Screenreaders, magnifiers and other ways of using computers

Dr. Alasdair King, 2011. I wrote this many years ago, and it is dated: in particular it fails to predict how many blind people would switch to iPads and iPhones, using VoiceOver and touch instead of a screenreader and keyboard on a Windows machine. But it's still okay as a backgrounder for desktop/Windows usage.

People who are blind or have a visual impairment have problems using computers, since computers are designed for mouse users who can see the screen. But computers offer so very much to blind people that it is worth using computers anyway even if it is hard.

The solution is to use *access software*: a screenreader, or a magnifier, or a non-standard application, or what native capacity there is in the operating system. Most blind people use some combination of all these strategies, knowingly or not. This chapter describes these categories of access software for blind people; how screenreaders work; the state-of-the-art in access software; and a review of some of the big challenges facing access software.

1 Categories of access software for blind people (dedicated software versus access software)

Access software for blind people can be categorized into three categories: *screenreader/magnifier software*, which lets you use the standard computer applications like Word or a web browser; *dedicated software*, which means you use a specially-designed bit of software, and *adaption and customization*, which means you take advantage of whatever you can do in the operating system or existing applications to make them work better.

Screenreaders and Magnifiers (e.g. JAWS)

A *screenreader* works out what is happening on the screen and sends its output either to synthesised speech or, far less commonly, to a Braille display (Evans & Blenkhorn, 2003) (Braille has some considerable advantages, is politically contentious, and is very rare.) Generally, you use a screenreader if you cannot see anything at all. A *magnifier* magnifies the screen so that if you have some vision you can use the mouse (Blenkhorn *et al*, 2003). Generally, you use a magnifier if you have some sight. In fact, most screenreader software provides both screenreading and magnification, because most blind people have some sight, and being able to peer at the screen and make an attempt at working out what is going on can be very useful at times. Traditionally magnifiers did not have an offscreen model but screenreaders did, but the distinction has blurred.

Using a screenreader effectively is hard. Screenreaders are trying to communicate quite heterogeneous user interfaces: Microsoft Word is quite different from a web page. The user cannot scan around the screen to try to spot the right function or the content he wants to read, he has to explore slowly and carefully. To compensate and speed things up screenreaders have acquired dozens of shortcut keys which are useful for power users but mean lots more to learn. And many things will not work at all (buttons with images but no text, for example.)

It would therefore seem natural that magnification is better for anyone who can see the screen to some degree. However, using magnification is really hard, since it is easy to get confused with the location of the mouse pointer and where one is on the screen, especially at high levels of magnification, and the quality of the display is often poor because of the pixilation of the magnified bitmap display (Blenkhorn *et al*., 2003).

Dedicated Software – self-contained alternatives to standard applications (e.g. Guide)

Screenreaders are powerful general-purpose access software solutions suitable for users who seek to use their machine just as a sighted person does. But sighted users, the success of the iPad suggests, seek to use their machines for a relatively limited set of uses – web browsing (though that means so much more, now), email, chat, video and music, a little occasional word processing. Why can't blind people access a similarly limited but customized system that suits their particular modality? If a non-PC interface is good enough for sighted people, why not blind users?

These systems do exist. The main commercial PC system is Guide from Dolphin in the UK. It allows a user to do email, web browsing and other essential applications with self-voicing menus. This is not dissimilar to some of the expensive dedicated portable notetaker or notebook devices that are sold with Braille input and output interfaces, or speech, and are often based on Windows machines: dedicated bookreaders might also be regarded as simplified user interface machines dedicated to blind users.

The situation is less clear-cut and more positive on non-PC platforms. The Mobile Accessibility for Android from screenreader.net replaces the essential features of a tablet or phone with a fully-accessible customized interface. The iPhone, or iPad especially, allows you to browse the web, listen to the radio, read and write email and the other essentials of normal computer use, with a self-voicing application that is both fashionable and works well. If blind people did not have to work in offices with standard desktop applications like Excel, then the iPad might have "solved" the screenreader problem for good (cost of the device aside – although with the cost of the access software taken into account the iPad is actually inexpensive.)

This leaves the option of using non-standard applications: rather than struggling to use applications designed primarily for mouse and screen users, why not employ applications that are specifically written for blind screenreader users? These can be designed to work well with any screenreader – for example, always placing a caret in a text area, keeping the number of controls on a form to a minimum, and including access and shortcut keys. Or they can replace the screenreader for that application, self-voicing: generating their own speech (or in theory Braille). The WebbIE suite of applications (web browser, RSS news reader, podcatcher, eBook reader) for Windows falls into the former category. The SpeakOn application includes a similar set of functions (play CDs, live radio, BBC stations from the last week) but also synthesises its own speech (and has no visible user interface!) Both are free projects. Less all-encompassing products include Qwitter, an accessible Twitter client, and PowerTalk for reading PowerPoint presentations using synthesized speech: when few screenreaders could handle the web, IBM's HomePage Reader filled the gap (Asakawa & Itoh, 1998) In five years many of these will have stopped being developed, and many more will have appeared: this is a rich but temporary ecosystem.

However, many blind users prefer to use standard common applications over non-standard applications. There are good reasons: skills and techniques learned from using normal, fairly inaccessible applications may be more transferable to novel situations when they are encountered. Non-standard applications are often free, or not the main commercial product for a company, and therefore lack support and often lag behind in features. Simplified user interfaces are great, but often functions are cut. But most importantly is perhaps the sentiment that non-standard applications present a crippled, fake experience, and deny blind screenreader users the authentic status of being users of the same applications as "normal" people. Even if their screenreader employs three non-visual accessibility APIs and a script and an off-screen model to communicate to the end user, still the user feels that he is using the standard application directly.

Adaption and customisation of existing systems (i.e. native OS/app accessibility settings)

Everyone approaches applications with their own set of needs and wants. People with disabilities are no different, but they add “works with my disability” to their criteria. Users gravitate where possible to applications that work with their access technology. “Which BitTorrent client should I use?” can be answered with “the one that works with your screenreader.” It is harder when the application is one that cannot be replaced, like Microsoft Outlook in corporate user. Users themselves, or carers or experts, swap tips on “accessible” applications on email groups or websites. Simple tools of limited utility to people without disabilities are often pressed into service. For example, TextAloud from NextUp provides text-to-speech facilities, ostensibly for anyone who wants to create audio from text. In practice this provides both a simple, low-cost way for blind people to generate audio for listening to later, and also a set of good speech synthesizers that can be employed in other applications. These are not applications intended for people with disabilities, but they have a strong following in these communities because of their longevity, simple interfaces, and low cost.

