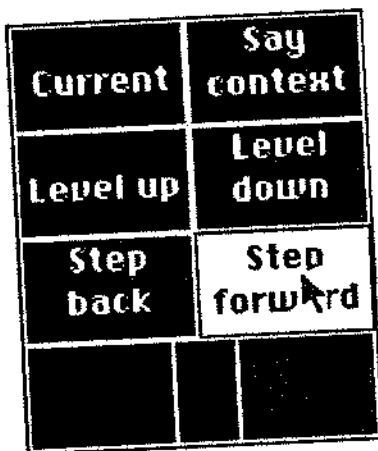
Figure 7.3a The level is **word** and the word *was* in the second sentence is selected.



The following sentence is true. **the** previous sentence was false.

Figure 7.3b. The Jump back control has been executed and the selection has moved to the first word of the sentence.

Selections can be *extended* if a **Shift** key is held down on the keyboard while the user alters the selection. For example, if the level is **word** and a word is selected (as in Figure 7.3b), then if **Step forward** is executed while a **Shift** key is held down the *end* of the selection will be moved to the end of the next word, but the start of the selection will not be moved, as shown in Figure 7.4.



The following sentence is true. **The previous** sentence was false.



Figure 7.4. The Step forward control is executed while a **Shift** key is held down. This causes the *end* of the selection to be moved forward, while the *start* remains unchanged.

The **Thumb bar** provides another means of moving the selection through large distances. It works according to the same principles as its visual counterparts. The position of the **Thumb bar** within the **Scroll bar** reflects the position of the selection

within the document. So, if the selection is at the start of the document the **Thumb bar** will be at the left-hand edge of the **Scroll bar**; if it is at the end of the document, the **Thumb bar** will be on the right and intermediate positions will be represented proportionately.

Not only do these objects act as indicators of the selection's position, but they can be used to alter it. The **Thumb bar** can be dragged within the **Scroll bar** and wherever it is 'put down' the selection will be adjusted to reflect the new position. In order to drag the **Thumb bar** the user must double-click within it, but hold the mouse button down on the second click. The mouse is then moved horizontally and the **Thumb bar** will follow the motion. If the button is then released the selection will be adjusted correspondingly. During the dragging the position of the **Thumb bar** is communicated by an intermittent 'bleep' sound. The frequency of the bleep increases from right to left, so that at the left-most edge it merges into an almost continuous sound, while the individual bleeps are clearly audible at the right-hand edge.

Naturally, the selection cannot be moved - by any of the above controls - if it extends over the whole document; all the movement controls will be *invalid*.

A single-click of the **Current level** control causes the identity of the current level to be spoken ("paragraph level" or whatever). A double-click will cause the current selection to be spoken. The text in the document will also be displayed on the screen, with the current selection highlighted. The speaking of the selection can be interrupted by the user clicking the mouse button. If the level is currently **word** or lower this will cause the speech to cease as soon as it has completed saying the selection. If the level is higher it will stop once it has completed saying the next unit corresponding to one level down from the current level. That means, for example, that if the selection is being spoken at **paragraph** level speech will stop at the end of the current sentence. Speech cannot be interrupted immediately because it is sent to the synthesizer in packets and the whole packet must be spoken before the computer can respond to any other events (such as a click of the mouse). Soundtrack uses packets corresponding to one level down from the current level. So, for instance, at sentence level speech is generated a word at a time, and it is possible to interrupt between words. Users will notice, though, that moving up to paragraph level the speech is of better quality - because the speech synthesizer can apply intonation to whole sentences. The method implemented is therefore a compromise. It does allow interruption, while maintaining the spoken quality of longer speeches. If the speaking of the selection is interrupted, the start of the selection is re-set to the beginning of the last unit spoken.

When a document is opened with the **Open** command in the **File** menu, then on activation the level will be **word**, and the selection will be at the end of the document, so that if the user commences typing without adjusting the selection, that text will be appended to the end of the document.

Typing always replaces the current selection. So, if the following text is displayed, with the word *cat* selected:

The **cat** sat on the mat.

and the user wants to replace *cat* by *dog* they just type the replacement. As soon as the initial *d* is typed, *cat* is deleted and replaced by the *d*:

The d|sat on the mat.

