Manual for hyText 1.0:
A hypertext library for wxWindows

Julian Smart
Knowledge Based Decision Support Group
Artificial Intelligence Applications Institute
80 South Bridge
University of Edinburgh
EH1 1HN

April 1993

Contents

Copyright notice

Chapter 1

Introduction

1.1 What is hyText?

This manual describes in detail the operation of the hyText library. hyText is a general-purpose hypertext library capable of displaying text with arbitrary blocks highlighted using different fonts and colours; an example of a hyText application is wxHelp.

This document is incomplete at present, but will be expanded when time permits. Most information is contained in the class reference section. The best way to use this library is to play with wxHelp, browse through the class reference (also available on-line as hytext.xlp) and then examine the wxHelp source (`wxhelp.h` and `wxhelp.cc`).

1.2 Description

The hyText class library is intended for wxWindows programmers who need hypertext functionality, that is, the ability to display text with highlighted words and phrases, with the ability to associate functionality with these blocks (such as viewing further information). A high-level API (Application Programming Interface) is provided to make these kinds of application easy to write.

wxHelp is one such application. Other possible applications include transcript editors and text output facilities within larger programs.

The main class in this library is **wxHyperTextWindow**, which is a type of canvas handles repainting, stores text, and supplies most of the API for manipulating hypertext files.

Before files may be displayed, the programmer must define a mapping between the integer block types and the font and colour styles which characterise text blocks. See the class reference for details, and `wxhelp.cc` for examples.

1.3 File format

An hyText file (usual extension `.xlp` consists of plain ASCII text, with blocks marked with codes as in the following:

```
\hy-X{Y}{Text}
```

where X is the block type and Y is the block identifier (unique within a file). The block type indicates the style of the block (font, colour, section), where the mapping between type and style is defined in a table (an instance of **wxHyperTextMapping**.

Note that blocks may be nested, in which case any styles in an inner block which have been the assigned 'default' characteristic will inherit the style from the outer block.

At the end of a file there is an optional index section, for example:

```
\hyindex{
"wxWindows Help"
101 102 "wx.xlp"
114 115
117 118
120 121
123 124
}
```

The first line indicates the start of the index, the second line is a title for the help file, subsequent lines (until a closing curly bracket) indicate the link between two block identifiers, with an optional filename after each pair of (long) integers.

This index is only stored and read by hyText, and must be accessed by the application in order to allow the user to actually traverse links.

Chapter 2

Classes and data types

The member functions are given in alphabetical order except for the constructors and destructors which appear first.

2.1 wxHTMappingStructure: wxObject

This class is used for storing mapping information for a block type.

wxHTMappingStructure::wxHTMappingStructure

void wxHTMappingStructure(**int** *blockType*, **int** *textSize*, **int** *textFamily*, **int** *textStyle*, **int** *textWeight*, **char** **textColour*, **char** **name*, **int** *attribute = wxHYPER_NONE*, **int** *visibility = TRUE*)

Constructor.

wxHTMappingStructure::~wxHTMappingStructure

void ~wxHTMappingStructure(**void**)

Destructor.

wxHTMappingStructure::Copy

wxHTMappingStructure * Copy(**void**)

Copies the structure.

wxHTMappingStructure::GetFont

wxFont * GetFont(**void**)

Finds or creates a font matching the characteristics stored in this structure.

2.2 wxHyperTextMapping: wxList

An object of this class stores a list of block mapping structures. The programmer needs to call **wxHyperTextWindow::SetMapping** with an object of this class, to specify how blocks are interpreted; several instances of **wxHyperTextWindow** could make use of the same **wxHyper-TextMapping**.

wxHyperTextMapping::wxHyperTextMapping

void wxHyperTextMapping(void)

Constructor.

wxHyperTextMapping::~wxHyperTextMapping

void ~wxHyperTextMapping(void)

Destructor.

wxHyperTextMapping::AddMapping

void AddMapping(int *blockType*, **int** *textSize*, **int** *textFamily*, **int** *textStyle*,
int *textWeight*, **char *** *textColour*, **char ****name*, **int** *attribute = -1*,
int *visibility = TRUE*)

Adds a mapping for a block type. *blockType* must be unique, and any parameters which have the default value (-1 for integers, NULL for strings) will be instantiated according to the context of the block. That is, if a block is nested with another block, the outer block's characteristics are used to fill in the default values.

See `wxhelp.cc` for examples.

wxHyperTextMapping::ClearMapping

void ClearMapping(void)

Deletes all members of the mapping list.

wxHyperTextMapping::FindByName

**void FindByName(char ****name*)

Finds a mapping structure by name.

wxHyperTextMapping::GetMapping

Bool GetMapping(int *blockType*, **int *** *textSize*, **int ****textFamily*, **int ****textStyle*,
**int ****textWeight*, **char *****textColour*, **char *****name*, **int ****attribute*,
**int ****visibility*)

Gets mapping values for a given block, returning FALSE if not found.

