The Institute for the Future of the Book *presents*

Sophie

A quick introduction to the future of reading & writing *by Dan Visel*

The philosophers have only *interpreted* the world in various ways; the point is to *change* it. (Karl Marx, "Theses on Feuerbach," no. 11)

INTRODUCTION

THE INSTITUTE FOR THE FUTURE OF THE BOOK will soon release the first version of Sophie, a project that we believe will transform the way we go about reading and writing in screen-based environments. This document will introduce the Sophie project in several steps:

- First, a look at exactly what's wrong with electronic books: both existing implementations and conceptions of what electronic books can and should do.
- Second, the specific needs that necessitate Sophie.
- Third, a description of exactly what Sophie 1.0 will do.
- And finally, we'll look at how Sophie will work with other applications and where Sophie can go in the future.

WHAT'S WRONG WITH ELECTRONIC BOOKS

There's no shortage of formats of various things that could be construed as electronic books. Text files of existing books have been around almost as long as the personal computer. While a bewildering plethora of formats exist, two basic models of electronic books have been presented and achieved some degree of success in the recent past. While both work for certain uses, both have significant inherent drawbacks that inscribe themselves indelibly on work based on these formats. It's because of these drawbacks that we're releasing Sophie: let's take a look at exactly how existing electronic book formats work and why that's not good enough.

THE PDF-BASED MODEL

PDFs are what many people think of when they think about electronic books and there's a reason: they're everywhere. PDFs are used for many sorts of documents, and there have been a number of attempts to use them as electronic books. Why? A quick look at history suggests that making books as PDFs might have been the path of least resistance. PDFs started off as a compressed version of Postscript, a format also designed by Adobe as a language in which computers and printers could talk to each other. The triumph of Postscript is shared by PDF: every printer or computer that understands how to decode a Postscript of PDF document will print or display the document in exactly the same way. This has been fantastic for print-based publishing, which is where the formats took off; indeed, the majority of printed books today spend some portion of their life as a PDF.

PDFs work for presenting material that needed to look exactly the same every time in a screen-based environments. For graphics and text it's a fine solution: it's entirely possible to take the text of a book, format it, and present it as a book. This is how electronic books formatted for Adobe Reader work. If you need to represent paper electronically, PDF is a good way to do it.

But wait just a minute. Isn't this document a PDF? Isn't it hypocritical to criticize a format that you're using?

Not necessarily. PDFs do certain things well: most people can read them, for example, and if you know how & have the right program you can make something pretty. But look at what you can't do with this PDF: try editing this to make your own new version of this introduction, for example. Or try incorporating part of this into something of your own. Or try wondering why this PDF of this document doesn't include video of Sophie in action, or audio of me reading it, or different translations you could choose from, or . . .

WHAT'S WRONG WITH ELECTRONIC BOOKS *(continued)*

But PDFs are problematic if you want an electronic book to do more. For all Adobe's efforts, they're not very interactive. While it's possible to put forms, buttons, and links into PDFs, as well as audio and video, there's no dedicated content creation system to let any but advanced users create such PDFs. Creating a PDF remains very similar to printing a document: the printout is static compared to the living document, a snapshot of one instance in a document's life. If you want to make changes to the text, you're generally out of luck: instead, you're going to have to remake the document in the program that created it, which you, the reader, might not have.

BROWSER-BASED BOOKS

The second model of electronic books existing today is based on a markup language like HTML or XML – in effect, a specialized use of the technology underlying most of the web. Content created in this way is more fluid than a PDF: in many browsers, for example, you can make the font size larger or smaller, change how many words there are in a line of text, or change the font. No two readers necessarily see the same thing. If you're a programmer, you can transform the form of a document made in XML so that it can be used in other ways: with a bit of technology, for example, you could take an XML-based book and make it appear both on the Web and as a PDF.