The user can then type the remainder of the word - including the space to follow it:

The dog |sat on the mat.

The fact that typing replaces the selection also gives the user a handy means of deleting text as an alternative to using **Cut**. Pressing the **Delete** key will delete the current selection. Notice that this will not put the deleted text in the buffer so it cannot be pasted back, but in fact it means that the paste buffer is unaltered - so that the last thing cut or copied can still be pasted.

While the **Current** control can be used to get the selection spoken, the **Say** context object enables the user to hear the text which surrounds the current selection. If the level is **point**, executing **Say context** will cause the two characters on either side of the selection point to be spoken. At other levels **Say context** enables the user to hear the text one level up without altering the selection or the level. For example, at **word** level, the user can hear the sentence containing the currently selected word. The position of the current selection within the spoken context is marked by a high-pitched tone. For instance, suppose level is **word** and a word is selected, as follows:

To be or not to**be** , that is the question.

If **Say context** is executed, then "To be or not to", will be spoken followed by a beep and then the rest of the sentence: "be, that is the question." In this way it is possible to know which of the occurrences of *be* is selected.

At **character** level, the user can hear the word containing the currently selected character. If **Say characters** is set then each of the letters of the word will be spoken, the selected letter being preceded by a beep, as described above. However, if **Say characters** not set then the word is spoken as a whole, and there is no beep to indicated the position of the selection.

There being two document windows, means that the user can be working on two documents at a time. That includes the possibility of making interrelated changes in two documents. For example, a paragraph may be copied from one document and pasted into another. By definition, the document window which was more recently active is the one on which menu entries act. For example, suppose both document windows are filled and the user activates **Document 2** and types something into it. If the user then activates the **File** menu and executes the **Close** command (see Section 6.1), it is **Document 2** which will be closed. Given this rule, there should be no confusion; there is no chance that in these circumstances **Document 1** would have been closed.

When a document is opened (via **New** or **Open**) it automatically becomes the current one. Otherwise the way to make a document the current one is to activate its window. So, as another example, suppose the user wants to insert the entire contents of the document they are currently working on at the end of another document which is on disc. They move the level in the current document to **Document** and execute **Copy**. Then they open the appropriate file from disc, so that it fills the other document window and becomes the current document. Executing **Paste** now will complete the desired operation. Notice that it was not necessary to activate the window containing the new document to make it the current one – though it would not have caused any harm to have done so.

Another example would be copying some text from the current document to the one displayed in the other document window. The sequence would be something like:

Adjust the selection to the desired text;

activate the **Edit** menu;

execute **Copy**;

activate the other document window (to make its document current);

activate the **Edit** menu;

execute **Paste**.

(Of course, this operation could be simplified if the keyboard equivalents for Copy and Paste – ⌘C and ⌘V respectively – were used).

The selection can also be altered by searching for strings in a document, using the Find command in the Edit menu (Section 6.2). The search will proceed forwards from the current selection. If the target is located it will become the new selection. The new selection will not necessarily be related to the current level. Suppose, for instance, that the level is **character** and a character is selected:

The quick brown fox jumps over the lazy dog.

If the user then attempts to find the word *fox* that word will become selected:

The quick brown **fox** jumps over the lazy dog.

If the string does not occur between the current selection and the end of the document, the selection will be unchanged. Searches are all literal, so that only strings which exactly match the target (including the use of upper- and lower-case letters) will be found. So, for instance, *Fox* would not have been found in the above example.

8. Errors and alerts

Soundtrack has been designed in such a way that the opportunity to make errors has been reduced. For instance, objects are made invalid (making it impossible to execute them) whenever it is not possible to carry out their action. That an object is invalid is signalled by a short buzz sound when its name is spoken.

Nevertheless errors can occur and the fact that they have occurred is communicated to the user through sounds and through the Alert window. Errors fall into two broad categories: user errors and system errors. For the most part, user errors are less serious. For example, attempting to execute an invalid object is a user error. It is not a damaging error, since nothing will be changed by the failed execution. If a user commits such an error, the fact is signalled to them by a buzzing sound. Often the user will then realize the nature of their error, and can proceed without further action. If, however, the user hears an error buzz which they cannot explain then they can get more information through the Alert window. Immediately after such an error, the Alert window will be filled with an alert as illustrated in Figure 8.1. If the user activates that window, and clicks on the Message object they will be given a spoken description of the error (e.g. "Cannot activate empty document window"). Executing the Okay object will cause the Alert window to be emptied.

