***eHyperTool***

**Design Specification**

Version 1.02

Richard A. Bartle

Beck & Bartle

August, 2011

**Business Requirements**

This section describes the parameters within which the product design must operate.

**Overview**

This is a design specification for an authoring tool (*eHyperTool*) that is to be used for creating interactive, hypertext stories. In these stories, the reader is presented with a situation in the form of a piece of narrative text; this text could be anything from a few lines to many pages in length. The user is given a number of options (in general, between two and five inclusive) for how the story proceeds. Based on the user’s selection, the resulting situation is described, and the process repeats. It ends when there are no options available because the narrative has reached its conclusion.

The tool is used to assist the authors of hypertext stories in three ways:

* It does all the housekeeping necessary to ensure that the stories are complete and consistent. Common mistakes such as forgetting to complete a narrative thread or building closed loops the reader can’t escape from are impossible in this system.
* It reminds the author of the options available to them, so that they write to the same standard all the time. They don’t go away on holiday and then return only to write to a different format.
* It can export “camera-ready” output that can be uploaded directly to a digital story platform. Authors may wish to leave this to editors, however, so it also has its own, flexible format for pre-publication work.

In addition to regular hypertext, *eHyperTool* should be able to create stories that include simple audiovisual materials – static images, sound effects, music, voice over or whatever. Most of the time it will indeed be straight text, but some products will benefit from additional materials (such as line drawings in stories for children).

Although at a later date the software could be extended to be used for other purposes that involve moving around virtual index cards (such as keeping track of screenplay scenes, plot elements and characters), right now it’s only intended to be used for creating hypertext stories.

**Users**

The users of the system will in the main be story authors whom we can regard as intelligent non-technologists[[1]](#footnote-1). Although they are not scared by technology, they may find too many complications daunting. It is therefore important to:

* Keep the number of options obviously available at any one time low. Additional functionality might be appreciated by advanced users, but it will alarm people with no affinity for technology who just want to write stories. If we do add sophisticated but rarely-used features, we hide them.
* Document everything. This means tooltips and help in the program; outside the program, it means (eventually) producing a manual and creating a user forum for authors to help one another.
* Make the interface as intuitive as possible. Use single-clicks where possible. Keep the screen uncluttered. Allow the visual display of structures to be minimised.
* The look and feel should be non-threatening and professional. We don’t want a science fiction chrome appearance, but we don’t want to look like a candy bar, either. If the look is too individual, it could influence how users feel about the book they’re using it to write. It may be a good idea to get a graphic designer in to help with this...

What we have here is a program that will spend 95% of its time doing the same, simple things, with the remaining 5% of its time spent doing more complicated things. We want the users to have access to the complicated things, but not in such a way that it puts them off using the simple things. Also, we don’t want those simple things to appear complicated.

**Technical Requirements**

The stories created by *eHyperTool* are to be formatted such that they will work on a number of common platforms, with the Amazon Kindle the primary target and the Apple iPad the secondary target.

The *eHyperTool* program itself is to run in a cloud environment accessed over the Internet, rather than on a home computer. This means that cross-platform compatibility is ensured, as users only need access to a browser. It also means that *eHyperTool* itself does not need to be digital rights-managed, because users will not have their own private copies of it. However, users will need a user name and a password so as to be authorised to use the server. Additionally, administrative support must exist (as a separate program) to add/remove users.

The tool should be regarded as having “white box” potential, in that separate servers can be set up that run rebadged instances of *eHyperTool* to suit the needs of individual licensees (for example publishers).

**Design**

This section comprises the bulk of this document, and concerns the design of the *eHyperTool* product itself.

**Definitions**

There are some technical terms that we will need in describing hypertext fiction:

* **Author.** An author is the person who creates the content of a hypertext story. Insofar as our software is concerned, they are a *user*. The author could also be the *publisher*.
* **Challenge Text.** The prompt that a *reader* is given introducing an *option* set.
* **Consequent Node.** The *node* that a *link* points at.
* **Guard.** Some *options* only appear in a *node*’s description if they satisfy certain conditions regarding the nodes that have (or have not) previously been visited by the *reader* (ie. whether or not these nodes are present in their *narrative history*). Such conditions are called guards.
* **Guard Link.** A *guard* refers to a *node* that has (or has not) previously been visited. The reference of the guard to that node is the guard link.
* **Link.** A link is a unidirectional connection between one *option* of a *node* and the node to which the *reader* progresses if they select that *option* (the *consequent node* for that link).
* **Narrative History.** The path from the Start *node* of a story to the current node, as undertaken by a *reader* of the story.
* **Node.** A node is the main element of a hypertext story. It is the narrative text (and associated images and sounds) that describes a situation or plot point concerning which a *reader* has to make a decision. These decisions are expressed as *options* which *link* to other nodes that continue the story.
* **Option.** Following the descriptive text of a *node*, the *reader* is presented with a number of decisions to make. These are expressed as a series of options. Each option acts as a hypertext *link* to another node known as its *consequent node*.
* **Player.** An individual reading (“playing”) the story output by *eHyperTool*. The term *reader* is preferred in this document.
* **Publisher.** A publisher is a person who handles the production and style details of a hypertext story. Insofar as our software is concerned, they are a *user*. The publisher of a work could also be the *author*.
* **Reader.** A reader is a person who uses the interactive fiction that is the output of *eHyperTool*. A reader is also commonly referred to as a *player*, but the term reader is preferred in this document.
* **User.** A user is a person who uses our product, either as an *author* or *publisher*.

**Overview**

There are two potential users of this software: the author and the publisher. In some cases, the same individual will be both author and publisher (*ie.* they will be self-publishing). The facilities for both authoring and publishing therefore need to be written.

From the user’s perspective, the tool we are creating has the following order of coarse functionality:

* Start a new project [author or publisher].
  + Set the default features that will persist for this project – name, author, style and so on.
* Import components [author]
  + Optional: existing stories or parts of stories can be read in.
* Create the story [author]
  + This is the major part of the work, described shortly.
* Produce front cover [publisher].
  + Optional: the user can have an image for the front of the book.
* Produce front matter [publisher].
  + Optional: the user can include acknowledgements, a foreword and so on.
  + Optional: the user can create the copyright and library cataloguing data, publisher’s website and so on. Possibility: the program can do this automatically using information provided by the licensee.
* Produce back matter [publisher].
  + Optional: the user can include an appendix, bibliography or other standard end-of-book text.
* Produce back cover [publisher].
  + Optional: if the material is of a kind that can be printed as a physical book (ie. it uses no timers and no option guards) then it may need a back cover, too.
* Export readable story [publisher].
  + Create the native “executable” files that can be accepted by the target platform.

The majority of the above are simple form-filling exercises that could be done in a regular word processor; we can regard them as “production” rather than “development”. The value-added component that *eHyperTool* brings is in the area of story-creation, and, by implication, project-creation and story export.

Story-creation progresses through a basic three-part loop:

* Author looks to see what story nodes are yet to be written or need to be edited, and selects one.
* Author writes or edits the story node.
* Author checks that the story node will appear as intended to the reader.