There's a major conceptual problem with this sort of design philosophy, though. The concept of a markup language assumes that all documents can be turned into an outline – markup works by taking all of the text in a document as a single string of letters and chopping it into differently-formatted pieces. This is very natural to computers – it's how most documents are stored internally – but it's not natural to people. While it does work with many documents, it's a constrained approach that limits what can be done with a book. Another problem is that XML-based approaches completely separate form from content: ask any web designer about the headaches of building a website that looks the same on every operating system and every browser. This doesn't matter so much for exclusively text-based ebooks: most people don't worry about how their pages of Jane Austen look. But for graphic-heavy books, this is a major problem; with HTML it's still quite difficult to represent exactly what's on a page the same way every time. And even text-based books that contain slightly unordinary content, like math books with formulae in them, tend to fall apart on the web.

BUT WHAT'S WRONG WITH THE WEB?

While it's not presently a workable solution for electronic books, let's not dismiss HTML entirely. It's worth noting that with the rise of the Web the reading public has become used to reading from screens and

And what about HTML? Should you want a more interactive version of this document, you can get one from our website (*www.futureofthebook.org/sophie*). It's not as nice looking as this PDF. Soon we'll post a Sophie version of this: then you won't need to choose between what you want and what you can have.

WHAT'S WRONG WITH ELECTRONIC BOOKS *(continued)*

now spend a fair amount of time doing so. While people still complain about reading on-screen, it's difficult to deny that millions of people are doing it every day. But no one would claim that the web is ideal as a reading environment (to say nothing of it as a writing environment). It serves both purposes well enough to get by.

While the various technologies provided by the web afford new opportunities and forms for writing – consider the blog – there's still not a good environment for easily creating unified works that take advantage of the multimedia possibilities of modern computers. Various technologies approach this desideratum from various angles. A plain text editor will let you write a book-length novel in HTML. Flash will let you make interactive multimedia presentations. Something like Flickr will let you make a slideshow of your photos. But it's difficult to mix forms with these technologies. If you want to present a Flickr slideshow of illustrations as a reader views your novel which you've posted on your website, you're out of luck. If you want to make an interactive presentation of comments in your blog using Flash, you're also out of luck, unless you want to hire a programmer.

WHERE DOES THAT LEAVE ELECTRONIC BOOKS?

Both PDF-based and HTML-based technologies are useful. But nobody's excited about electronic books right now – and that's with good reason. Both betray a paucity of imagination. Here, in a nutshell, is the problem with these ideas about electronic books: they treat a book as being essentially a text. A text, however, is not a book: a text is only part of a book. A book presents a *reading environment* which is something we're interested in creating with Sophie. Electronic books could be much more: there's no reason to stop with what we have.

The hardware red herring

Popular concepts of electronic books can be traced back to Alan Kay, who came up with the concept of the Dynabook in the late 1960s. The Dynabook was "a portable interactive personal computer, as accessible as a book," which Kay visualized as an educational device. Thus: the ebook. In the years since Kay defined the Dynabook, computers have become ever smaller: in terms of hardware, a notebook computer is a great deal like what Kay was talking about. But there's still a major problem: while there are pieces here and there, we still don't have a dedicated software interface for a Dynabook. Sophie aims to change that.

WHY SOPHIE: THE NEED FOR SOPHIE

In addition to the enduring problems with existing forms of electronic books, some more specific needs have engendered Sophie. Some of these problems were addressed in Sophie's historical precursors – see Sophie's History for more on this – some are more recent. Here are a few of them.

THE TWO-HUNDRED YEAR PROBLEM

A major problem emerging with digital work right now is that of time. Paradoxically, digital work, as abstractable as it is, doesn't seem to last as long as print-based work. Some print books have lasted half a millennium; even the most poorly printed book should be readable in half a century. The same's not true of digital files. Media change: the 5.25" floppy disks used to save files twenty-five years ago are entirely useless now, and it's increasingly difficult even to find a drive that can read 3.5" disks. Formats change: presentations designed to run on the operating systems of ten years ago might not work today.

Even in the world of the Internet, untethered from the confines of hardware, links break and code rots: look at what remains of the rusting hulks of websites from five years ago left preserved for history at the Wayback Machine. Despite the laudable aims of the preservationists, most sites don't last: functionality might be tied to a specific server, and if that's no longer the same, the page no longer works.