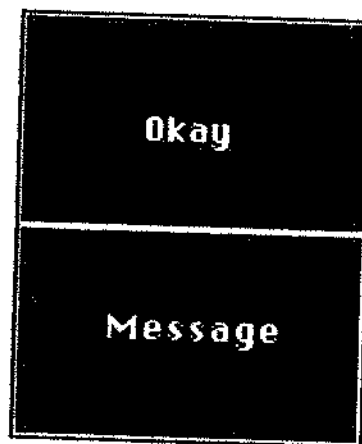


Figure 8.1. An alert.

Notice that in the case of user errors, the alert will be modeless, so the user is *not* obliged to activate the **Alert** window. If the user performs any other action, the **Alert** window will be automatically emptied. System errors, however, are generally more serious and indicate an event has occurred which must be responded to before the user can continue. For example, if the user has specified that a document should be saved on disc, but there is insufficient space on the disc then the save cannot occur. The user must be made aware of this and that the save operation has not occurred so that they can take appropriate action (i.e. deleting other files to make space). In such a case the **Alert** window is modal in operation, which means that the user cannot activate any other window, but must activate the **Alert**, should read its message and then execute **Okay**.

It is possible that you will exhaust the memory of your Macintosh. If you do so the **Alert** window will be filled with the alert illustrated in Figure 8.2. This is a modal alert, and once you have executed **Okay** Soundtrack will automatically close down. You will have the opportunity to close any open documents - as if you had selected **Quit** (from the **File** menu, Figure 6.1)

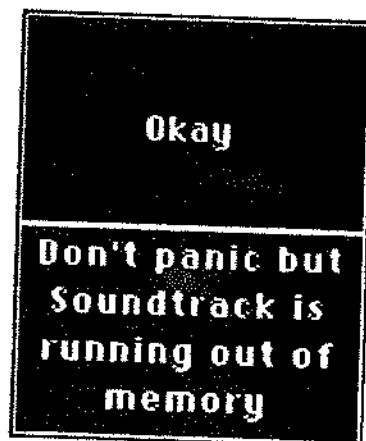


Figure 8.2. The **Don't panic** alert.

When this situation occurs Soundtrack does release some free memory, so that you should have sufficient memory for the close-down to occur gracefully. Once the program has closed down in this way, you can re-start it (with a clean sheet in memory, as it were) and continue your word processing.

It is possible that a fatal system error may occur while you are running Soundtrack. In such a case the dreaded 'bomb' system alert will appear on the screen, but unfortunately there will be no audible signal of this fact, all you will know is that the program becomes completely unresponsive. All you can do is press the Enter key which should cause the system to re-boot. Failing that you may have to switch the power off and on again.

9. Using Soundtrack with other adaptations

Adaptations are beginning to become available to make the Macintosh accessible to people with different forms of disability. These are often *transparent* software adaptations which allow access to other Macintosh programs.

Close View is one such adaptation for partially sighted users. It is a piece of software which enlarges the contents of the screen display. Partially sighted Soundtrack users might find it easier to use if they have access to the visual display as well as the auditory feedback, and some of them might benefit from using *Close View*.

Physically disabled users who find the mouse difficult to manipulate may prefer to control the cursor through the keyboard. *Easy Access* makes this possible, and can also be used with Soundtrack. Some people may prefer instead to use a different hardware device instead of the mouse, such as a *track ball* or a *bit pad*.

10. Limitations

Soundtrack 1 was a prototype, which had a number of acknowledged omissions and, although Soundtrack 2 is intended to be a piece of production software, it still has some limitations. These are described below.

10.1 Soundtrack is not an operating system

One intention in the design of Soundtrack is that it should provide the facilities which a blind user would need to word process on the Macintosh. So, the user can:

- create documents,
- print them,
- save them on disc,
- modify them,

- rename them ⁴,

and when finished with them delete them from the disc. In other words, as far as possible a blind user should be able to use Soundtrack without assistance from someone able to use the visual Finder interface. However, the facilities included had to be restricted; it was impractical to attempt to provide all the functions of the *Finder* within a word processing package. This explains the absence of some features which experienced Macintosh users might have expected, and these are listed below.