There are also applications that are of particular use to people with disabilities, such as OCR programs or speech recognition programs. Adding these to an otherwise standard build can make all the difference.

Users can also employ the built-in accessibility features of operating systems or applications (Kurniawan *et al*, 2006). These have gradually improved over the last fifteen years, with notable milestones including the Microsoft developments from 2000 of the Narrator screenreader, a magnifier, on-screen keyboards and system customization; the free VoiceOver screenreader in the Apple operating system; and the touch-based screenreader in the iPhone and iPad. The story is quite different on Windows desktop machines, where the operating system access software (Narrator, Magnifier) has traditionally been very limited compared to the commercial access software (Narrator supports neither Internet Explorer nor Microsoft Word, for example) and has been quite unsuccessful, versus Apple machines, especially the iPad and iPhone, where VoiceOver has been more successful, helped by the lack of commercial competition and the greater homogeneity of the Apple environment.

2 How Screenreaders Work

It's important to have an idea about how screenreaders work to understand their limitations and the politics and technology behind them.

Screenreaders communicate to the blind user what is happening on the screen. To do this they hook into keyboard input and hook into system events, such as new windows or menus being activated. This allows them to echo back what the user types and make the user aware of changes to the screen, such as pop-up windows or web browsers loading a new page. They also have to communicate the contents of windows – user interfaces and the contents of documents like web pages or Word documents. So the key techniques for screenreaders are to communicate with other applications to identify when they change and what they are now displaying.

Crudely, there are three ways for a screenreader to access another application and communicate its contents to the user. First, through an *offscreen model*. Second, through an application-specific *application API*. And third, through an operating system *accessibility API*.

The offscreen model could be regarded as the “traditional” screenreader mechanism (Evans & Blenkhorn, 2003). Every application has to draw UI components, text, and graphics on the screen, which usually means they call low-level operating system routines like DrawText on Windows. The operating system updates the video display driver, and the text and windows appear on the screen. An offscreen model is built by intercepting these calls and, if possible, querying the operating system for the current state of the screen display. The screenreader can then communicate the contents of the offscreen model to the user as appropriate. This approach can be immensely powerful: the screenreader has access to the whole visible contents of the screen, as text, and can use sophisticated heuristics to do things like work out the caption for a text field by checking for text above or to the left of it (or the right in Arabic or Hebrew), or watch for the redrawing of the caret graphic to spot focus changes. However, there are three drawbacks to the offscreen model: first, such low-level interaction with the operating system can lead to system instability; second, they are really hard to build and maintain; and third, some applications use their own internal mechanisms to render text and graphics and hence are blank voids to the offscreen model (Java applications using Swing are a classic example, but DirectDraw applications like Internet Explorer 9 or Firefox 7 are increasingly common.) For these reasons newer screenreaders like NVDA have elected to go with application-specific and operating system APIs.

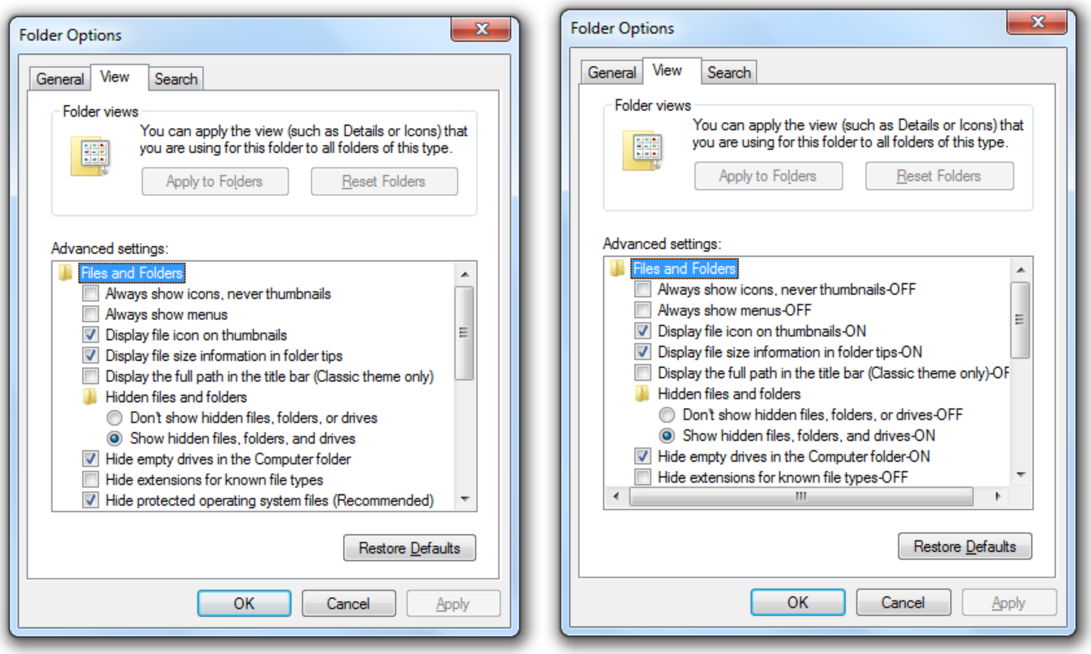
Application-specific APIs allow a screenreader developer to write code to connect to and query an application about its current state. For example, on Windows machines a developer can use COM to talk to Microsoft Word to find out the current line, the formatting, the caret position, and launch the spellcheck - all through the Word Object Model API. These APIs are powerful but application-specific: a developer must write code to support each one. And the API content does not necessarily reflect the application experience for a blind user – menus activity is not generally reflected in the API, for example – so the application cannot be completely supported through the API alone.

Operating system accessibility APIs were developed starting with Microsoft Active Accessibility (MSAA) in 1998. An application that complies with an accessibility API guarantees that screenreaders and other access software will be provided with information on the current control, the window, activity and anything else that is needed. For example, it might say that all buttons will announce themselves as type “button” and provide the caption and whether the button has, or can take the focus. Now, no matter how the button is actually drawn on the screen and made to operate – whether Windows API, .Net, Qt, GTK, Java, or any other graphical toolkit or underlying programming language – a button looks and works the same to a screenreader. So the screenreader no longer has to maintain an expensive and unreliable offscreen model: instead it can interact with this parallel representation of the program. The accessibility API draws the user interface to a screenreader just as the conventional code draws the user interface to the screen for a mouse user. In theory accessibility APIs are the best solution for access software operation.