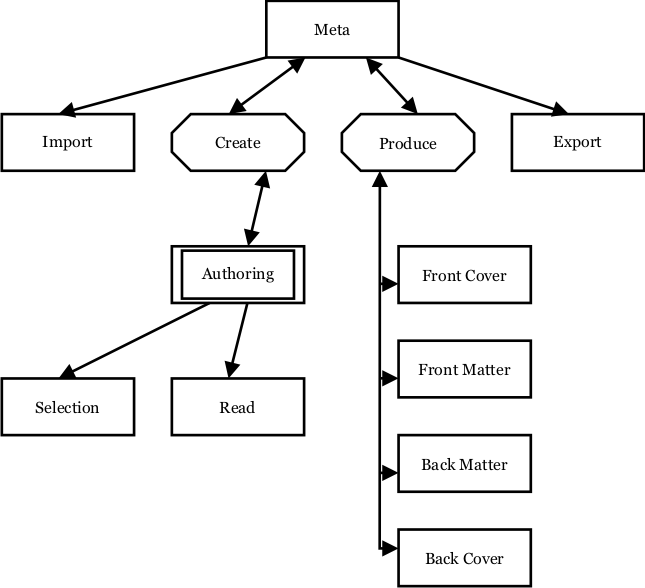
This means there are three different modes in the story-writing part of the system:

* Selection mode – the author is selecting a node to work on.
* Authoring mode – the author is actually creating a node.
* Reading Mode – the author is running their text as a reader might.

We are now in a position to look at how all the components of the product hang together.

**Architecture**

Here is a diagram indicating the functional relationship between the various components of the system. Note: this is in terms of views – it’s not a class diagram.



The rectangles in this are physical views; the octagons are logical collections of views; the double rectangle is a physical view that the user may want more than one of open at the same time.

In theory, each view could be displayed in one of three ways:

* As a window.
* As a tab on a window.
* As a pane within a window.

In practice, having three display possibilities for each view is an unnecessary indulgence which would confuse the users.

Looking at the relationships, anything that has a unidirectional arrow should appear in its own, separate modal[[2]](#footnote-2) window; once the task is complete, control returns to the originating view[[3]](#footnote-3). For the octagonal boxes, they should represent a single window with the boxes under them as tabs (or, in the case of double rectangles, multiple tabs).

The result of this is:

* Meta is the main window. It opens Import and Export as modal windows, and loads Create and Produce views into its window.
* Create is tabbed for multiple instances of Authoring views that the user can add or remove.
* Authoring views open Read and Selection as modal windows.
* Produce is tabbed, with tabs predefined for Front Cover, Front Matter, Back Matter and Back Cover. The user can’t add or remove tabs.

Not shown on the above diagram is the Help view. This is orthogonal to all other views, and opens as a separate, non-modal browser window.

We can now look at each view in more detail.

**Import**

When *eHyperTool* is run for the very first time, the user is asked to enter their name and password. In addition to the security this offers, it also means that if more than one person works on a project, their contributions can be tracked by name[[4]](#footnote-4). The user has the option to log in automatically from that computer in future (while the cookie lasts).

If the user has already created a project, the Import window will not appear; the last-read project will be loaded automatically, open in Authoring view at the last-used node. The user can create a new project from the Authoring view menu bar if they so wish, or load a different existing project (which may be a back-up copy of a project).

Future versions of *eHyperTool* may allow the user to import and export projects held locally on their own computer, but at launch we only need to store projects on the server.

A project-creation window (whether for a new project or the reloading of an existing project) is a form that asks for or displays the following information:

* Author (defaults to user name, but can be changed here *eg.* to a pen name).
* Project title (*eg.* “The Mystery of the Sightless Sculptor”).
* Project type (one of Book, HTML or eText; these are explained shortly).
  + Multimedia use (this depends on the project type).
* Challenge text (the default prompt the reader will see before the options block, *eg.* “Do you:”, “What happens next?”, “What should Miss D’Arcy do?”).
* Time and date of project creation (not editable).
* Time and date of last change to project (not editable).
* Publisher (not editable; if this is the vanilla, unbadged version of *eHyperTool* then it will be the user’s own name).
* Comment (space for the creator to make notes about the project – see the later General Note regarding its functionality).
* List of names of extant nodes (there will only be one for a new project, the automatically-created Start).

There is also the future possibility that we will have a number of styles and templates available, which are to stories what cascading style sheets are to web pages. However, at the moment we can get away without this level of sophistication.

The project type enables the user to create projects that the output of which will run on specific media. The type can be changed at any time; it’s basically just a simple way to unclutter the interface for projects that don’t need the full power of the system. The simplest form is Book, followed by HTML, followed by eText. The differences are as follows:

* Book: no sounds, no randomness, no timers, no guards.
* HTML: no guards.
* eText: full functionality.

There are also two options dependent on the project type:

* Images (available for all types; check this if the body of the work – not the front matter or other production components – contains images)
* Sounds (available for HTML and eText only; check this if the body of the work contains sounds).

The purpose of these options is so that the user isn’t continually asked to input sound files or whatever when there are never going to be any.

It is important to note that the project type only affects display. If you convert an eText to a Book and back again, you won’t lose any sound references; Book-type projects merely don’t *display* the sounds associated with nodes, they don’t delete them.

Option: new or existing projects can import nodes from other projects. This allows partial works (for example a section written by another person) to be absorbed into a larger work. In the event of name clashes between imported nodes and existing nodes, the user is given the following choices:

* Rename the imported node. All references to this node in the imported project will be updated to use the new node name (the default being the old name with a number appended to it in parentheses, starting at 2 and rising until there is no clash with nodes in the existing or imported projects[[5]](#footnote-5)).
* Use the original node and discard the imported node.
* Use the imported node and discard the original node.
* Use the text of the original node and the options of the imported node.
* Use the text of the imported node and the options of the imported node.
* Use the text of the original node and the options of both nodes (original first).
* Use the text of the imported node and the options of both nodes (imported first).
* Use the text of both nodes (original first) and options of both nodes (original first)[[6]](#footnote-6).

Later versions of this program may allow for projects to include customisation text. For example, at the start of a story a user could be asked for their name, which is stored in a variable (say, %name). The variable could be inserted into the body of the story, to be replaced by the value entered by the user at run-time. This is not something we need at launch, though.

General Note: the comments section of the project view is basically a box into which the user can enter freeform text. There are many boxes in this project that allow freeform text entry, particularly in the Authoring view. These boxes should all be editable and have as many spell-checking and other text-friendly features as can reasonably be added. It is nevertheless expected that some users will prefer to use a more powerful word processor with which they are familiar. At the moment, they can do this but will have to cut and paste to get text from the word processor into the text boxes. A courtesy solution would be to allow the user to specify (from the menu bar) which word processor to use, and allow them to open up that word processor in a separate window in order to edit the text in a box. Closing the window would store the text into the box. An even better solution would be to embed the word processor in the text box itself. At launch, we don’t need to do either of these; however, it is something we are likely to need later when we start to pick up authors who are not tech-savvy.

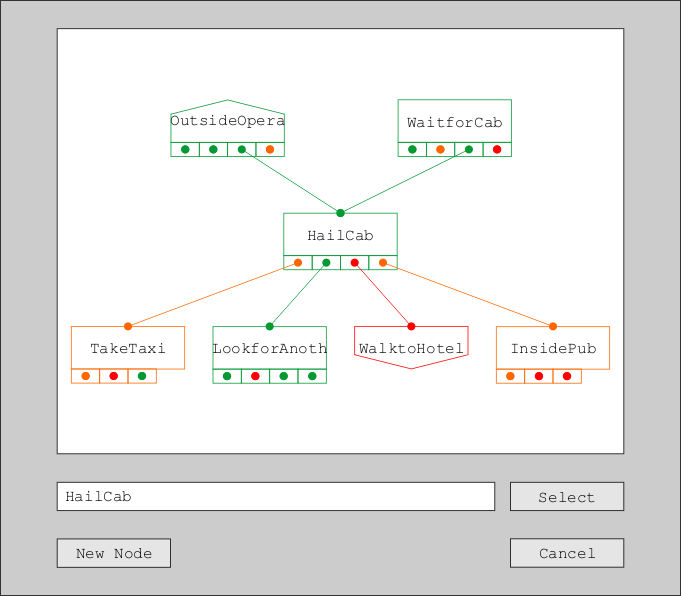
**Selection**

The purpose of the selection view is to assist the user to choose a node upon which to work, based on a visualisation of that node in terms of its relationship to other nodes. It’s actually only one of many ways for the user to select a node (see the Authoring view for the other ways). The Selection view is a modal window (*ie.* the rest of the program is not accessible until it is closed, as with a file-selection window) that pictures the current node and its immediate “family” of nodes. It is opened either from a new (empty) project-creation view or from an Authoring view. It returns either an existing node, a new node, or nothing (the user cancels the selection).