10.2 Features which were too difficult to implement

Although Soundtrack 2 is intended to be usable in real applications, its primary *raison d'être* is as a research vehicle. This means that it has been considered not worthwhile building in some features which Macintosh users might expect. The Macintosh is so fundamentally visually oriented that some features which are part of most Macintosh software are difficult to translate into an auditory form. Other features would also take a significantly greater effort to implement.

Limitation on maximum document size

No document to be edited by Soundtrack 2 can consist of more than 32767 (32 K) characters. This is a restriction of the data structure used to contain the contents of documents. The data structure (a *TERec*) is a part of the Macintosh Toolbox, which makes it convenient to use, but it does have this limitation. Also, editing large (but smaller than 32 K) documents is rather slow. These shortcomings could be avoided - but only at the expense of considerable programming work.

Restrictions on multiple fonts and type styles.

These facilities are very attractive to most Macintosh users, and probably account for some of its popularity. However, to have included them in Soundtrack would have implied a significant extra programming effort. All of a Soundtrack document must be in the same font, point size and type style. That style may be emboldened, which may make documents more readable for some partially-sighted users.

Printed documents are always given the same default page layout. This is because the layout can only be altered through a visual dialogue, as explained below.

⁴ Renaming can only be done indirectly: open a file and then execute *Save as*, creating a new copy and giving it a different name.

Undo is not implemented.

A feature of many Macintosh programs which helps users avoid making mistakes is the ability to *undo* commands. Having carried out a command, the user is not committed and can restore the program and data to their previous state by selecting the command Undo. Although such a facility might be particularly useful in an auditory word processor, it has not been implemented in Soundtrack because of the complexity of implementing a general facility.

10.3 Soundtrack does not make use of folders.

All text files handled by Soundtrack must be in the same folder as the program (if the hierarchical file system, HFS, is in use). Although it would have been possible to include facilities within Soundtrack to handle files in different folders, to have included a means of creating new folders would have been to stray into the territory of the operating system.

10.4 The Macintosh is just too visually orientated

It is not possible to reset the default print format from Soundtrack.

This is simply a practical limitation imposed by the design of the Macintosh system software. The only way of resetting the defaults is via a (visual) dialogue. To get around that limitation would require significant programming ingenuity, so this will only be contemplated in future releases of Soundtrack if there is considerable demand for it (indeed, future releases of Soundtrack will only be developed at all if there is such a demand!) As it is, (with due apologies) the default format can only be changed by someone who the conventional visual interface.

Desk accessories are not available.

Desk accessories are special applications which can be run at the same time as other applications. These are accessed by sighted users through a special system menu, marked with an Apple symbol, which is always accessible, whether an application is running and from the *Finder desk top*. Because desk accessories are accessible only through this (visual) mechanism and they are themselves visual it is not possible to access them through Soundtrack.

For the most part, the absence of desk accessories does not affect the user's ability to carry out word processing, but there are a few system parameters which can be set only through desk accessories. In particular, changing the identity or the type of the printer to be used (i.e. Laserwriter versus Imagewriter) can only be done through the *Chooser* desk accessory.

Another important desk accessory is the *control panel*. This enables the user to select such parameters as the keyboard configuration (US or international) and the rate of movement of the cursor. The speed of double-clicks is also set through the control panel (that is a means of specifying the amount of time which must elapse between two clicks for them to be treated as two single clicks, rather than one double-click).

Unfortunately, any user of Soundtrack who wants to change any of these settings will have to get help from a sighted colleague.

References

- Edwards, A. D. N., (1987) *Adapting user interfaces for visually disabled users*, PhD Thesis, the Open University, UK.
- Zachmann, W. F., (1987) *A Look into the Near Future - 13 Predictions for the World's Computer Industry*, International Data Corporation, Framingham, MA.
- Scadden, L. A., (1984), Blindness in the information age: equality or irony? *Journal of Visual Impairment and Blindness*.