There are three problems, however, with accessibility APIs. First, the interfaces they represent are complex and designed for visual and mouse use: these alternative representations are not equal in terms of task profile (Bennett & Edwards, 1998). If the user presses a button that makes text in another control bold, the sighted user can see the change and understand it. A screenreader could announce that the button has been pressed, or announce the text that has been bolded, or play a tone and move the user focus to the selected text – it is not trivial to communicate the change nor for the user to understand it. The accessibility API may announce different events – button pressed, text changed, selection changed – depending on the implementing application. A naïve screenreader relying only on the accessibility API will still encounter different behaviour and characteristics of common controls, where the similarities in function are obvious to a sighted user – between Microsoft Word and OpenOffice, for example. Screenreader developers must still write application-specific code despite the accessibility APIs.

The more significant problem with accessibility APIs is that, like many standards, different people use a different standard. Sometimes there are political reasons: it is unsurprising that Sun never supported the Microsoft MSAA in Java applications written for Windows, or that Microsoft Office on the Apple Mac does not support the Apple Mac OSX Accessibility Model. Sometimes there engineering reasons: the simple MSAA API cannot handle the complexity of the Microsoft Word document area, so Microsoft has had to introduce a new accessibility API, User Interface Automation (UIA). Sometimes there are pragmatic reasons: accessibility API support usually comes later on in the development of an application, maybe the second or third release, and lags behind commercially-important user interface developments. So while, in theory, accessibility APIs reduce screenreader development requirements and obviate the need for offscreen models, in practice the workload is still significant. To support a reasonable "minimum specification" for a screenreader on Windows, supporting Windows Explorer, Notepad, Microsoft Word, Internet Explorer, Firefox, and Java applications on a Windows machine you would need to handle MSAA, the Microsoft Word Document Object Model, the Internet Explorer Document Object Model and/or UIA, IAccessible2, and the Java Access Bridge. In this area, homogeneity and monopoly is a great boon: if everyone still used exclusively Internet Explorer 6 and Microsoft Office then the work of the screenreader vendors would be greatly eased!

Finally, the existence of a defined accessibility API for a technology does not mean that complying with it is automatic or trivial. It is not a panacea. Developers must still attend to the use of their application by a screenreader user (e.g. can you tab around the whole user interface with the keyboard, or do you need to use the mouse?) and work around any problems encountered in user testing: this work is often not done, although automated tools to identify accessibility problems like Accessibility Checker for Windows can help. The requirements of complying with the API may be at odds with other requirements, such as the appearance or security concerns. An interesting example is Windows Explorer. The Folder Options dialog contains a list of checkbox options, which one would assume would work well with MSAA or UIA by default. However, Microsoft contains a “screenreader flag”, a global internal setting which when set indicates to applications that they should make themselves accessible if they were not before. If this is set then the checkbox list in Windows Explorer suddenly gains explicit text in the label for each textbox that shows the checked state. Clearly either the MSAA/UIA interface for this dialog could not be correctly implemented or testing indicated that it could not be used. Such design compromises are both necessary and infrequently the top priority for application developers.


Folder Options

General | View | Search

Folder views

You can apply the view (such as Details or Icons) that you are using for this folder to all folders of this type.

Apply to Folders | Reset Folders

Advanced settings:

Files and Folders
- ☐ Always show icons, never thumbnails
- ☐ Always show menus
- ☑ Display file icon on thumbnails
- ☑ Display file size information in folder tips
- ☐ Display the full path in the title bar (Classic theme only)
- Hidden files and folders
  - ○ Don't show hidden files, folders, or drives
  - ● Show hidden files, folders, and drives
- ☑ Hide empty drives in the Computer folder
- ☐ Hide extensions for known file types
- ☑ Hide protected operating system files (Recommended)

Restore Defaults

OK | Cancel | Apply

Folder Options

General | View | Search

Folder views

You can apply the view (such as Details or Icons) that you are using for this folder to all folders of this type.

Apply to Folders | Reset Folders

Advanced settings:

Files and Folders
- ☐ Always show icons, never thumbnails-OFF
- ☐ Always show menus-OFF
- ☑ Display file icon on thumbnails-ON
- ☑ Display file size information in folder tips-ON
- ☐ Display the full path in the title bar (Classic theme only)-OF
- Hidden files and folders
  - ○ Don't show hidden files, folders, or drives-OFF
  - ● Show hidden files, folders, and drives-ON
- ☑ Hide empty drives in the Computer folder-ON
- ☐ Hide extensions for known file types-OFF

Restore Defaults

OK | Cancel | Apply

Figure 1: Windows Explorer with the screenreader flag off (left) and on (right) showing how text is added to support screenreaders.

Here, then, is how a new screenreader without an offscreen model might enable a user to work with Microsoft Word:

1. The screenreader uses the Windows API to identify the foreground window as Word.
2. The screenreader uses MSAA to get the currently-selected control in Word. It turns out to be the document area (where you type, as opposed to the ribbon control area).
3. The screenreader queries the Word Document Object Model to get the caret position and the surrounding text.
4. The screenreader hooks a low-level keypress by the user indicating that the current line in Word should be read aloud.
5. The screenreader queries the Word DOM and gets the text.
6. The screenreader sends the text to the speech synthesizer or the Braille display.

Screenreaders tie together a profusion of interfaces and techniques to try to present a coherent audio/Braille user interface for a blind user. It is a difficult task, made more so by the incompatible standards employed and the rapid changes in applications and operating systems common on modern machines.

How the desktop user experiences a screenreader

A desktop screenreader user is using a non-visual modality – touch or more usually speech. This means that he cannot quickly grasp the contents of the screen – what's happening? How many buttons? What can I click on right now? – but must explore the current application to find buttons, controls and other features. It is much more a process of learning how a particular application works with the user's screenreader than is the more obvious mouse/screen interface. Screenreader users therefore need to build accurate mental models of applications, and to use them efficiently they often employ dozens of keyboard shortcut keys. This is cognitively challenging, for a start. It needs good keyboard skills: multiple-key combinations can be a challenging combination of keys to hit and most blind people cannot touch-type. You cannot operate some applications with the keyboard alone – your focus can get trapped in an input field in a badly-designed web page, for example, and you cannot tab out of it. And changes in applications – for example, the introduction of the ribbon in Microsoft Office 2007 – can mean the loss of painstakingly-acquired knowledge of how to do things or even the complete loss of the ability to use an application (if the changes are major.) Blind screenreader users therefore often hate change even more than most users, and are loath to migrate from one screenreader to another, or even to update their software, operating system or screenreader if they can avoid it. Using a full screenreader is therefore difficult and for some people too unrewarding: they stop using a computer at all, or rely on squinting or magnification. Using a screenreader is not the same as using the screen and mouse, no matter how skilled a screenreader user becomes. Their task profile, what is easy and what is not, and the fundamental interfaces they are using, keyboard and API rather than mouse and window, means their user experience is very different – even if they do not know it.