The selection process has the following functionality:

* It displays nodes symbolically, along with summary identifying details.
* It displays links between nodes symbolically.
* It can display nodes connected either by regular links or by guard links[[7]](#footnote-7).
* It can be zoomed-in or zoomed-out to give a higher or lower degree of visual fidelity. Not all the functionality described here need necessarily work at all zoom levels.
* The current node is identified in some way (*eg.* location on the screen).
* Nodes indicate the author’s view of their status (*eg.* icons, different background colours[[8]](#footnote-8)).
* Nodes indicate the program’s view of their status (*eg*. different border colour).
* The Start node is identified in some way (*eg.* different shape).
* End nodes are identified in some way (*eg.* different shape).
* Unattached nodes are identified in some way (*eg.* different shape).
* Nodes can be searched-for by name or by their text content.
* New nodes can be created. They can only be connected to the existing nodes through the Authoring view, however.
* More details about a node can be obtained without having to close the window.
* A selected node can be chosen to be the value returned to the calling window, at which point the window closes.
* The selection process can be cancelled with the result that no node is returned to the calling window.

Here is an example of what this might look like at minimum zoom:

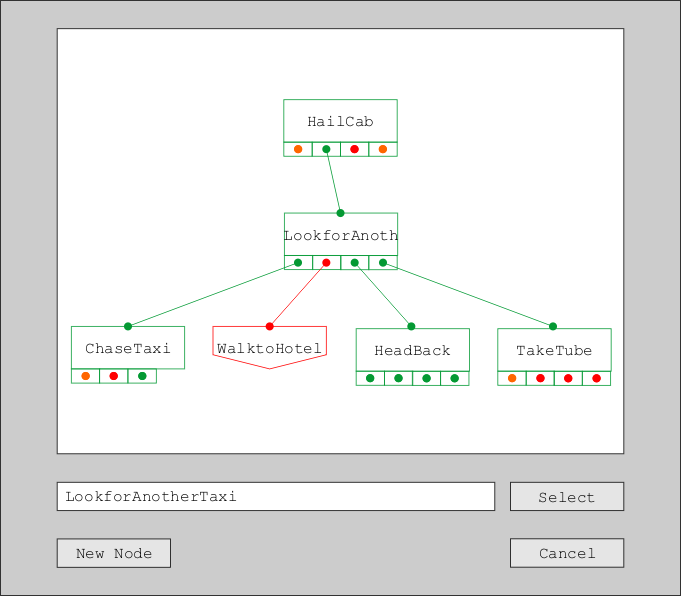


The interface for the selection window is to some extent operating-system specific. For Windows, this means:

* Zooming in/out is with the mouse wheel.
* Clicking on a node redraws the view centred on that node. It becomes the view’s “current” node.
* Clicking on a “Select” button when a node is selected closes the window and returns the current node to the calling window.
* Double-clicking on a node closes the window and returns that node to the calling window.
* Clicking on a link source redraws the view centred on the consequent node of that link.
* Mousing over a node gives more details about it[[9]](#footnote-9).
* Links from a node are displayed in the order they appear in the node’s options list[[10]](#footnote-10).
* Typing into the text box auto-completes with the next match found; hit return to accept what it currently shows, which will centre the view at the existing level of zoomedness on the newly-selected current node.
* When not being typed into, the text box shows the name of the currently-selected node.
* New node creation is by a “New Node” button click. The new node is created using the name currently typed into the text box (which must be unique).
* Clicking on a “Cancel” button closes the window and does not return a node. Other operating-system specific mechanisms can also do this (*eg.* clicking an X on the window’s tab).

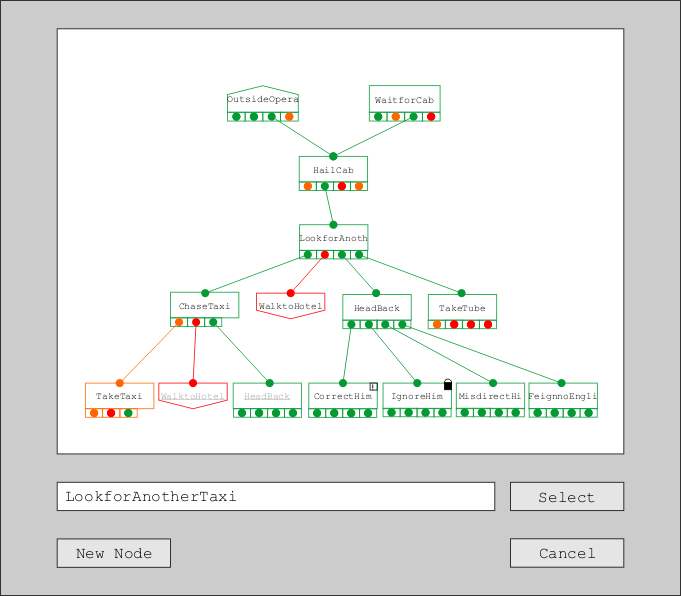
There is no provision for users themselves to move boxes around on the canvas for whatever aesthetic or readability reasons they may have, regardless of how legitimate they are. However, if it doesn’t make a lot of difference to the overall complexity of the rendering mechanism, hey, go for it.

Here is another view of the node set above, assuming that the user has clicked on the LookforAnotherTaxi node:



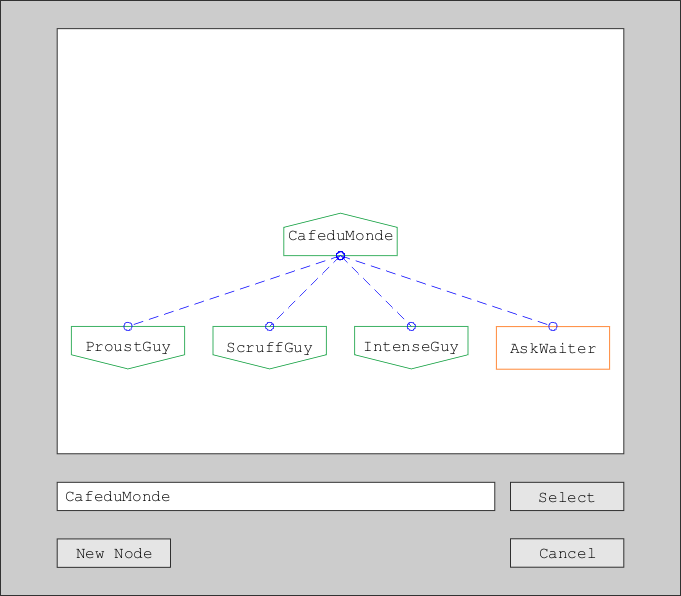
LookforAnotherNode can currently only be reached via the second option of HailCab. Its outline is green because it is viable (see the section on the Authoring view for a list of possible node statuses). If it had been partial, its outline would have been orange. WalktoHotel has a red border because it is empty. It also has a shape that says it is an end node (it currently has no options – all empty nodes are end nodes, but not all end nodes are empty). The central node, LookforAnotherTaxi, is the “current” one, which will be selected should the user click on the Select button. Only the current node and its immediate family are displayed: parent nodes (only HailCab in this example) are ones that have this node as a consequent node; child nodes (ChaseTaxi, WalktoHotel, Headback and TakeTube) are ones that this node has as a consequent node. Other nodes are not displayed, but their nature is indicated by the colour of the link ends associated with them; for example, the first link from HailCab is orange (because, as we saw in the first view, it points to the partial node TakeTaxi). Clicking on a link end will recentre the view on that node (so clicking on the first link from HailCab above will draw the view with TakeTaxi in the centre – even though TakeTaxi itself doesn’t directly appear in the above).