This is an enormous issue for libraries attempting to archive digital collections. It's pointless to attempt to preserve material if it can't be read in ten years, or can only be read with specialized equipment. If you want to understand what happened in the Iran-Contra scandal ten years ago you can go to the library and look at the newspapers of the time. What happens in twenty years when someone wants to understand what happened in the Valerie Plame leak scandal, where the newspapers are following blogs?

Sophie is an attempt to address this. Because Sophie is written in Squeak, a platform-independent environment written in Smalltalk, it's not tied to any particular device. Sophie books will open in exactly the same way across any platform that Squeak runs on – be that a Mac, a PC, a Linux box, or some yet to be invented ebook reader. If an operating system comes out in fifty years that's entirely different from current operating systems, your Sophie book – and any other Sophie book – will still run in exactly the same way, provided that a Squeak virtual machine can be built for it.

THE ANNOTATION PROBLEM

Books are unique media objects in that they can be written on as well as read from. If I give you my copy of *Ulysses* which I've marked up, with notes in the margins at all the parts that I found interesting, it's a different object than a fresh copy of *Ulysses* from a bookstore. If I have useful things to say, that copy might be a more useful book.

This is a functionality which isn't as present in other media, and whether or not your write in your books, this capability is part of what makes them special. You generally can't mark a specific point in a specific song on a CD as being especially good and worth returning

I have a Trick of writing in the Margins of my Books, it is not a good Trick, but one longs to say something.

(Hester Thrale Piozzi, 1790)

WHY SOPHIE: THE NEED FOR SOPHIE *(continued)*

to. You can't attach your own subtitles to a movie as a note. In many electronic books, you can't even scribble notes to yourself – or, if you can, annotation is only possible in a very proscribed fashion.

This won't be the case in Sophie. Sophie will blur the lines between reading and writing in a way past the capabilities of print books. Imagine that I'm reading a book: I find something that I disagree with. If it's a print book, I might write "This isn't true," or, if the margins are wide enough, I might start scribbling an explanation of why I think it's not the case. Were I using Sophie, I could add a note – a note that's longer than the book, if need be. If I own the book, I could attach that note to the book permanently. Or I could spin new note off as its own, linked, book. Or to go furthest of all: if I have permission to do so, I could rewrite the content entirely.

Other sorts of less confrontational commentary will also be possible. If I'm reading a book and I find something that reminds me of a webpage, I can attach a link to it. I could pass this annotated book on to a friend; she could read the book, read my annotations, and attach her own. If I had a hundred friends, they could all do the same.

While reading a book seems like a solitary activity on the surface, underneath it has always been a quietly social experience, a communication between the author and the reader across time and space. Sophie will make these metaphors concrete.

THE ACADEMY'S NEEDS

While Sophie can be used many other places, it's being aimed squarely at the world of education. A platform is needed to teach composition – not just writing but also how to use multimedia effectively, increasingly a demand in today's world. Camera phones and video cameras, to just give two examples, make acquiring multimedia a snap: plenty of schools let students create video reports or are interested in using multimedia. We live in a world full of graphic design, full of video. Sophie can be used to encourage media literacy, so that students aren't merely subjects of such a world: if you know how to create a video presentation, you're better equipped to understand it.

But: the programs currently available and being used aren't up to the job. More and more students are taught to make presentations with Powerpoint; it's a limiting program that, as Edward Tufte has pointed out, encourages gimmicky special effects at the expense of coherent thinking. Students often spend more time working out flashy transitions between slides than writing the report the slides are intended to accompany. No one gains anything from this.

Sophie will make it possible to integrate the writing of the report with the slideshow that would have accompanied it. Sophie treats all media equally: if adding a slide show would be helpful to a primarily written report, the student can add it on to the page it's intended to illustrate. No longer do you need to switch from Word to Powerpoint.