The fact is that the dominant paradigm for screenreader use is functional completeness: that a screenreader user should have access to all the information on the screen that a sighted user has. This is sometimes an idea more strongly-held by sighted people than actual screenreader users, who are forced to be more pragmatic. In consequence the idea of eliding content to improve aspects of usability is regarded with suspicion. Rather than the concept of "maximum output for minimum time" (Blenkhorn & Evans, 2001) – truncating or simplifying user interfaces to permit screenreader users to operate them faster, at the cost of some theoretical "full" understanding – the philosophy is to make sure that every element is made available somehow, if only the user has the patience. This is not unreasonable given the history of incomplete and outdated "accessible versions" of web pages and other products. But this is not an accurate task analysis for screenreader users employing a different modality from sighted mouse users: prioritizing completeness over speed of operation may reduce operational efficiency to the point where the user may simply give up on the screenreader (King, 2007).

How a screenreader vendor experiences the desktop

Even if every application fully complied with the operating system accessibility API, the nuances and complexities of real-world usage in anything beyond the most trivial user interface means that the chances of a screenreader working efficiently “out of the box” with a given application are small. Decisions that have no impact on the standard sighted mouse user are very important to screenreader users. When you open an email in your email client, where is focus placed first – the From field, the Subject field, the body of the email? Is there a caret? Are the From/Subject fields fixed labels, or are they text areas? Even if a developer attempts to comply with the accessibility API requirements she is unlikely to provide the same quality of user experience to a screenreader user. This requires that a screenreader vendor (or a third party) writes code (or a script) to handle the application. For example, when an email is opened the screenreader may detect this event, read the From/Subject fields, then instruct the application to place the focus and a caret in the email body. This can be very effective, but the burden of work is thus placed on the screenreader vendor, and there are hundreds of applications in common use and thousands upon thousands in general use on each platform. Each represents an investment of time by the screenreader vendor. Unless a competitor is claiming better support for a popular application, or your existing customers are insisting on support, you will have little incentive to do more work to support yet another application, now and in future, and especially with ever-faster-changing user interfaces. What this means is that with any given fixed resource of developers, a screenreader vendor can support and maintain only a limited set of applications. With bigger commercial products able to support more developers (JAWS) this set can be bigger. With fewer developers (NVDA, other screenreaders) it will be smaller. So if you need email, word processing, web browsing and a media player, and don't mind what application you use to achieve these ends, then anything from VoiceOver to JAWS will suit you, and you can use one of the free screenreaders: home users like this have many possibilities. But if you need to use a particular specialised application for work then you will probably need to use JAWS on a Windows machine – and pay for the latest version too.

3 State of the art of Access Software (screenreaders and magnifiers)

Just as there are other office suites apart from Microsoft Office but relatively few people use them, there are other screenreaders apart from Freedom Scientific's JAWS. JAWS dominates the screenreader market, both commercially and, via piracy, uncommercially. It is the gold standard against which everything else is measured: alternative screenreaders must work like JAWS, applications must work with JAWS, documents must be formatted to be usable in JAWS, and HTML accessibility standards must be based on how JAWS operates. Freedom Scientific provides a 40-minute free demo version, which although strictly forbidden is widely used by sighted people to test their application or website. Cracked versions are widely available and used, especially in the developing world. With the ubiquity of JAWS in mind, here is the current state of screenreader and magnification technology.

Access Software for Microsoft Windows

Windows remains the main platform for accessible desktop machines because it has the most access software, is the most widely-used and available operating system, and it is used in work. Access software in its current form also developed in the late 90s/early 2000s, when Windows was almost unchallenged, and since no-one ever likes updating or changing their access software, much access software is still Windows-based.

Native applications and APIs in Windows

Windows Narrator has provided a basic screenreader since Windows 2000 and is essentially unaltered in function since then: it does not support, for example, Microsoft Word or Internet Explorer (or any standard web browser) although the Developer Preview version for Windows 8 Metro has been beefed up. Regulatory, commercial and practical concerns have probably kept Microsoft from building a capable screenreader – that may change in Windows 8. Windows Magnifier provides a basic magnifier, and has been greatly improved in Windows 7, but it lacks smoothing: this was to have been provided by the graphical display system used in Windows Presentation Framework, but this ran into the delayed Windows Vista problems and has never been supported: in theory, the next version of Windows after XP was to launch in 2005 with vector graphics, which of course can scale perfectly in magnification. This eventually arrived in Windows Presentation Foundation, so for a brief period smooth magnification worked. Then WPF was changed to remove this functionality to meet requirements in the underlying operating system. This may change again in Windows 8 in 2012 with the move to HTML applications rendered using DirectDraw and accelerated graphics.

The operating system also allows for font sizes and colors and the screen resolution to be changed to make text and graphics larger and easier to see. Microsoft's own applications now generally support this well, but third-party applications usually fail, leading to (for example) overlapping text in list boxes or dialogs that will not fit on the screen. Screenreaders usually enable these changes when they are installed, and AT professionals and some organizations proselytize about these features, but their relative complexity and inconsistent application makes them less popular than they might be. For example, you can make text bigger in Microsoft Outlook, but Outlook will not notice that you can no longer make use of the preview pane and turn it off or place it below your Inbox rather than beside it. The cascade of configuration and changes you need to make lead to a level of complexity such that only experienced volunteers for blind charities – usually retired IT workers – employ these techniques, being technical, poor and having the time. (There is a general problem with access software: first, you have to realize you have a problem. Second, you have to realize there are solutions. Third, you have to be able to get them and operate them.)

Microsoft also provides two Windows operating system accessibility APIs. The first, Microsoft Active Accessibility (MSAA), was developed for Windows 2000 (it worked on Windows '98 with an update). The second, User Interface Automation (UIA) appeared in Windows Vista. Crudely put, MSAA is good for buttons, checkboxes, and plain text, and is accessible through COM and the Windows API, while UIA adds support for rich text areas, like the document area in Microsoft Word or web pages, and is accessible through the .Net framework. UIA always falls back to MSAA, so in practice developers use the MSAA interface and various non-accessibility COM interfaces for complex applications like Microsoft Word (Word DOM) or Internet Explorer (the W3C DOM or DHTML DOM). Third-party accessibility APIs on Windows include IAccessible2 API (Firefox) and the Java Access Bridge (Java).