Sometimes, the same node will appear more than once in a view. Here is the above view zoomed out a level to show grandparent and grandchild nodes:



The nodes WalktoHotel and HeadBack appear twice[[11]](#footnote-11), as they are children of both LookforAnotherTaxi and ChaseTaxi. They are identified as being duplicates by having their name in grey and underlined[[12]](#footnote-12). Although the children of TakeTube *should* be displayed, there isn’t room to display them so they are left unexpanded[[13]](#footnote-13). The node CorrectHim has an exclamation mark on it to show that its user status is “marked”; the node IgnoreHim has a padlock symbol on it to show that its user status is “locked”.

The view for guard links is the same kind of thing, but from a guard link perspective. Here is an example:



This means that CafeduMonde is itself a guard for no nodes. ProustGuy, ScruffGuy and IntenseGuy have no guards themselves but are guards for CafeduMonde. AskWaiter, a partial node, is a guard for CafeduMonde and (we know because of its shape) has guards of its own; you’d have to click on it to see what they were, though.

The guard view is something of a luxury, and need not be present at launch.

General Rule: whatever the view, selecting a node in a particular way (*eg.* double-clicking in Windows) will open the current tab of the calling window on that node. It is also possible (*eg.* right-clicking and selecting from a functionality list) to open the node in a new tab.

**Authoring**

The Authoring view is at the heart of *eHyperTool*. It is multi-tabbed, each tab showing the structural details of a single node[[14]](#footnote-14). There is no such thing as an empty Authoring view: they always open on nodes that already exist. Only one such node exists by default: the *Start* node.

A great deal of functionality can be associated with the Authoring view – feature creep is a real danger here. Also, the more functionality there is on display, the harder it is going to be for people with no great technical expertise (*eg.* authors) to make sense of even basic tasks. As our aim is for the system to be accessible, we should therefore keep the interface as clean as possible while allowing easy access to more sophisticated functionality should the user require it.

The Authoring view has to be able to do the following basic things:

* Accept a title for the node.
* Accept a background sound for the node.
* Accept a node description.
* Accept a set of user options.
* Accept a closing remark for the node.
* Edit and display meta-information about the node.

Let’s break these down in more detail.

A **node title** is a human-readable summary of the node. It could be a location, a time, a decision, a person – whatever the user decides. It does not have to be unique: two or more nodes can have the same title[[15]](#footnote-15). Titles can not be empty, however[[16]](#footnote-16).

A **background sound** is an optional music file that will be played when the node is visited by the reader. It could be music, it could be atmospheric noise (a crowded pub, for example), it could be the text of the node read out loud – whatever the user determines. Nodes do *not* have to have a background sound. It should be possible for the user to propagate the sound file to the nodes that this node’s options currently[[17]](#footnote-17) link to; this is to make it easier for the user to carry over a common sound effect between nodes (*eg.* several nodes all set in the same crowded pub). Note that background sounds may not be displayed (and therefore may not be added) for particular project types, for example Book projects.

A **node description** is a series of *description elements*. A description element is either a non-empty text box or an image file. One text box is the *primary* text box, and does not have a guard condition – it will always appear for this node (and will be the only box present when a new node is created). All other elements can have a guard condition (but don’t *have* to have one). New elements can be added at the start or end of the node description[[18]](#footnote-18). Guards are described shortly. Note that guards may not be displayed (and therefore may not be added) for particular project types, for example Book projects.

A **user option** is what the reader can do in response to the challenge text (which should also be displayed, once, for the whole set of user options). User options are the most complicated part of the Authoring view. They have the following components:

* An optional guard (described later).
* An optional timer. A timer is a way of making an option appear or disappear after a number of seconds. Most nodes will have no timer, but those that do will either be available only for a certain period of time or become available only after a certain period of time[[19]](#footnote-19). For example, an option to get into a waiting taxi might only be available for 10 seconds; a matching option to wait for a taxi might appear after 10 seconds.
* An optional sound effect. This will be played (over the background sound) when the node is selected by the reader.
* A textual description of the option. This acts as the hyperlink text, so that when the reader selects it (“clicks on it”), the link’s consequent node is loaded.
* The consequent node to which this option links. If this node does not exist when its name is added, it is created as a new node (but not yet opened).

The options are listed in the order that they will be presented to the user. New options can be added at the start or end of the option set[[20]](#footnote-20). Note that guards, timers and sound effects may not be displayed (and therefore may not be added) for particular project types, for example Book projects. Note also that unlike the case with node descriptions, it is possible for a user to create a set of options with guards set up such that none of the options display. Such situations will be caught in the sanity check that precedes story exports (described later).

A **closing remark** is the same as a node description, except that it appears after the user options have been displayed and does not have a primary text box. It can have multiple elements, guarded or otherwise. Nodes do not have to have a closing remark.

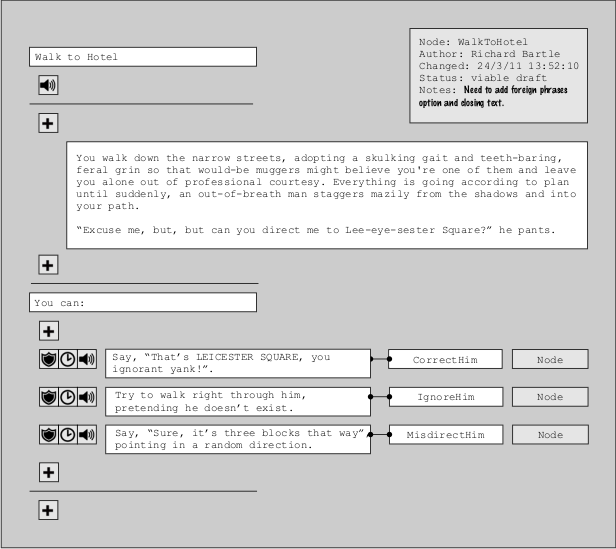
**Meta-information** is information about a node. It’s useful to the user, but none of it is ever seen by the reader of the resultant work. Some of it is editable by the user and some of it is not. It has the following components:

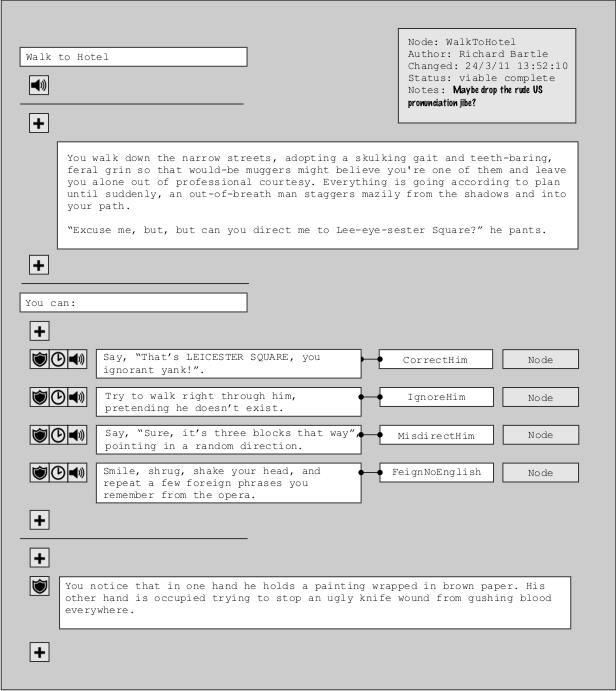
* **Node name**. Editable. This is a unique identifier for the node. It defaults to the node title with non-alphanumerics removed (node title “Outside Pub” would default node name to “OutsidePub”). If there already is a node with that name it appends the number 2 in parentheses to the end of the name, and if this also already exists it increments it until it finds one that is different (say, “OutsidePub(4)”). As outlined in the General Rule presented in the Selection section, any time a node name is selected in the right way (*eg.* by double-clicking) means the current view will be replaced by that of the selected node[[21]](#footnote-21).
* **Node author**. Editable. This defaults to the current Project author, but can be altered for individual nodes.
* **Last change**. Not editable. This is a time-stamp recording when the node was last changed[[22]](#footnote-22).
* **Status.** Part-editable. A node’s status comes in two parts: hard and soft. Hard status is determined automatically and is one or more of:
  + **Empty.** No content. This is the default when the node is created.
  + **Partial.** Some fields have been completed but some have not.
  + **End.** The title and/or the description elements have been at least partially completed, but the options are all empty.
  + **Orphan.** The node has no nodes linking to it. Only one orphan node is allowed in an exported story (the Start node, which doesn’t *have* to be an orphan – it wouldn’t be if one of its descendents links to it[[23]](#footnote-23)).
  + **Viable.** Sufficient information has been provided for the node to be acceptable to the reading software (subject to sanity checks). Multiple end nodes can be viable as well as being end nodes, but only the Start can be both viable and an orphan.

Soft status is chosen by the user and is one of:

* + **Blank.** The user hasn’t given the node a status.
  + **Marked.** Can be applied to any node. This is so the user can search for it easily later.
  + **Draft**. Can be applied to any node.
  + **Complete.** Can be applied to any viable node.
  + **Locked**. Can be applied to any viable node. This write-locks it; to unlock it, the user sets its status to blank, marked, draft or complete.
* **Notes.** Editable. This is where the user can add personal notes and reminders concerning this node (or anything else!).

Here are two mock-ups of an Authoring view of a simple node at different stages of completeness:





The components of this are as follows:

* The first box is the node's title, in this case Walk to Hotel.
* The big grey button is the status box. Users can click on this to alter some of the details (for example to change the notes).
* The speaker button as it stands means there is no sound (*eg.* music) associated with this node. The user can click on it to open a pop-up which will add sound[[24]](#footnote-24). The button will then change (*eg.* its colour) to indicate that its feature is active.
* A plus button means that a new element can be added here. The first one we see means the new element would be added before the primary text element; the second would mean it appears after it. Click on the plus to add a new element. A pop-up will appear asking whether you want it to be an image or text[[25]](#footnote-25). Depending on your reply, it will either open up an image-selection window or create an empty text box.
* The main block of text is the node’s primary description. All nodes have one of these and it can't be deleted. Other text boxes that have been created can be deleted from the menu bar or by right-clicking and selecting Delete[[26]](#footnote-26). The text box will increase in length if the text gets long, but beyond a certain[[27]](#footnote-27) length will switch to scroll bars; this is because some books could have many, many pages between reader decisions.
* The You can: line defaults to whatever the challenge text for this project is set to be. The user can change it for this particular node, but it is not recommended (for reasons of immersion: if the reader sees the same prompt every time, they will recognise it as the standard signal in this work that means options follow and they won’t register it as something “different”).
* The option lines have three components to them: a bank of buttons, the option text, and the node you go to when (as a reader) you select the option.
* The guard symbol next to an option is for if you want to add a guard. This particular example has no guard, but if it did have one then the button would change appearance to indicate this.
* The clock symbol is for timers. If you want an option to appear or disappear after a certain amount of time, you click on this to set it. The time to appear or disappear is in seconds[[28]](#footnote-28).
* You click on the speaker symbol if you want a sound effect to be played when the reader selects this option, *eg.* a creaking door. It opens a sound-selection window, as with the speaker symbol for the node as a whole, but the sound file is particular to this option.
* The first box contains text for the option. The box will increase in length to accommodate the text[[29]](#footnote-29).
* The second box contains the name of the node that selecting this option will take you to – the consequent node. You can either type the node name in directly or select/create it from the Node button. If you type it in, the system will predictively fill it so you only need type a few letters to pick up a long node name, and you will also know if you mistyped (so won't easily create a new node by accident). The Node button opens the node selection modal window so you can choose (or create) a node that way.
* The plus button at the end of the first example is for text or images that follow the options. In the second example, some text has indeed been added. The guard symbol next to the text means that this can be made subject to a guard, but in this particular event it hasn’t been – the text will display every time.

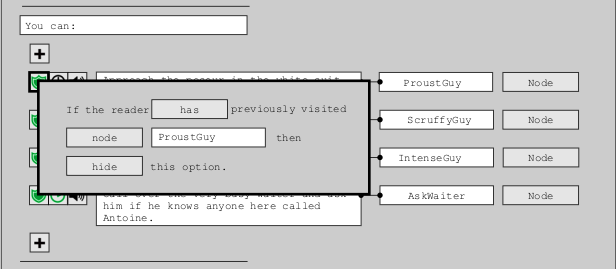
A **guard** is a condition, the purpose of which is to ensure that a description element or option is only displayed and available to the reader under certain circumstances. The circumstances concern only the narrative history of the reader of the story in reaching the node in question. Basically, a guard names a node, and depending on whether the node has or hasn’t previously[[30]](#footnote-30) been visited by the reader while following the story, the guard will either block or allow progress. It is very easy to make guards far more complicated than this, but at launch this is going to be sufficient.

The reason that guards are necessary is to stop the needless and tedious repetition of almost-similar nodes by allowing for “hub” nodes. For example, suppose you (as the reader) are supposed to be meeting someone in a café and there are four people there you could meet. Only one is the right one, but you can approach them in any order. At this point, your options are to try speak to A, B, C or D. Suppose you spoke to A, but this was the wrong person. You should now have options B, C and D. Likewise, if you chose B you would have options A, C and D; if you chose C you would have A, B and D; if you chose D (the right person) then the story can continue. Thus, there are three nodes there that could really just be one. It’s actually worse than that, because if, having chosen A, you were then to choose B, you would need a node for C and D; if you chose C, you would need one for B and D. The same sort of thing applies for if you chose nodes B or C first. It’s much easier if you can have just one node which removes options you have already taken (*ie.* which are in your narrative history). That’s what guard conditions do.

Here is a mock-up of a screen that might be that of an advanced user, which incorporates guards:



The buttons that are in a different colour are active. We can see that there is a sound file associated with this node (the user would have to click on it to hear the associated sound[[31]](#footnote-31)) and a timer on the AskWaiter option (the waiter will go away if you don’t quickly catch his eye). There are guards on all four options and on a secondary element of the node description text. To set, edit, or display a guard, the user clicks on that guard’s button. Here is an extract from a mock-up of what might happen if the user did that on the guard of the first option:



What this is saying is that the option Approach the poseur in the white suit who is ostentatiously reading Proust. will not appear if the user has previously visited the node ProustGuy. As ProustGuy is the node that this option itself leads to, what it’s basically saying is that if you (the reader) have already tried this, you won’t see it again as an option but you will still see the other options – unless their guards have also kicked in. Note that there are two ways that the same guard could be phrased: the above is If the reader [has] previously visited [ProustGuy] then [hide] this option. (otherwise display it); alternatively, it could be If the reader [hasn’t] previously visited [ProustGuy] then [display] this option. (otherwise hide it). Either will work – it’s up to the individual user as to which they prefer.