AUTHOR/READER INTEROPERABILITY

A metaproblem: I'm writing this document in Adobe InDesign because it lets me create attractive-looking PDFs. Bob wants to edit it. Not being a designer, Bob has no reason to have InDesign. I can send Bob a PDF, which he can attach notes to, or I can (arduously) vacuum all the text out of the InDesign file and save it as a Word file that he can edit. I could write my text in Word, then lay it out in InDesign, but I want to edit my text and layout simultaneously.

This is far more complicated than the situation needs to be. With Sophie, anyone can – assuming they have permission – edit any book they're reading. You don't need a special author program, or a special reader program: you just need a copy of Sophie.

WHY SOPHIE: THE NEED FOR SOPHIE *(continued)*

DEMOCRATIZING THE WORLD OF MULTIMEDIA

Many of the things that Sophie will let users create are possible with existing technologies. The problem is that they're an enormous amount of work – so much so that in many cases you'd need a staff of programmers or designers on staff to make them possible. In the mid-1990s, creating a CD-ROM needed the attention of half a dozen producers, programmers, and designers. Creating a sustainable website (an ongoing investment)can easily take twice that. Creating a film and getting it into shape for distribution can require even more.

Because of this, media production tends to be the province of the corporate: corporations generally have the money and the people to throw at interesting media projects. A corporation can afford to hire a programmer to create a custom platform if they need one. A corporation can hire someone who knows Flash to create a website. An individual or a small non-profit that wants to create something competitive is comparatively at a disadvantage: most of us don't have someone who knows Flash at our beck and call, nor are we interested in going to the trouble of learning to use it for a one-time website.

Software has traditionally played to this dichotomy between authors and designers. Because I have design experience, I can make a PDF with Adobe InDesign that includes niceties like a table of contents and hyperlinks, as well as nicely laid out pages. I'm an exception, though: most authors aren't designers.

As Dick Higgins, a self-described intermedia artist noted, writing has been unique among the arts because it requires no materials but a pencil. Text is special this way: anyone can write. Sophie aims to level the disparities between text and other media along with the disparities between the individual voice and the corporate. With Sophie, authors won't need specialized training to assemble complex multimedia books. With Sophie, an average user should be able to make something like this document – or something more exciting.

This isn't to say that there aren't corporate uses for Sophie. If you want to hire someone who can create a custom Sophie application, you certainly can (see **Where Sophie Can Go**, below); we fully expect that there will be a micro-industry of Sophie technicians. But for most uses, this shouldn't be necessary.

I want to defend literature. It's a poor man's art. You can think, even when you can't feel comfortable among the cigarred princes and the knockkneed venerables in miniskirts that run our visual art scene. You can write when you can't afford the fancy poncy materials to make art – canvas, silk screens and the right kind of paint.

(Dick Higgins, "Seen, Heard and Understood", 1972)

WHAT SOPHIE WILL DO

As we've noted, Sophie is an all purpose tool for dealing with media. It will allow users to easily create books that can contain any sort of media on hand – text, images, sounds, videos, animations. Sophie does for media what a physical book does for text and images: with Sophie, authors can create multimedia books. You might think of it as a wrapper for anything digital, but it's more than that.

NEW STRUCTURES FOR BOOKS

It's a mistake to think that a book is just a container for text. A well-designed book presents text in a useful way. Similarly, Sophie's more than just a wrapper for media: it will allow users to structure information intelligently. Sophie differs from previous platforms for electronic reading by giving the author as much control over the form of what they're making as the content.

Here's an example. Let's suggest you have a cookbook. Like many types of books, a cookbook isn't sequential in nature. The vast majority of the time that you use a cookbook, you're not going to open it to the first page, read the introduction, and then read each and every recipe in the book. Instead, you're going to find the recipe you want (possibly after a bit of browsing) and read that. Unfortunately, current models of electronic books lock content into a Procrustean bed of sequentiality. Nonlinear content is strapped to a hierarchical outline, which make a number of normal reading behaviors difficult, if not impossible.