The summary for Microsoft Windows might be: good accessibility APIs, the dominant operating system for accessibility for PCs, and a wide range of access software software: but a heterogeneous and fast-changing environment of programs leaves access software vendors struggling to make their screenreaders work everywhere on the latest products. If Internet Explorer 9 is coming out in a new version every year you have to run to stay still.

Commercial Access Software

Windows provides the widest range of commercial access software for blind users. The ubiquitous JAWS from Freedom Scientific is dominant as a screenreader and is the *de facto* standard. Other important screenreaders include WindowEyes from GW Micro and SuperNova from Dolphin in the UK. Most screenreaders now can be *scripted*. That is, third-party developers can make and sell sets of instructions that enable screenreaders to operate with and support other applications. This widens the range of supported applications and provides an ecosystem of people interested in developing for the platform. For example, Jawbone is a script that enables JAWS to operate with Nuance's Dragon NaturallySpeaking, a speech recognition program. Without it the utility of Dragon is greatly reduced: with it, there is yet another reason to use JAWS.

These products all fit the model of "standard computer plus software." However, many people use Windows machines with expensive hardware Braille displays, especially in Germany and other countries with good state support through medical insurance. The manufacturers of these Braille displays may use JAWS – all the commercial screenreader products are able to drive Braille displays – but often use their own screenreader software or re-use products under licence from commercial vendors. These users are a tiny minority – very few blind people know any Braille, and even fewer can afford a Braille display – but they are often influential advocates for technology and blind people.

Free software

For thirty years, from the 1980s onwards, screenreaders and other access software were commercially-available and usually expensive. There are now several free pieces of software.

The Thunder screenreader from screenreader.net launched in 2006 and is freeware – no cost, but copyright. It is aimed at novice blind users. However, its failure until recently to work with a web browser directly (it relied until 2010 on the WebbIE text browser), its copyright status, and its limited feature set compared to JAWS have limited its support amongst advocates.

The NVDA screenreader, by contrast, is free software under the GNU Public Licence. Developed by two blind Australian developers, it has acquired a positive reputation and a strong following, even if most users may still also use JAWS and other commercial applications. It relies commercially on grants from large companies, attracted by the good publicity, eager to do something charitable, and perhaps also interested in annoying their competitors. For example, the Mozilla Foundation has funded NVDA to provide support for the Firefox web browser, which enables Mozilla to proclaim that their browser is "more accessible" than Microsoft's Internet Explorer and helps them to break down the Microsoft lock on access software. NVDA, like many good open-source products, is strongly influenced by the market leader, JAWS, so it is familiar and useful for experienced blind users and receives their approval. After four years NVDA is looking increasingly mature and has enabled many users to move away from the commercial software upgrade cycle.

Finally, System Access To Go is a commercial screenreader that is delivered over the web. Users navigate to a simple small download that runs without installation and starts speaking: this operates on the whole machine, not just in the web browser that ran the download.

Access Software for Apple Mac OSX

Apple shipped the 10.4 version of its Apple Mac operating system in 2005 with the new built-in VoiceOver screenreader. Superficially, this operates in a similar manner to Microsoft's built-in Narrator, in that it relies on applications complying with the Apple operating system accessibility API, the Mac OS X Accessibility Protocol, just as Narrator relies on applications complying with MSAA/UIA. However, while Narrator is generally unpopular, VoiceOver has received favourable attention and has a following. There are two reasons for this.

First, VoiceOver is a fully-featured screenreader, with all the shortcut and hotkeys that blind screenreader users expect and can use to maximise their productivity. It works with dozens of Braille displays out-of-the-box, and has multiple high-quality voices (as of the Lion OS 10.7 update) in many languages.

The second, more important reason for VoiceOver's success is the widespread correct adoption of the Mac accessibility API. In Window, few applications have been written to support MSAA/UIA fully. On the Apple platform applications such as Safari (web browsing), Pages (word processing), iTunes (media), Finder (file explorer) all work very well with VoiceOver "out of the box" and are all standard applications. A blind screenreader user with an Apple Mac can enjoy the use of the same applications as his sighted peers because they have been written to support the Mac accessibility API.

There is a significant caveat, however. All these applications support the Mac accessibility API because they have been developed by Apple itself. Other non-Apple applications, like Mozilla Firefox, or OpenOffice, or Microsoft Office, fail to do so or do so as an afterthought. This is not uncommon: Apple's iTunes for Windows appeared in 2003, but blind users had to wait until iTunes 8 and the threat of legal action in 2008 to get MSAA support. Blind people on the Mac therefore find themselves in a walled garden. They are fine so long as they are happy with the Apple applications, but if they need to step outside that garden then their screenreader breaks down. For home users this is perhaps not a problem: the Apple applications are robust, powerful and inexpensive. For business users unable to use Microsoft Office this is unsatisfactory and perhaps prevents more widespread adoption of VoiceOver.

It is interesting to compare the Windows ecosystem, where competing third parties create screenreaders, and the Mac ecosystem, where now only VoiceOver exists (at least since the end of OutSpoken from Berkeley Accessibility). On Windows the Mozilla Foundation was able to grant the NVDA project money to support Firefox: Apple was able to persuade WindowEyes to develop iTunes support. The screenreader developers could then trumpet their new feature – "works with Firefox/iTunes!" – and the application developer could claim accessibility and avoid lawsuits, negative publicity and potential loss of sales from corporate and government customers. On Apple there is no such leverage: Microsoft is unlikely to pay Apple to make VoiceOver work with Office, Apple is unlikely to take the money and do it. The situation on the Mac – VoiceOver with Apple products, or nothing – is therefore unlikely to change.

Access Software for GNU/Linux

On the face of it the GNU/Linux environment ought to be a good one for access software: all the underlying technologies from the kernel upwards are available as source code, the high costs that are alleged to be a significant barrier to the deployment of access software are missing, and the ability to review, re-use and re-distribute code means developers can cooperate rather than employ the patents and trade secrets of the commercial screenreader vendors. In practice, however, the free-wheeling development environment is at odds with access software development, which requires consistency and compliance with agreed standards. The situation is better when developers are interested in "scratching their own itch", developing solutions for their own needs like Google's T.V. Raman, or corporations are willing to pay for development to meet some corporate or political goal, such as Mozilla funding for NVDA.