The default values for the guard pop-up are has and hide, as above. However, others are possible. The guard on the The waiter frowns at you with an air of haughty disapproval. element is If the reader [has] previously visited [AskWaiter] then [display] this element.[[32]](#footnote-32).

Note that at the moment there is no provision to put in place a guard based on whether or not a timer triggered. We can’t have the waiter permanently ignore you on the basis of your not having attracted his attention soon enough earlier. This may be something we could add to a later version if it’s something that users request.

There are some menu bar[[33]](#footnote-33) functions also required for Authoring view:

* New project. Allows the creation of a new project. If the current project has been changed since its last save, an option is given to save the current project first.
* Save project. Saves the current project in a format that the program can read back in. Because this is cloud-based, we can save every few seconds anyway, but users will feel more secure if they can force a save. Note that this is not the same as exporting the project, which writes it in a format to be read separately.
* Load project. Loads a saved project. As with new project, an option to save appears if the current project has been altered since the last save. We’ll need a permissions system if users are to be allowed to load each other’s projects.
* Edit project details. Allows the user to change the details of the current project (see the Import section for a list of which details can and can’t be edited).
* Produce. Opens the Produce view (for adding front matter *etc.*).
* Enter Reading Mode. The user takes on the role of reader for testing purposes.
* Select node. Open the Selection view pop-up to find a node to open in the current tab.
* Delete node. Removes the current node and reloads the one that was previously in the tab (or if there wasn’t one, opens the Selection view centred on the Start node). The node is removed entirely: all references to it from other nodes are removed. This will invalidate those other nodes. Note that the Start node can not be deleted.
* Home. Opens the Start node.
* Browse. Allows the user to revisit a node that was displayed in the tab previously (assuming it is still extant). So long as they don’t load another node, they can subsequently browse forward again to a later node – even if they edited the earlier node. Note that this is nothing to do with narrative history – the user should use Reading Mode for that. We may not need this option, as the users will probably use their browser’s native browsing capabilities and won’t expect that a change to a previous node won’t clear the browsing cache for nodes that follow.
* Copy node. Copies the contents of the current node as they stand[[34]](#footnote-34).
* Paste node. Replaces the contents of the current node with that of the last-copied node. If the current node is non-empty, the user is warned that this will destroy its existing contents[[35]](#footnote-35).
* Undo/redo. Although the operating system may handle this for some situations automatically (in text boxes, for example), we also want to be able to undo/redo structural changes that have been made such as deleting a node or pasting over its contents.
* List history. Open a static list of the names of all extant nodes that the user has in their browse history for the current tab. Selecting one of these will load it into the current view. Again, we can probably ignore this and use the browser’s own history system.
* List parents. Open a modal window containing a static list of the names of all extant nodes having one or more options that link to the current node. Selecting one of these will load it into the current view.
* List children. As list parents, but it lists all nodes that the current node links to (*ie.* that are in the options section).
* List guard parents. As list parents, but it lists all nodes that have a guard condition mentioning the current node. The current node itself does not appear in this list.
* List guard children. As list parents, but it lists all nodes that the current node mentions in guards. Again, the current node itself does not appear in this list.
* List marked nodes. As list parents, but it lists all nodes that the user has marked.
* List draft nodes. As list parents, but it lists all nodes to which the user has given draft status.
* List locked nodes. As list parents, but it lists all nodes that the user has locked[[36]](#footnote-36).
* List empty nodes. As list parents, but it lists all nodes that have been created but not yet filled in.
* List partial nodes. As list parents, but it lists all nodes that have been partially filled-in.
* List end nodes. As list parents, but it lists all non-empty nodes that have no options.
* List orphan nodes. As list parents, but it lists all nodes that have no parent nodes. This will always include the Start node.
* Search for node. This opens a pop-up duplicating the Selection view search facility.
* Search for text. This opens a pop-up allowing the user to search for a node based on the text in the node’s title, description, options and remarks[[37]](#footnote-37). We may also want to allow a replace facility, so the user could (for example) replace all occurrences of Mr Smith with Mr Jones.

There is other functionality we could also add – search by sound file and image file, for example – but the above is sufficient for launch[[38]](#footnote-38).

One final piece of functionality is that if the user types copy (^C for Windows) on a node view (but not while focus is in a text box), or right-clicks on a node name and selects Copy, then the node name is copied to the clipboard. This means that users can paste node names into link boxes when they’ve visited the target node or seen it in a list.

**Reading Mode**

Reading Mode allows the user to experience the story as a reader would.

Although ideally there should be a Reading Mode for each project type, initially we don’t need to have this. The purpose of Reading Mode is to make sure that the story flows well, that nodes link to where they should link, and that there are no obvious plot holes; it is not intended to be used to test the visual aesthetics of the final story, which is platform-dependent (it would not look the same on a Kindle as it would on an iPad). That said, Reading Mode does respect the project type of the current project: if you have set up a Book project, for example, then Reading Mode will not play any music you may have associated with nodes (which you could have put in place before switching project types, for example).

The basic functionality of a hypertext story reader using our story format is as follows:

* Subject to a guard: if the node has a sound file associated with it and the sound file is not the same as the sound file already playing, then the current sound (if there is one) is faded out and the node’s sound is played instead; if the node has no sound file but sound is playing, the playing sound is faded out and not replaced. If the node has the same sound file as the sound already being played, then that sound continues to be played (it’s *not* restarted, even if the sound file has completed).
* The node’s title is presented.
* For each description element (text or image), subject to a guard, display that element.
* If any options have a timer on them, warn the user and pause. Do not display any options until the user unpauses. Then, start the timers and continue.
* For each option, subject to a guard and a timer, display that option.
* If a timer is within 5 seconds of expiring, fade it out over those 5 seconds[[39]](#footnote-39).
* For each closing remark element, subject to a guard, display that element.
* When the user selects a displayed option, play any sound effect associated with that option and load the node to which the option links (then proceed again as above).
* The playing of sounds can (globally) be turned on and off. The status of whether to play a sound or not persists until changed. The default is that sounds are played.
* If the user elects to go back[[40]](#footnote-40), the previous node in the user’s history is loaded (if there is one).
* If the user elects to go forward, the next node in the user’s history is loaded (if there is one).
* The user’s current position in the narrative (and their narrative history) can be saved.
* The user can load a saved position in the narrative (and associated narrative history).
* The user can restart from the Start node. They will be given an option to save their narrative history if their current node has changed since their last save.
* The user can reconsult the Start node. This opens it but does not display any options apart from Return to Story (equivalent to the Back command). This functionality exists purely so that the user can check the story’s context or look at any maps or other information presented at the beginning. We probably don’t need this at launch.
* The user can quit. They will be offered a save if they have changed node since the last save, but can decline this if they wish to exit without saving.

Because Reading Mode is for testing and debugging purposes, it has some optional additional functionality which the user can call upon[[41]](#footnote-41):

* The story can run backwards. Above the title (indicated by a different colour, size, font or whatever) the user is shown the nodes that link to this one. The user can select one of these to go backwards in the story – towards the Start node.
* The user can display their narrative history (*ie.* what nodes lie on the path from Start node to this node).
* The user can clear their narrative history.
* The user can add/remove nodes by name to/from their narrative history, irrespective of whether they have visited those nodes or not.
* The user can save the current narrative history.
* The user can load a saved narrative history. This will not be possible if the user has since deleted a node in the saved narrative history.
* A narrative history can be one-stepped through.
* The shortest route from the Start node to the current node can be displayed. The user can set this to be their narrative history.
* The shortest route from the current node to any other node can be displayed, if one exists[[42]](#footnote-42).
* The shortest route from any other node to the current node can be displayed, if one exists.
* Guards can be displayed. If they are displayed, the user can toggle individual ones on and off, for example by clicking on them. This does not affect the narrative history, just the displaying of the particular guard.
* Reading Mode can be left by exit-to-edit. This means that if a problem is found with a node, Reading Mode can be left and the node it was displaying will be loaded into the Authoring view. The node can then be changed, and (if desired) Reading Mode restarted.
* If Reading Mode is restarted from the node at which it was last exited, the narrative history from that previous Reading Mode session is preserved. Otherwise, the narrative history is reinitialised.