If we were to make a cookbook with Sophie, we wouldn't need to constrain ourselves to an artificial model – we can take advantage of the virtual space afforded by the computer. A Sophie cookbook might be designed as a tree which could be approached several ways: a table of contents might take readers first to dishes based around vegetables, then to dishes based around spinach, then to a recipe for spinach quiche. Or the book might be searched by ingredients: you could look at all recipes that include spinach, whether or not they are based around it.

When you have the recipe open, you could also look at other parts of the book. The recipe for spinach quiche might link to general tips for cooking quiches of all sorts: this could be opened in another window so that it could be read simultaneously. A more ambitious cookbook might suggest an appropriate wine to serve with spinach quiche; clicking on this link might take you to another Sophie book explaining the virtue of that particular wine.

A reading experience analogous to the one just described could be carried out – indeed, probably is being carried out right now – with physical books: someone has a pile of cookbooks open on their kitchen table all at once. But as yet there's no good way to do it with an electronic book. You might object that it's possible to make something vaguely like this in HTML. Sure. But you're going to have to spend a lot of time concentrating on the structure of the website you're creating: do we really need to think about information architecture when we're trying to make things? Sophie makes this seamless.

MULTIMEDIA

Sophie is media-agnostic: all media is the same inside of Sophie. You could make a book based around a long piece of text (like a traditional novel). Or you could make a book based around a series of photographs (something like a slideshow), adding narration or a soundtrack to play with the rhythm. Or you could make a video-based book, or a book based around a single photograph, annotated with audio to tell a story. Or you could mix any of these (and more) forms together to create something entirely new.

WHAT SOPHIE WILL DO *(continued)*

Here's a more concrete example. Let's make a book that's based around a movie. Imagine that you're a drama teacher; your class has just rehearsed a play, which you've videotaped. When the film clip's in Sophie, you can do an enormous number of things with it. With the text of a script of the play, you could present subtitles with what's written in the script while the actors are saying it. Or you could record a commentary track about what's going on in the play; the readers of the book could switch back and forth between the recorded sound and your soundtrack – or overlay them if they wanted.

Let's say there's been another rehearsal which has been videotaped. You could add the new video to your Sophie book; you could play the two videos side by side if you thought that might be useful. If a scene in one is four minutes long and a scene in the other is five minutes long, you could pause the first for a minute at the end of the scene so that the two videos don't get too badly out of synch.

Or let's say you've gone to Istanbul on vacation. You've taken a lot of really nice photos; you want to share them. Sophie will make it easy to make a slideshow out of them; you can add captions or an audio narrative to go along with them. Instead of making one long slide show of 500 photos, you could split it up. You could start your readers off with twenty of the best pictures; if they wanted to see more pictures on a certain subject – the Topkapi Palace, for example – they could click a button on a picture of it to see a slideshow of details. Or you could have a link that leads to videos you took, or a funny story you wrote about your time in the bathhouse. The options don't stop.

INTERACTIVITY & THE USE OF TIME

Back to history: some of the most ground-breaking products released by Voyager in the early 1990s were the CD-ROMs based around pieces of classical music. To use one as an example, Beethoven's *Ninth Symphony* would play; as it played, text and graphics would explain to the reader what was going on with the music. The music could be paused so the reader could absorb the text; when the reader went to the next page of text, the music would keep up. Modules of the program would break down pieces of the symphony in greater detail.

Since then, however, such interactive books have been almost entirely absent. This absence has much to do with the lack of tools to make them. While it could be hard-coded – like Voyager did it – or done in something like Flash, it's still an enormous amount of work; certainly it's not the sort of thing a professor of music could produce without help.

That this should be the case seventeen years later is ridiculous, and we aim to change it. With Sophie, anyone (with some practice) should be able to make a book as sophisticated as Voyager's *Beethoven*. Or you could make something else using the same techniques. Imagine, for example, a copy of *Madame Bovary* in French. When you move your mouse over a sentence, it might be spoken aloud in English. The same book could be set up so that it reads itself aloud, turning pages as it goes so that a student learning French could follow along. Or take the example of the play: the teacher could turn blocking notes into cues that show over the video.