Acknowledgements

Special thanks are due to Tim O'Shea who supervised the research leading to the production of Soundtrack 1. He provided many ideas and much of the inspiration behind the research as well as invaluable guidance as to how it should be pursued.

Most of the improvements of Soundtrack 2 over the original prototype arose from the suggestions of volunteer subjects who took part in the evaluation. They are to remain anonymous, but their vital contribution is gratefully acknowledged. The evaluation subjects were recruited from the Royal National Institution for the Blind (RNIB) and the Association of Visually Handicapped Office Workers (Avhow). Thanks can be passed to Corri Barrett and Geoff Jackman of the RNIB and Karl Farrell of Avhow for organizing the volunteers.

Other people who suggested features and improvements were: Steve Draper, Bill Gaver, Thomas Green and Richard Young.

Assistance in developing Soundtrack 2 was received from Apple Computer Inc., through their sponsoring attendance on a *MacCollege* course. Thanks are due to Alan Brightman, their Director for Special Education as well as to Sharon Fleschute for organizing the course and Rick Blair for presenting it.

Unfortunately thanks are *not* due to the UK Apple Developers' Group for any assistance with overcoming technical problems during the development. They seemed unable to fulfil this role, leaving queries unanswered over several months.

Trademarks

The following are trademarks of Apple Computer Inc: Apple, Macintosh, MultiFinder, Laserwriter, Imagewriter, Finder, CloseView.

Macintosh is a trademark of McIntosh Laboratory Inc, Licensed to Apple Computers Inc.

University of York, Department of Computer Science, (YCS) Reports

This report is one of a series published by the Department of Computer Science at the University of York. The series is designed for the purpose of informing friends and colleagues of our experience and views in a manner which strikes a balance between ad hoc memoranda and formally refereed papers. It is intended that appropriate reports should be revised in the light of comments received, and thereafter offered for publication in the technical literature.

- [YCS 1] Developments in Real Time Software. I. C. Pyle & I. C. Wand, December 1976.
- [YCS 2] Heating by Means of a Flowing Medium - Translation of "Über Erwärmung Vermittels Durchstromender Medien" by A. Anzelius. A. J. Willmott, December 1976.
- [YCS 3] Report on the Central Computing Service, Computer Centre, August 1975.
- [YCS 4] Structured Real-Time Programming. I. C. Pyle, March 1977.
- [YCS 5] Experience with the Programming Language Modula. J. Holden & I. C. Wand. June 1977.
- [YCS 6] Heat Transfer Coefficient Correlations for Thermal Regenerator Calculations - Transient Response. A. Burns. June 1977.
- [YCS 7] Toward More Effective Programming. C. Runciman, January 1978.
- [YCS 8] Annual Report to SRC on Project B/RG 76970: Real Time Programming Languages for Industrial and Process Control. I. C. Pyle, January 1978.
- [YCS 9] The Recuperator Analogy for the Transient Performance of Thermal Regenerators. A. J. Willmott & A. Burns, February 1978.
- [YCS 10] Annual Computing Service Report 1976/77. Computing Service, February 1978.
- [YCS 11] The Adaptable Terminal: A User Adjustable Man-Computer Interface. R. C. Thomas & I. C. Pyle, March 1978.
- [YCS 12] Methods for the Design of Control Software. I. C. Pyle, April 1978.
- [YCS 13] Functional Specification of the Modula Compiler. I. D. Cottam, June 1978.
- [YCS 14] MCODE. I. C. Wand & J. Holden, August 1978.
- [YCS 15] Dynamic Resource Allocation and Supervision with the Programming Language Modula. I. C. Wand, August 1978.
- [YCS 16] An Assessment of Modula. J. Holden and I. C. Wand, November 1978.
- [YCS 17] Modula Distribution and Promulgation 1978. I. C. Wand, January 1979.
- [YCS 18] Annual Computing Service Report 1977/78. Computing Service, January 1979.
- [YCS 19] Reflections About Computer Science. N. Wirth, February 1979.
- [YCS 20] Functional Specification of the Modula Compiler Release 2. I. D. Cottam, March 1979.
- [YCS 21] Final Report to SRC on Project B/RG 76970: Real Time Programming Languages for Industrial & Scientific Process Control. I. C. Pyle & I. C. Wand, May 1979.
- [YCS 22] Proceedings of the "IRONMAN" Languages Seminar. W. Freeman(Ed.), June 1979.
- [YCS 23] Proceedings of Networkshop 4. J. D. Service, July 1979.
- [YCS 24] The Use of Walsh Functions in Data Multiplexing: A Survey. S. M. Gill, August 1979.
- [YCS 25] Indirect Computer Programming. T. J. Froggatt, August 1979.
- [YCS 26] Modula on the INTEL 8080 Microprocessor. C. J. Nolan, N. P. Hawkins, I. C. Pyle & I. C. Wand, October 1979.
- [YCS 27] Sequentially Masked m-Sequences for Noise Generation; I: Simple Sequences up to $n=24$, W. Freeman & S. M. Hebden, November 1979.
- [YCS 28] Annual Computing Service Report 1978/79. Computing Service, December 1979.
- [YCS 29] Towards a Run-Time System for Ada. I. C. Wand & J. Holden, December 1979.
- [YCS 30] Explicit Terminators for Language Constructs. D. G. Burnett-Hall & I. C. Pyle, January 1980.
- [YCS 31] Review of Standards in Software for Real-Time Systems. I. C. Pyle, February 1980.
- [YCS 32] Sequentially Masked m-Sequences; II: Higher Order Entropies. W. Freeman, February 1980.
- [YCS 33] Functional Specification of the Modula Compiler Release 3. I. D. Cottam, April 1980.