Although ideally we would have all of the above at launch, the debugging features are actually of lower priority than the hypertext-reading features.

**Create Front Cover**

This is a simple view that allows the user to store an image file for the document to be used as a front cover (should the book have one – it doesn’t have to). It will not in general be used by authors unless they are self-publishing their material.

If the book doesn’t already have an image, this page opens on a file-selection modal window to select one. If it does have one, or if the file-selection window is closed without a suitable image file having been selected, it displays the image (or a blank place-holder). The image can be in either JPEG or PNG format[[43]](#footnote-43). The possibility exists in future versions of this software to allow images in other formats (including, for some project types, motion formats such as AVI); however, we don’t need this at launch. Also possible after launch but not necessary at launch is a cover-creation system, which would overlay a background image with text entered by the user.

The image will normally be displayed as it comes, but if possible[[44]](#footnote-44) we should offer preview options appropriate for each output platform. For example, previewing it for a Kindle would display it in 16 levels of greyscale (at 600x800 pixels full zoom).

**Create Front Matter**

Front matter is what appears in a book between the front cover and the body of the book. All of it is optional from the point of view of the author; some of it may be non-optional for the publisher.

The front matter view consists of a number of tabs, each of which contains a text box for the appropriate words (if any) to be entered (or, if they have already been created, edited). Most of these will only be appropriate for Book project types; however, because they could still appear in any project type, they aren’t accessible only from Book projects. The tabs are for:

* **Title Page.** This is filled in by the publisher, and contains the book’s title along with meta-data such as publisher name/address, copyright information, ISBN number, edition number, cataloguing information, disclaimers, warranties, and other details as may be required by law.
* **Foreword.** This is written by an individual other than the book’s author, saying how great and important the book and its author are.
* **Preface**. This is written by the author, and describes the circumstances obtaining to the creation of the work.
* **Acknowledgement.** Here, the author acknowledges those who helped in the work’s creation. Sometimes, it is folded into the Preface.
* **Introduction** . The purpose of this is for the author to explain the goals of the book. In practice, this is where the author would explain any idiosyncrasies in their hypertext that made it different from the “standard”.
* **Dedication.** The dedication is where the author thanks (usually) those people who had nothing to do with the creation of the book yet for whom the author wrote it; “To my loyal readers”, that kind of thing.

The book does not have a formal Prologue section; this is because the prologue (if there is one) appears in/as the first node – the Start. This node would also include useful reference material such as maps that the user may wish to go back to occasionally.

Note that unlike most books, this kind of work does not have a contents page for any project type.

**Create Back Matter**

Back matter is much the same as Front Matter, although less important in electronic media because people don’t easily turn to it. It appears after the body of the work[[45]](#footnote-45). Its tabs are:

* **Appendix.** This is where the author can describe things that may be of interest to the reader but which are not part of the text. For example, a story set during the Napoleonic wars may describe the way that impoverished gentlefolk were regarded in that period. It’s where the author gets to show off their research. Note that there can be any number of appendices, so there has to be an option for the user to add additional appendices as needs be[[46]](#footnote-46).
* **Glossary.** Here, any obscure terms are defined. We would probably want to make any use of those terms in the body of the text hyperlink to the glossary (or perhaps give a definition in mouseover).
* **Bibliography.** Should the author be so proud of their research that they wish to cite the primary sources they read while preparing for the work, this is where they would put it.

Traditionally, books can have other back matter too, but we don’t need it: the epilogue/afterword is folded into the text itself; the introduction is narrowly defined so there is no need for a matching outro; the work is not autobiographical so we don’t need a postscript; it is not polemical, so we don’t need a conclusion; the material that might go in a colophon nowadays goes on the title page instead.

**Create Back Cover**

This works much the same as the front cover: the user uploads an image file created separately.

In practice, a back cover is more formal than a front cover as it has to contain certain material in certain formats – most importantly, bar codes for any physical version of the book. It will also typically include author biography, blurb and favourable reviews from eminent personalities.

A physical book would also need a spine. For our purposes, the spine is part of the back cover (the strip to the right)[[47]](#footnote-47).

**Export Readable Story**

The function of this pop-up is to write out the story in a particular format. It works much like image exports do from graphics programs: a destination for the file(s) is proposed (which can be changed manually or through a Browse button); a destination format is selected from those available; if the target already exists then the user is queried as to whether to overwrite it or not; the story is written out in the appropriate form and the pop-up closes. This file is placed on the user’s local machine – it’s not placed on the server. Anyone who needs an exported copy gets one from exporting from the server to their client computer.

In all cases, the user has the option of not exporting the front cover, front matter, back matter and back cover (indicated *eg.* by unchecking check boxes).

Additionally, if the story contains features that are not supported by the output format (for example sound in a plain text format), the user is warned of this before exporting begins; if the user assents to the export, exporting proceeds but does not include the proscribed features.

In any event, prior to the writing of the story a sanity-check is run to ensure that all the nodes connect to the Start node and to an end node (one with no exits), and that all guard conditions reference nodes on a route from the node in which they occur back to the Start node. If there are unconnected nodes or nodes that capture the reader in inescapable loops, the user is warned of this and given the option of cancelling the story export (which the user can choose to ignore if, for example, they actually want to trap readers in inescapable loops). Also, all nodes have to be viable: if they are not, the export function informs the user and does not export the story.

The output formats supported are:

* **Plain text.** Each node is numbered, with the Start node as number 1. All remaining nodes are numbered in a random order and listed numerically from what results. Appended to each option for a node is a note saying *[Go to* N*]*, where N is the number of the consequent node. If the node is an end node (*ie.* it has no options), it says *[End]*. This format cannot support images, sounds, timers or guards.
* **Vanilla HTML.** Each node is an HTML page and the options are hyperlinks to the consequents nodes. This format can support images and sounds but not timers or guards. Readers can access the story using a simple, regular browser[[48]](#footnote-48).
* **DHTML.** This is a full-function version; it’s the same as Vanilla HTML but with JavaScript to handle the timers and guard conditions, and cookies to allow the reader to save where they got to (and how they got there) so they can reload partially-completed games. If we were to implement style sheets, this is where they could be applied.
* **Kindle.** This is also a full-function version, its output being in a data file format accepted by our proprietary Kindle story-reading software.
* **Kindle native.** This is the same as the Kindle format but it bundles together the data file and the means by which to read it (rather like the way that a self-extracting installation file bundles in the means by which its data is extracted). This enables the story to be loaded onto a Kindle as a stand-alone, readable product (*eHyperText*). It should be in a digital rights managed (DRM) format[[49]](#footnote-49).

Later versions of *eHypertool* may support additional formats, for example Word, iPhone, iPad. It is also possible that we might want to be able to export full Reading Mode versions in DHTML (*ie.* including debug functionality).