There is a difference between merely supplying a production apparatus and trying to change the production apparatus. (Walter Benjamin, "The Author as Producer")

WHAT SOPHIE WILL DO *(continued)*

THE PAGE & THE CANVAS

A physical book has pages: we take it for granted that pages follow pages in a sequential order, like spoken words follow each other. You can make books like this with Sophie – it's the default setting. But as has been hinted above, a Sophie book can have a more fluid relationship with pages. Let's look at another sort of reading: how you might use the desktop of your computer. There might be a movie player in the upper left with a video you want to see, there's a music player in the bottom right, there's a text editor open with something you're reading. You might be doing more than one of them at once, bouncing from one to another. Most importantly, however: these are three independent objects. You can look at the movie without playing the audio, or you could play the audio while you're reading the text.

You could very easily create a Sophie book just like this, where a number of different things co-exist on a single page. While we haven't thought of this sort of presentation as a book before, it's largely because we haven't had the chance to consider it as a tool for structuring information. We're used to very determined and linear models of how a book should work: you might think of a book as having a spine going through it, which pages hang off. The canvas model allows us to do away with the spine – or to have many spines, because you can intertwine these different models of book construction. A book constructed as a canvas could present information in several different ways and not pass judgment about which presentation is the more important. In a print book that's mostly text, photos or graphic tend to function simply as illustrations of the text. In a print book that's mostly photos, the texts function as captions: they're secondary information. Sophie doesn't force such implicit ordering.

What can you make with this kind of a Sophie book? We're not sure yet, but we're excited to see what people might do with them.

SOPHIE & THE NETWORK

Sophie will live in a networked environment. This will be possible in many different ways. As mentioned, Sophie books can link to other Sophie books. However, the two Sophie books don't have to be resident on the same computer: the linked Sophie book could very well be on a remote server over the Internet.

Delving deeper into the network, you could also connect to a Sophie book using a web browser. In this case, the Sophie application works as a plugin to your browser, like Flash does. The book, which would be living on a remote server, would open inside your browser, but changes that you make – the same sort of changes you could make to a book on your computer – would be made on the book on a remote server, and can be seen by other people accessing that book. A teacher might post a book for her class to collectively edit; they could all log in to it and make changes.

Books aren't the only thing that can be remote: resources used inside of Sophie – for example, streaming video – can be remote. Because Sophie is extensible (see below), it can also link to or include information outside of Sophie – a webpage, for example, or information from a database. You could put an RSS feed inside Sophie; you could update a Sophie book with information from a blog.

Where does Sophie end? It should be clear that Sophie is not just an environment for creating and reading books in – it's an environment for creating and reading that's conscious of its place in the ecosystem of information.

WHERE SOPHIE CAN GO: SOPHIE SERVER

You can do a lot with Sophie and the Internet right out of the box. But a new world of potential will be released with Sophie Server, a dedicated network environment for Sophie. Sophie Server will allow streaming access to books and their content, and will afford collaboration both in the creation and reading of Sophie books. While this functionality won't be part of Sophie 1.0, it's not far off.

WHAT WILL SOPHIE SERVER DO?

Sophie Server is an application that runs on a server. It will provide a single place for the residence of all Sophie books that exist on a given network. The Sophie Server will thus let users search for interesting Sophie books already in existence. Users connected to a Sophie Server would be able to publish their book to the Sophie Server from inside of Sophie Author.

This repository of books is also useful as a repository of reusable content. Sophie Server users will be able to take content from Sophie books that live on the network and reuse it in their own books. Sophie Server won't only serve as a repository of books: it also can serve as a repository for information that can be used to make books. If, for example, a teacher has a set of images that every member of a class was to use to create individual reports, the images could all be placed in Sophie Server.

If a book is hosted on Sophie Server, it can be streamed – that is, the file can stay on the server, while the remote viewer sees the content within their browser or in their Sophie Reader application. Multiple access to the same book would also be allowed. Each of these readers can create their own set of annotations to the book being read. When annotations aren't saved on Sophie Server, they're saved locally. When Sophie Server is involved, these annotations will be saved on the central server, alongside the book. Users would be able to select and view annotations made by other users, as well as upload their own.