Discussing Linux accessibility is hard without going into distros and window managers (e.g. GNOME versus KDE). A few general points: Linux, like Unix, has a greater emphasis on commandline usage than OSX or Windows, which may be more accessible for screenreader users – it is at least linear and text-only – and the Speakup screenreader has been available to read shell windows in GNU/Linux since 1998. Like OSX and Windows, GNU/Linux has system-wide accessibility settings for font size, colour, contrast and magnification: like them, too, applications may or may not respect these settings. There are three interfaces analogous to MSAA, the widely-supported Assistive Technology Service Provider Interface (AT-SPI), the Accessibility Toolkit (ATK) used in GNOME applications, and the IAccessible2 interface used in Firefox (as in Windows), and the situation is further confused by different approaches of the two main desktop environments, KDE and GNOME. But all the fundamentals are there.

The current mainstream screenreader for GNU/Linux is Orca (continuing a tradition of naming screenreaders after sea animals) taking over from Gnopernicus. It is installed by default with the GNOME desktop, used in the Ubuntu distribution, but various system and applications settings have to be changed to best employ it. The Vinux distribution addresses this by creating an Ubuntu distribution in which Orca and other accessibility settings are enabled by default. One other GNU/Linux system of note is the Emacspeaks application, which enables the mighty Emacs text editor to control an entire system, read the web, do email, all with speech.

One significant issue for GNU/Linux systems is that high-quality speech synthesizers are not so readily available as on the Windows and Apple platforms. MBROLA voices and eSpeak voices are free, and commercial voices are available, but generally human-sounding voices are the preserve of the commercial platforms. However, screenreader users often prefer the more “robotic-sounding” voices as being more reliably-understandable and amenable to being speeded up to an astonishing degree.

Access Software on tablets and mobile (cell) phones

These two types of device are being taken together because they share operating systems: in ten years when we all are using tablets instead of PCs this may seem a ludicrous aggregation.

It is certainly the case for sighted people that a tablet is not the same as a phone: for a blind user it may or may not be more similar.

Texting and using an address book on a cellphone has been a problem for blind people. Nuance’s Talks&Zooms for the Symbian phone operating system from Nokia enabled users to use a cellphone and provided an enormous degree of independence. However, it has since been eclipsed by the hugely influential iPhone from Apple, using the iOS operating system and including the free VoiceOver screenreader. Apple has built VoiceOver right into the heart of how the iPhone (and the later iPad) operates, so every application developed by Apple for the iPhone/iPad works with VoiceOver – including the web browser, Safari, which allows the blind user to explore a webpage by touch in two dimensions (speaking what is under the finger), quite different from the traditional re-presentation in a screenreader as a linear stream of text items. In addition, the control that Apple exercises over the applications permitted in the Apple App Store and the constrained environment in which they operate means that most applications are accessible by default to an extent that is not the same on the Apple OSX desktop, let alone in Windows. The iPad, which is larger and therefore requires less fine control and dexterity to operate by touch, also allows people with some sight to easily bring the display up to their face, and zoom web pages and other content easily with gestures, so people can take advantage of what sight they have. Both can drive Braille displays by Bluetooth. Taken together they represent a revolution in accessibility for blind people – at least those able to afford the hardware and to understand and utilise the powerful features. Anyone seeking to use a different screenreader from VoiceOver, however, will be disappointed because of the structure of iOS: only one application can run at once, so you cannot run a screenreader at the same time your email program or web browser. Self-voicing applications are possible, and eBook readers on the iPad often have this feature, but third-party screenreader development on the iOS platform is currently impossible.

Until Windows 8 Mobile comes out in 2012 the only other serious player in the cellphone space is Android from Google. Android has a screenreader developed by Google, Talkback, and Android 1.6 “Donut” in 2009 introduced an accessibility API for third-party applications. At present iOS has the momentum and popularity, but Android is selling in greater absolute numbers and becoming more powerful with each new version. Its more open architecture may allow for more third-party technology, such as the Safe & Sound applications from screenreader.net.

4 Big specific issues for Access Software

It is useful to review some key challenges for access software that will shape new and existing products over the next five years.

The new Web (HTML5, the cloud)

It is possible that everything in this chapter about screenreader applications running on desktop systems will be as redundant in twenty years as the green-screen terminal. The current fashion is for applications to be written as web pages – that is, HTML plus JavaScript and CSS, variously known as “HTML5” or “Web 2.0” or “Ajax” or “DHTML” or “web applications”. These applications can now be as fully-featured, complex, and hence unusable for screenreader users as desktop applications.

How does the blind user approach a web application? At present, of course, he uses his screenreader to access the standard browser which is rendering the web application. At a superficial level, then, this is a problem of degree, not of kind: web applications are web pages rendered in browsers, and screenreaders know how to read web pages in browsers, so all is well. There are two problems with this optimistic view.

First is the degree of complexity. Microsoft Word is more than a text box with a toolbar stuck on it, and Google Docs is more than a TEXTAREA element in a web page with some links. Saying that the individual components are accessible does not mean that the whole is usable in any practical sense. In effect, the problems of supporting applications described above now extends to websites: if your screenreader vendor can support ten applications, but now must consider Facebook and Microsoft Office 365 and Google Docs all as fully-featured applications in their own right, with all the scripting and thought and work required to support them fully, then the number of native non-web applications on your machine that can be supported is reduced by three. This would not such a problem, if the web application market were as dominated by a few major players as the desktop market: if everyone on the web used Google Docs, and no-one used Microsoft Office 365, then you needed only write the code to support Google Docs. But the web is in a state of flux: we do not know which web applications will win out, or even if the monocultures enjoyed by screenreader users in the past will ever be achieved again. So in effect the access software industry must suddenly support dozens of innovative new online applications, new ones are being developed at a rapid rate, and applications are able to change literally overnight without user resistance.