**Help**

///

///tooltips

///help window – non-modal, browser

///manual

///user forum

**Appendix**

**Usage Examples**

///

1. Stereotypes are always dangerous, but unless we want to wait until we have conducted a user competence analysis, it’s probably wise to assume low technological expertise than medium-to-high expertise. [↑](#footnote-ref-1)
2. In the sense that when it is open, the rest of the program is unavailable until it is closed, like with a “Save As” pop-up window in MS Word. [↑](#footnote-ref-2)
3. It may be that users want to keep the Read window open while they add text in the authoring window, using it as a kind of review panel. This would mean it couldn’t be modal, and a better solution would then be to have it as a pane in the tab for the authoring window with which it is associated. However, it really has to be modal in order for it to function like it would if a reader used it. [↑](#footnote-ref-3)
4. We may later record more details and have a user profile view. However, at this stage we probably don’t need that. [↑](#footnote-ref-4)
5. For example, a node called TakeTaxi would default to being renamed as TakeTaxi(2), although the user can choose another non-clashing (with the original or imported project) name instead. [↑](#footnote-ref-5)
6. This may be useful for self-contained “libraries” of story fragments, such as one for taking a train. [↑](#footnote-ref-6)
7. Or both at once, if it’s not too scary to program. [↑](#footnote-ref-7)
8. As usual when colours are involved, if we did this we would need to give the user the opportunity to change the defaults. [↑](#footnote-ref-8)
9. Alternatively, if there is room, the details of a node could be presented in a separate panel. The reason this is not proposed, however, is that users will then want to edit those details from this panel as well as from the Authoring view. [↑](#footnote-ref-9)
10. In theory we could allow users to switch these around here if we wanted, but as it’s a window intended to be used purely for visualising the directed graph, not editing it, we don’t really need that at launch. [↑](#footnote-ref-10)
11. Ultimately, this is because the Selection visualiser displays as a tree what is actually a cyclic directed graph. [↑](#footnote-ref-11)
12. Remember, this is just a mock-up – you don’t have to give it this exact look... [↑](#footnote-ref-12)
13. As many as can be displayed should be displayed – it’s not an “all or nothing” thing. Note that it’s also possible that a node has so many parents that they can’t all be displayed; if this is the case, the node should have as many as will fit plus “...”. If the user clicks on ..., the listed parents cycle to show the next unshown ones. [↑](#footnote-ref-13)
14. Multiple views of the same node are in theory possible, but they would all show the same thing so it’s probably wisest not to allow them. That way, users can’t lose track and re-edit a node they’re already editing. [↑](#footnote-ref-14)
15. For example, you may want to have multiple identical nodes (in terms of text) for the same location which have different effects later as they lead to different branches. [↑](#footnote-ref-15)
16. Well, put it this way: if one is, they all have to be. We could make null titles an option during Project Creation. [↑](#footnote-ref-16)
17. If the user were to add new options or change the consequent node of an option link, that would not automatically pick up the propagated sound; this is because it may be that the user wants to propagate the sound to some consequent nodes, but not all. [↑](#footnote-ref-17)
18. We could also allow them to be added between existing elements, allow options to be exchanged or allow existing elements to be moved around by dragging them. This is adding more complexity, however, for functionality that would be rarely used and that can be easily replicated by cut-and-paste anyway. [↑](#footnote-ref-18)
19. It is expected that the ones that appear will do so in order to replace a node that has simultaneously disappeared. [↑](#footnote-ref-19)
20. Again, we could allow options to be moved around but probably don’t need to at launch. If we do allow (for example) a “swap option positions” from the menu bar, or dragging and dropping, we should also offer similar functionality for description elements. [↑](#footnote-ref-20)
21. Unless the node is already open on a tab, in which case the tabbed view is opened instead. If the selected node is the very one in which its name was selected, nothing happens. [↑](#footnote-ref-21)
22. We may also be asked to provide one for when the node was created, but in practice this isn’t all that important – it’s typically only wanted for curiosity reasons. [↑](#footnote-ref-22)
23. For a new project, the Start node will be an end node, an empty node and an orphan simultaneously. [↑](#footnote-ref-23)
24. If possible, this pop-up should also allow the sound to be played (so the user can make sure it’s the right one). [↑](#footnote-ref-24)
25. Right-clicking on the plus should let you do this too. Another way to do it would be to have two buttons, one for text and one for image, so the user isn’t asked every single time which one they want to add. You won’t be asked (or the image button won’t appear) if the project option for images is set to off. [↑](#footnote-ref-25)
26. This may be too much for some users, who will want a delete button. Maybe we should give them one? [↑](#footnote-ref-26)
27. Say, 10 lines. [↑](#footnote-ref-27)
28. We may allow it to be a range of seconds from between which the reading software chooses at random at run-time. We don’t need this at launch, though. [↑](#footnote-ref-28)
29. We can use scroll bars if it gets too long, but really, if it gets too long there’s something the user is doing wrong... [↑](#footnote-ref-29)
30. The first time a node is visited, it does not appear in its own narrative history. If, by some circuitous route, it is visited again, it will appear in its history. In this way, a user could add a guard that would, say, have a barman greet you with a friendly wave on visits subsequent to the first. [↑](#footnote-ref-30)
31. The reader of the story would hear it automatically, of course, unless they muted their device or switched the sound off from the reading software. [↑](#footnote-ref-31)
32. Note the change of “option” to “element” in the pop-up, depending in whether the guard is guarding an option or an element. [↑](#footnote-ref-32)
33. Although I say “menu bar”, this doesn’t mean you can’t also put some of them on a tool bar if it’s going to work better. Entering Reading mode, for example, would probably make a good tool bar icon. [↑](#footnote-ref-33)
34. We also need Copy node name – just use the operating system default for that (^C in Windows). [↑](#footnote-ref-34)
35. We could have an option to paste-and-copy, which stores the content of the node, replaces it with the last-copied contents, then updates the last-copied contents to be that which was just deleted. I don’t know if this would get enough use to justify it, though. [↑](#footnote-ref-35)
36. We could have a list completed nodes, too, but it wouldn’t see much use. The same applies to list viable nodes. [↑](#footnote-ref-36)
37. And, possibly, user notes in the meta-data, if we like. [↑](#footnote-ref-37)
38. Indeed, it’s more than sufficient – we could probably leave out the list-by-type functionality for now. [↑](#footnote-ref-38)
39. Playtesting may result in a change to that figure of 5 seconds. [↑](#footnote-ref-39)
40. For example, using a browser-like “back” button. We don’t want to use the browser’s own back button because the server has no access to it; the server needs to have access to it so it can monitor the narrative history, which is necessary for processing guards. [↑](#footnote-ref-40)
41. This is not on by default, but can be set to be on from the menu bar. The debug status is remembered between openings of the Reading Mode window. [↑](#footnote-ref-41)
42. It might not exist, because the target node could be higher up in the directed graph than the current node, or not be on any path to it. [↑](#footnote-ref-42)
43. Because these are non-proprietary. We may also have to accept TIFF, which is proprietary but popular on Apple machines; the Kindle can also read it. [↑](#footnote-ref-43)
44. In other words, if there’s some code lying around that can be made to do this easily, we’ll do it; otherwise, it’s probably going to be too much effort to program from scratch for launch. [↑](#footnote-ref-44)
45. In our case, its components will appear as a hyperlink from the front page when readers actually load the work, so they don’t have to scan through to the end to find it. [↑](#footnote-ref-45)
46. For example by using a “+” tab like web browsers do. [↑](#footnote-ref-46)
47. We may nevertheless have to introduce a separate spine view tab, as the spine would show up in the back cover of an electronic book if we did it this way. Although some readers might appreciate seeing the spine in an electronic version, others might not as it would make for a slightly reduced image compared with that of the front cover. [↑](#footnote-ref-47)
48. This won’t trigger warnings about “ActiveX controls” that browsers often throw when they’re run on local files. [↑](#footnote-ref-48)
49. This may require more information from the user, such as a DRM key – it depends what Amazon needs to know. [↑](#footnote-ref-49)