- [YCS 34] Modula and a Vision Laboratory. C. Runciman, August 1980.
- [YCS 35] Resolving Overloaded Expressions in Ada. C. Runciman, October 1980.
- [YCS 36] On Functional Relationships Between Different Methods of Contra-Flow Thermal Regenerator Operation. S. A. H. Smith, November 1980.
- [YCS 37] A System of Asynchronous Communication with Protection Against Noise. W. Freeman & S. M. Hebden, December 1980.
- [YCS 38] Annual Computing Service Report 1979/1980. Computing Service.
- [YCS 39] Ada Workbench Compiler Project 1980. C. W. Johnson, I. C. Pyle, C. Runciman, I. Walker & I. C. Wand, January 1981.
- [YCS 40] Distributed UNIX Project 1980. G. M. Tomlinson, I. C. Wand & A. J. Wellings, December 1980.
- [YCS 41] Sequentially Masked m-Sequences; III: Distribution and Extreme Values of Gap Lengths. W. Freeman, S. M. Hebden & B. J. Wright, March 1981.
- [YCS 42] Research in Progress 1981. W. Freeman, P. K. Lala, K. C. Mander, I. C. Pyle, J. D. Service, C. J. Tully, I. C. Wand & A. J. Willmott, May 1981.
- [YCS 43] Towards Specifying an Information System. I. C. Pyle, June 1981.
- [YCS 44] An Ada View of Specification and Design. K. C. Mander, November 1981.
- [YCS 45] Sequentially Masked m-Sequences; IV: An Apparatus for the Investigation of Sequences. W. Freeman, S. M. Hebden & R. Pack, December 1981.
- [YCS 46] Information Technology 1831-1981 : An Exhibition in the J. B. Morrell Library. W. Freeman, December 1981.
- [YCS 47] Distributed UNIX Project 1981. I. C. Wand, A. J. Wellings & G. M. Tomlinson, January 1982.
- [YCS 48] Ada Workbench Compiler Project 1981. J. S. Briggs, C. H. Forsyth, C. W. Johnson, J. A. Murdie, I. C. Pyle, C. Runciman, I. Walker, I. C. Wand & A. J. Williams, January 1982.
- [YCS 49] Annual Computing Service Report 1980/81. Computing Service, March 1982.
- [YCS 50] The Modelling and Regulation of a Regenerator System Under Varying Operating Conditions. A. J. Willmott & A. E. Wraith, March 1982.
- [YCS 51] Computing at York 1976-1981. Computing Service, April 1982.
- [YCS 52] A Model Lift Shaft as a Test-Bed for Student's Real-Time Programming Experiments. W. Freeman, M. Baslington & R. Pack, May 1982.
- [YCS 53] Sequentially Masked m-Sequences; V: Models for the Distribution of Gap Lengths. W. Freeman & J. Roberts, June 1982.
- [YCS 54] Functional Specification of the York Ada Workbench Compiler Release 0. J. A. Murdie, October 1982.
- [YCS 55] Non-symmetric Thermal Regenerators. A. J. Willmott & R. J. B. Hilton, November 1982.
- [YCS 56] Gill - An Interpretive Language for Small Systems. I. Walker, November 1982.
- [YCS 57] Ada Programming Course - Lecture Notes. C. H. Forsyth, C. Johnson, C. Runciman, I. C. Pyle & I. C. Wand, December 1982.
- [YCS 58] PULSE Project 1982. D. Keeffe, G. Tomlinson, I. C. Wand & A. J. Wellings, December 1982.
- [YCS 59] Ada Workbench Compiler Project 1982. J. S. Briggs, C. H. Forsyth, C. W. Johnson, M. R. Manning, J. A. Murdie, I. C. Pyle, C. Runciman, I. C. Wand & A. J. Williams, February 1982.
- [YCS 60] Research in Progress 1983. W. Freeman, P. K. Lala, K. C. Mander, C. Runciman, H. W. Thimbleby, I. C. Pyle, C. J. Tully, I. C. Wand & A. J. Willmott, March 1983.
- [YCS 61] Notes on a System Specification Method. K. C. Mander, March 1983.
- [YCS 62] Functional Specification of the York Ada Workbench Compiler Release 1. J. A. Murdie, December 1983.
- [YCS 63] Using Ada for Specification of Requirements and Design. I. C. Pyle, December 1983.
- [YCS 64] Dimensionless Parameters for Thermal Regenerators. A. J. Willmott, November 1983.
- [YCS 65] Advances in Single-Phase Regenerator Design Theory. A. J. Willmott, November 1983.