Sophie Server would also keep track of all users currently reading a Sophie book on the network. Sophie Reader will allow concurrent readers to start a chat session; this chat will be logged, and stored along with the Sophie book on the network for future reference.

HOW COULD YOU USE SOPHIE SERVER?

Let's say a teacher assigns a paper to be written in Sophie. The students all write books using their own copy of Sophie; they publish them to the university's Sophie Server. The teacher can thus easily read all the copies and attach her own comments to them which the students can then read. Or: the students all write their books and publish them to a Sophie Server. Then they all read each others' books (through their own copy of Sophie Reader, which downloads the books from Sophie Server). Using Sophie Reader, they make comments on each others books; these comments are uploaded to Sophie Server. When everyone has commented on everyone else's work, the original author can look at the original book and see all the comments that everyone else wrote.

A travel guide company could administer a Sophie Server and use it to publish a series of books on destinations. Travelers could download the books into their mobile version of Sophie and use it as a guidebook. However, it's an interactive guidebook: users could also add their commentary or change the listings entirely. They could also upload related media – photographs taken while at a specific place, for example. A chat function inside one of these books would allow the user to see who's actually at the place they're visiting when they're there.

WHERE SOPHIE CAN GO: EXTENSIBILITY

Sophie is designed to be extensible. This isn't something most users of Sophie will use, or need to think about. But it's a major part in the Sophie philosophy and a major argument in its favor: extensibility will ensure that Sophie books will be readable ten, twenty, or fifty years from now. Extensibility's something that's happening under the hood of Sophie whether you're using it or not. It's also something that happens on several different levels: different kinds of extensibility are possible, depending on how deep inside of Sophie you're willing to go – even if you're not a dedicated Smalltalk coder you'll probably be able to use our scripting language.

Here are the various ways you can extend Sophie, arranged from the simplest to the most powerful.

SCRIPTING

Out of the box, Sophie can do a lot: it will probably suit the needs of the majority of users. If you're using Sophie on a grander scale, however – making a slideshow of 5,000 images, for example – you might want to automate some of the work rather than doing it all manually. Or you might want to make some kind of a book that's different from anything that we've thought of making with Sophie: consider, for example, a book in the form of a randomly shuffled deck of cards – going to the next page takes you to a page chosen entirely at random. A scripting language for Sophie – being developed for release shortly after Sophie 1.0 – will make both of these sorts of things possible. Scripting Sophie will be the simplest way to extend a book.

THE API

Why is the Internet particularly interesting right now? A lot of the answer is because of APIs. The API is an interface that lets one web application talk to another application; using APIs, web sites can be connected by savvy users, letting them use Google Maps to display the location of apartments on Craigslist.

Sophie's API will be released at the same time that Sophie 1.0 will be released. What does it mean to have book-making software with an API? It means that you can connect books to databases: you could generate books from databases, or you could take information from Sophie books and put it into a database. You'll be able to use Internet services from inside Sophie: to use an example we've already mentioned, you could include Google Maps inside of your book. A Sophie book could include content from a blog, or a blog could contain live-updated information from a Sophie book.

PLUGIN-BASED ARCHITECTURE

Because Sophie's written in Smalltalk, it has all the advantages of Smalltalk, a fully object-oriented programming language. Sophie is constructed modularly. This means that it's easy to attach new functionality to Sophie through plugins. While Sophie 1.0 won't ship with a text engine that handles right-to-left type (like Arabic or Hebrew), a plugin could easily be made that will handle those kinds of languages; when installed, this capability becomes part of Sophie. It's also possible to replace parts of Sophie to build a custom version .

OPEN SOURCE

And finally, it should be clear that Sophie is open source software. You're free to take the source code to Sophie and reassemble it as you like. Should you wish, you can distribute your own proprietary version of Sophie.

The Institute for the Future of the Book will maintain the Sophie code base and release new iterations of the program regularly.