The second problem is that the technology employed in building web pages is changing. Microsoft's domination of the market with Internet Explorer 6 has ended, and the result is a sudden rate of change in HTML and associated technologies not seen for a decade. It is not just the extent of the changes, but their nature: for example, input elements, like buttons, which are correctly identified for screenreaders by browsers, can now be replaced by other arbitrary elements (e.g. DIV elements) with the appropriate click and focus events. For the mouse user they look and work the same way as a "real" button. For the web developer new exciting features can replace the old boring button. For the screenreader user, however, his identifiable button element has disappeared, and suddenly he cannot use the web application at all. Of particular note is the HTML5 CANVAS element, which completely abandons the idea of HTML rendering content and JavaScript providing only functionality. This may never have been true in practice – try turning off JavaScript in your browser and going to Facebook or the BBC iPlayer –but the general principle was that a static, accessible HTML page should fulfill all the tasks for which the page was written, without recourse to the complicated and inaccessible features of JavaScript. A CANVAS element by contrast is a blank, textless area on which images and text are drawn by JavaScript – exactly like desktop applications before the accessibility APIs, when screenreaders had to rely entirely on offscreen models and intercepting drawing messages. CANVAS is already being employed for games and graphical applications, and browser developers have committed to developing graphic acceleration techniques to make it more responsive. As graphical toolkits are developed for CANVAS, just as toolkits like GTK or Qt or Java Swing have been developed for desktop operating systems, we can expect to see more applications employ user interfaces built entirely with CANVAS. It seems almost certain that these applications will be inaccessible or at least unusable without further web browser and screenreader development.

The situation on HTML5 is analogous to the position of desktop machines in the 1990s before MSAA, but it is not identical. We start from a position of advantage. Desktops have progressed from off-screen models and inaccessible applications to accessibility APIs and the assumption that accessibility should be expected. This that route provides a road-map and an example for HTML5 development. The accessibility API for the web has been created, the WAI-ARIA standard, which defines buttons and roles and events in much the same way as the MSAA/UIA/Apple guidelines. Operating system vendors have mapped WAI-ARIA to their particular operating system accessibility APIs. As HTML5 matures, graphical toolkits like the Yahoo! User Interface Library will produce user interface widgits that comply with WAI-ARIA standards, just as Windows toolkits like Qt have produced UI widgits that comply with MSAA. Accessibility, although too-often defined as “works for JAWS users”, has won the argument and is regarded as something that vendors and developers must support, and is also a good way for developers to demonstrate their expertise. Web developers will therefore work to support accessibility - assuming that they get it right. The increasing complexity of the accessibility guidelines would suggest that we are have become better at using them, but this is not necessarily the case. For example, after fifteen stable years, the simple alt attribute is still widely misused by HTML writers: the new accessibility elements in HTML5 and the WAI-ARIA attributes are more complicated. It seems unlikely they will be correctly used, if they are used at all. Huge arguments have erupted in the HTML5 specification process at the dropping of the little-used LONGDESC element, for example: the sorry truth is that it is almost never employed correctly.

The problem is that traditional HTML web pages, limiting as they were for developers and designers, were fundamentally accessible “out of the box”. It is hard to write an inaccessible HTML3 web page if you populate the alt attributes on your images. By contrast it is surprisingly easy to write an inaccessible HTML5 web page: there is so much more opportunity to get things wrong, from using the wrong mechanism to hide error text to trapping the caret in a text area while you try to validate form input. WAI-ARIA assumes that this situation will be resolved by the developer doing more work to add back in the “this is a button” kind of information to allow screenreaders to operate, but experience suggests that this often fail to happen. Since we are moving from enforced accessibility (thanks to the limitations of HTML4) to optional accessibility (the more powerful HTML5) then initially at least HTML5 will be a retrograde step for accessibility. A prosaic online example of this for screenreader users is Macromedia Flash, which like HTML5 can be made accessible but usually is not: it seems likely that this will be the case for much of the HTML5 web in future.

It remains the case that individual sites can now be as complex as applications, and will now require as much investment as individual applications by screenreader vendors, so much of the web will become less accessible. However, highly-used applications like Facebook or Google Mail (or their future incarnations) will be made accessible just as Microsoft Word is now. The general idea that just by mastering your web browser and screenreader you can access every website that may be lost. But our experience with the desktop environment helps us to know how to proceed, and although the process will be painful we can hope to achieve yet more accessible and powerful applications delivered through the web to blind users.

Media (radio, podcasts, television)

One of the enormous advantages of the Internet is the sheer quantity of media available. Films, TV, radio can all be obtained by the blind user from their machine. Audio description can be provided for online formats by the media author or by third-parties.

The obstacles that exist are primarily political and economic, not technical. Commercial sources of media, such as the BBC or NetFlix, restrict the user interfaces and media available for licensing reasons: they seek to prevent users from being able to copy media or access it in unapproved ways. For example, the BBC makes television programmes available to PC users only through a Flash interface that is difficult for many blind people to use rather than making available their existing H.264 video streams for iPhone, and has taken legal action against anyone attempting to make these streams available for non-iOS users. Blind users therefore receive a less-accessible service.

The obvious question is why blind users do not simply pirate content. Piracy software is often inaccessible, pirate sites even more so, and the high level of fakes and malicious virus-infected content may be off-putting for many blind users who struggle with security issues. However, the true answer is probably demographic: blind people tend to be older, and for technical and social reasons are less likely to pirate. This may change with time, as a quick search for illegal cracked copies of JAWS will show.

Finally, the design of the delivery of multimedia content can make it hard or easy to use the content. The Youtube site, for example, auto-plays videos as soon as a page is loaded. The BBC iPlayer site requires the user to navigate to and click a button on the page to start the video. Whilst auto-playing content is generally disapproved of, especially by technically-skilled screenreader users who want to retain control over when the content plays and how it interferes with hearing the speech from their screenreader, the YouTube solution is much better for many screenreader users – possibly the majority of them. Standards and expert opinions are not necessarily the best guide to the best experience for the majority.

Office and Email (Microsoft and Others)

Consumption of content is relatively simple to define in terms of user interface, but creation of content is far more complex. This makes screenreader support for content-creation programs more challenging.

The challenge for blind people is that the content they create must be of a good standard for sighted consumers of the content. gETTING YOUR CASE WRONG IN wORD will make you look foolish but is easily done with one accidental keystroke if you cannot see the text you are typing. Tabs, font sizes and other easy traps await. Screenreaders therefore provide mechanisms to allow users quickly to identify their current styles, and to alert them to unexpected content like ALLCAPS passages. Ideally, perhaps, content creation in email or word processing would be more like content creation in online blog interfaces like WordPress. Here the user may enter only plain text but the rendered final output acquires attractive and consistent styles and the effect is pleasing to both author and consumer. However, you still need some knowledge of styles or even markup, which is fine for technical users but beyond most people, sighted or not. More generally, although sighted people routinely use layout and formatting to communicate semantics (headers, font sizes, layout) blind people are more reliant on third-party help or trusting that their audience will understand their limited repertoire of visual effects. As a sighted person, websites created by blind people are often obvious as soon as they are seen: linear text, no layout, no indentation.