- [YCS 66] Improved Lumped Heat Transfer Coefficients for Thermal Regenerators. R. J. B. Hilton & A. J. Willmott, December 1983.
- [YCS 67] PULSE Project 1983. D. Keefe, G. M. Tomlinson, I. C. Wand & A. J. Wellings, February 1984.
- [YCS 68] Lumped Heat Transfer Coefficients for Reversing Regenerator. R. J. B. Hilton, February 1984.
- [YCS 69] An Educational Bit-Slice Microprogrammable Tutor. G. Morgan, R. Pack & P. K. Lala, April 1984.
- [YCS 70] Filter Synthesis Using FIDES. M. H. N. Potok & I. D. Benest, September 1983.
- [YCS 71] The York UNIX-X25 Communications Software System. K. S. Ruttle, S. J. Smith & I. C. Wand, August 1984.
- [YCS 72] Pulse Project 1984. I. C. Wand, D. Keefe, G. M. Tomlinson & A. J. Wellings, August 1984.
- [YCS 73] Failure in the Technical User-Interface Design Process. H. W. Thimbleby, August 1984.
- [YCS 74] Experiences of "Literate Programming" using CWEB (a variant of Knuth's WEB). H. W. Thimbleby, August 1984.
- [YCS 75] Two Working Papers on Formalizing Interactive System Design. A. Dix, M. D. Harrison, C. Runciman & H. W. Thimbleby, June 1985.
- [YCS 76] An Inversion Technique for Functional Programs. C. Runciman, July 1985.
- [YCS 77] Using Ada for the Analysis of Protocols. I. C. Pyle, September 1985.
- [YCS 78] A Review of Ada Tasking. A. Burns, A. M. Lister & A. J. Wellings, September 1985.
- [YCS 79] Adapting SECD and Combinator Reduction Machines to Display Snapshots of Functional Computations. I. Toyn & C. Runciman, November 1985.
- [YCS 80] Equal Opportunity Interactive Systems. C. Runciman & H. W. Thimbleby, January 1986.
- [YCS 81] A Robust Method for Regenerative Heat Exchanger Calculations. A. Hill & A. J. Willmott. January 1986.
- [YCS 82] An Interpretive Code for OCCAM. A. J. Fisher. April 1986.
- [YCS 83] Research in Progress. M. D. Harrison (Ed.) April 1986.
- [YCS 84] On the Recursive Generation of Intransitive Verb Phrases and Subordinate Time Relativisation. R. I. Bainbridge. September 1986.
- [YCS 85] The Modelling of Maldistribution within Thermal Regenerators. M. E. Penney & A. J. Willmott. November 1986.
- [YCS 86] A Simulator Program for Evaluating and Improving the Nottingham Muse Architecture. N. K. Barrett, D. F. Brailsford & R. J. Duckworth. November 1986.
- [YCS 87] York Ada Workbench Compiler System, Release 2. J. R. Firth, C. H. Forsyth, L. Tsao, K. S. Walker & I. C. Wand. February 1987.
- [YCS 88] Heat Exchanger Design, Methodology & Simulation. M. P. Henry, A. Hill & A. J. Willmott. March 1987.
- [YCS 89] The Headingly Cricket Scoring System. J. S. Briggs, March 1987.
- [YCS 90] Delaying Commitment, H. W. Thimbleby, August 1987.
- [YCS 91] Programming and Debugging Distributed Target Systems, A. D. Hutcheon, D. S. Snowden & A. J. Wellings. August 1987.
- [YCS 92] An Author's cross-referencer, H. W. Thimbleby, August 1987.
- [YCS 93] The Notion of Priority in Real-Time Programming Languages, A Burns, A J Wellings, October 1987.
- [YCS 94] The ADAM associative memory, J Austin, T J Stonham, October 1987
- [YCS 95] Practical parsing of generalised phrase structured grammars, A J Fisher, November 1987.
- [YCS 96] Assurance in High Integrity Software, J A McDermid, November 1987.
- [YCS 97] York Ada Compiler Release 3 (SUN/UNIX) - User Guide, J R Firth, C H Forsyth, I C Wand, January 1988.

- [YCS 98] A Declarative Approach to Intensional Logic Reduction, R I Bainbridge, February 1988
- [YCS 99] The Life and Times of ded, text display editor, R Bomat, H W Thimbleby, February 1988.
- [YCS 100] Towards Models that Clarify the Manipulability of Interactive Systems, M D Harrison, A J Dix, February 1988.
- [YCS 101] Robust and Very Rapid Thermal Regenerator Calculations, A Hill, A J Willmott, February 1988
- [YCS 102] Image Pre-Processing with a Generalised Grey Scale N Tuple Operator, J Austin, March 1988
- [YCS 103] Inverted Montague Syntax, R I Bainbridge, August 1988
- [YCS 104] Figures of merit for interconnection strategies in large homogeneous networks - Part I, W Freeman, August 1988
- [YCS 105] Accurate and Rapid Thermal Regenerator Calculations, A Hill, A J Willmott, August 1988
- [YCS 106] Using Sentinels in Insert Sort, H W Thimbleby, August 1988
- [YCS 107] Finding fixed points in non-trivial domains: Proofs of pending analysis and related algorithms, A J Dix, August 1988
- [YCS 108] The Alpha Text Formatting System, A J Fisher, November 1988
- [YCS 109] Overview and User Manual for Doubleview, S J Holmes, January 1989
- [YCS 110] Reliable Computing Systems - A Review, R F Stone, January 1989
- [YCS 111] Security Models and Enterprise Models, J E Dobson, J A McDermid, January 1989
- [YCS 112] Some Issues in The Formal Design of Redundant Systems, J A McDermid, G Morgan, January 1989
- [YCS 113] Database Support for Complex Objects, A W Brown, January 1989
- [YCS 114] Variable-Order/Variable-Stepsize Rosenbrock-type Algorithm for solving stiff systems of ODEs, H S M Zedan, February 1989.
- [YCS 115] The Virtual Node Approach to Programming Distributed Embedded Systems in Ada, A D Hutcheon, A J Wellings, February 1989.
- [YCS 116] Priority Inheritance and Message Passing Communication, A Burns, A J Wellings, February 1989.
- [YCS 117] Ada9X - The Need for Change, A Burns, A J Wellings, February 1989.
- [YCS 118] User Modelling with a Neural System, R Beale, J Finlay, March 1989.
- [YCS 119] Towards the Machine Translation of Swedish to English, E S Hill, March 1989.
- [YCS 120] Soundtrack 2 User Manual (Version 2), A D N Edwards, April 1989.