Another problem for screenreader users is spelling. Users who were once sighted have probably learned to read and write English, so the irregular orthography is at least familiar. Users who have never seen the written word find it much harder to communicate using English (users who have never heard English either, being deafblind, face greater challenges still.) Homophones like "there" and "their" and "they're" can elude the blind user when reviewing text, and the handy red underline is not so useful for screenreader users, who must rely on per-application spellchecks or copy and paste text a great deal to and from Word.

PDF (Adobe Reader, PDF accessibility)

PDF is the *de facto* format for documents to be printed, exchanged or archived, and is increasingly common for eBooks. As a format designed for printing, not electronic consumption, it can easily be completely inaccessible to screenreaders (containing only image data, not text, and requiring an Optical Character Recognition (OCR) image-to-text step) or accessible but completely unusable (text in the wrong order, big file sizes, very complex user interfaces and options, and an experience based on a per-page model that works really well only with mouse users.) PDFs have therefore acquired a reputation for inaccessibility. In amelioration the *de facto* standard client, Adobe Reader X, has a "save to text" option, built-in self-voicing options, and there is now a widespread understanding that you must create an "accessible" (non-graphics) PDF file for distribution. In recent years OCR programs have dropped in price, free accessibility tools and client accessibility APIs are become available, and most screenreaders now support Adobe Reader, so the situation is much improved. It is not unfair to say that PDF files now, although still more cumbersome to read than, say, a Rich Text File format document, are now accessible for the majority of users, and suffer from the inherent tension between original print and current electronic function rather than any specific problem.

Printed words (OCR, copyright)

Ebooks, like other media, are restricted more by politics than by technology. The latest Harry Potter novel can be pirated in minutes (usually as a PDF file). However, the official legal channels of Bookshare and other organizations dedicated to providing accessible formats are increasingly speedy and the range of material is increasing. Publishers have been loath to provide content in electronic formats, seeing it as enabling piracy and concerned about the additional costs and difficulties in creating accessible formats. However, it only requires one person to create a PDF and make it available through the Internet, and so eBooks are increasingly available illicitly. In some jurisdictions it is even legal to make and distribute copies of books for people with disabilities (e.g. the Copyright (Visually Impaired Person) Act, 2002, in the UK).

The availability of an eBook, perhaps as a PDF, is not the same as having a fully-accessible version of that eBook, perhaps in the specially-designed DAISY eBook format, with features like shortcuts to chapters, an index, formatting to allow for magnification for people with some sight and even embedded text-to-speech. These improved formats are especially important where the book is not a linear novel but a textbook where skipping around chapters and referencing particular passages is vital. These truly-accessible formats are still rare, but may become less so as more of them are produced by publishers and agencies, now cognizant of how to make these books and seeking to work better with iPads and Kindles.

Security

Computer security is a concern with everyone, but may be particularly worrying for blind people who are older and nervous about computer use. They also find it hard to pick up on visual cues (font, spelling, color and graphics) that allow sighted people to discriminate between genuine and malicious pop-up messages and other attempts to make users enable malicious code, like “Your machine is infected, click here!” pop-ups. When you have become accustomed to your computer launching unexpected user interfaces at you, and these user interfaces contain little except “OK” buttons, it is unsurprising if you assent to something dubious without meaning to do so.

Voice output

Most screenreader users employ synthesized speech. It is often surprising to sighted people how some blind users speed up their speech synthesizers to a rate that is almost incomprehensible to someone who is not used to it. This helps blind users to compensate for their low rate of reading compared to sighted people and their inability to scan and quickly analyze information sources or user interfaces. The speech synthesizers that cope with this process often employ techniques that make them sound very regular but very robotic and mechanical. Conversely, some blind people cannot understand how some other, perhaps less technical blind people, find more naturalistic, human-sounding voices more acceptable and will refuse to listen to a system that is “too robotic”. Getting the right speech synthesizer to a user may be as big a factor as getting the right support or the right screenreader or other access software.

Free, high-quality text-to-speech voices are more and more readily available for European languages and Chinese, either as part of the operating system (Apple’s Lion OSX 10.7 comes with more than forty voices.) However, for screenreader users in non-Western languages, synthesized speech is poor quality, expensive, or non-existent: the only voices available are often produced by universities, do not work on commercial platforms like Windows and disappear when the student responsible graduates. This may change as non-Western countries like India further develop their vibrant software industries, or operating system vendors include more high-quality voices in their operating systems for free.

References

Asakawa, C., Itoh, T. 1998. User Interface of a Home Page Reader. In *Assets '98: Proceedings of the third international ACM conference on Assistive technologies.* Blattner, M.M., Karshmer, A.I. (Editors) Marina del Rey, California, USA. ACM Press, New York, USA, 1998. pp149-156.

Bennett, D.J., Edwards, A.D.N. 1998. Exploration of Non-seen Diagrams. *The Fifth International Conference on Auditory Display* (*ICAD98),* Glasgow, U.K.

Blenkhorn, P., Evans, G., King, A., Kurniawan, S.H., Sutcliffe, A. 2003. Screen Magnifiers: Evolution and Evaluation. In *IEEE Computer Graphics and Applications*, Volume 23 Number 5, September/October 2003, pp54-62. http://www.alasdairking.me.uk/research/Blenkhorn2003-ScreenMagnifiers.html.

Blenkhorn, P. & Evans, G. 2001. Considerations for user interaction with talking screen readers. *Technology and Persons with Disabilities Conference* (CSUN) 2001. Northridge, California, USA.

Blenkhorn, P., Evans, D. G. 2001. The Architecture of a Windows Screen Reader. In Marinček, C., Bühler, C., Knops, H, & Andrich, R. (Eds.) *Assistive Technology – Added Value to the Quality of Life (AAATE 2001)*, IOS Press, Amsterdam, pp 119 – 123, 2001.

Evans, G., Blenkhorn, P., 2003. Architectures of Assistive Software Applications for Windows-based Computers. *Journal of Computer and Network Applications* 26, pp213-228.

King, A. 2007. Re-presenting visual content for blind people. PhD Thesis, University of Manchester. http://www.alasdairking.me.uk/research/PhD.htm.

Kurniawan, S.H., King, A., Evans, G. & Blenkhorn, P. 2006. Personalising web page presentation for older people. *Interacting with Computers* (18) 2006, pp457-477.