2.3 wxHyperTextWindow: wxCanvas

Objects of this class represent a canvas on which hypertext files are drawn. Most of the functionality of the library is accessed through this class.

Note that the class defines behaviour for **OnEvent** and **OnPaint**.

wxHyperTextWindow::wxHyperTextWindow

void wxHyperTextWindow(wxFrame ***_parent_, **int _x_, **int** _y_, **int** _w_, **int** _h_, **int** _style_)

Constructor; for details see **wxCanvas** in the wxWindows class reference.

wxHyperTextWindow::~wxHyperTextWindow

void ~wxHyperTextWindow(void)

Destructor.

wxHyperTextWindow::AddBlock

Bool AddBlock(int _xStart_, **int** _yStart_, **int** _xEnd_, **int** _yEnd_, **int** _blockType_, **int** _blockId_)

Adds a block from the first row/column to the second row/column, with given type and unique identifier. The display will not change until the functions **Compile** and **DisplayFileAt** are called.

wxHyperTextWindow::ClearBlock

Bool ClearBlock(int _blockId_)

Clears the given block. The display will not change until the functions **Compile** and **DisplayFileAt** are called.

wxHyperTextWindow::ClearFile

void ClearFile(void)

Clears the current hypertext file.

wxHyperTextWindow::Compile

void Compile(void)

Compiles the current hypertext file, that is, traverses the block structure of the file associating actual fonts and other attributes to text chunks. This must be done before a file may be displayed, and may also require the functions **SaveSection** and **RestoreSection** to be called in order to save and restore the current position in the file, since compilation destroys section pointers.

After a **Compile** (which is necessary after marking up or any operation which affects the display) the file must be displayed with **DisplayFileAt** or **RestoreSection**.

wxHyperTextWindow::DiscardEdits

void DiscardEdits(**void**)

Discards any edits (just sets the internal *modified* flag to FALSE).

wxHyperTextWindow::DisplayFile

void DisplayFile(**void**)

Draw the text at the point found by **DisplayFileAt**.

wxHyperTextWindow::DisplayFileAt

void DisplayFileAt(**long** *blockId*, **Bool** *refresh = TRUE*)

Positions the file at the given block, drawing the text only if *refresh* is TRUE. If *blockId* is -1, the file is displayed at the top.

wxHyperTextWindow::DisplayFileAtTop

void DisplayFileAtTop(**void**)

Displays the file at the top (first section).

wxHyperTextWindow::DisplayNextSection

void DisplayNextSection(**void**)

Finds and displays the next section.

wxHyperTextWindow::DisplayPreviousSection

void DisplayPreviousSection(**void**)

Finds and displays the previous section.

wxHyperTextWindow::DrawOutline

void DrawOutline(**float** *x1*, **float** *y1*, **float** *x2*, **float** *y2*)

Draws a rectangular outline for rubber-banding using the given top-left and bottom-right coordinates

wxHyperTextWindow::FindBlock

wxTextChunk * FindBlock(**long** *blockId*)

For a given block id, returns the text chunk at the start of the block.

wxHyperTextWindow::FindBlockForSection

long FindBlockForSection(**wxNode** **sectionNode*)

Pointers to blocks which mark sections are stored in the data member **sections**. This function takes a node which is known to point to a text chunk marking a block, and returns the block id. This is a fairly trivial function since it just gets the **wxTextChunk** from the node and returns its **block_id**.

wxHyperTextWindow::FindChunkAtBlock

wxNode * FindChunkAtBlock(**long** *blockId*)

For a given block id, returns the position in the text chunks list of the first CHUNK_START_LINE chunk before the block. A **wxNode** pointer is returned to allow the programmer to efficiently traverse the text chunks list from this point. The data stored in this node is a **wxTextChunk** object.

This function may not be very useful for programmers; it is mainly for internal use. Normally functions returning and taking block ids are used for manipulating blocks.

wxHyperTextWindow::FindChunkAtLine

wxNode * FindChunkAtLine(**long** *blockId*)

For a given block id, returns the position in the text chunks list of the first chunk on the given line. A **wxNode** pointer is returned to allow the programmer to efficiently traverse the text chunks list from this point. The data stored in this node is a **wxTextChunk** object.

This function may not be very useful for programmers; it is mainly for internal use. Normally functions returning and taking block ids are used for manipulating blocks.

wxHyperTextWindow::FindPosition

Bool FindPosition(**float** *mouseX*, **float** *mouseY*, **int** **charPos*, **int** **linePos*, **long** **blockId*)

Finds the character and line position of the given point, plus the id of the block found. Returns FALSE if no character was found at this position.

wxHyperTextWindow::GenerateId

long GenerateId(**void**)

Generates a unique identifier for a block; may be overridden to supply a different generator.

wxHyperTextWindow::GetBlockText

void GetBlockText(**char** **buffer*, **int** *maxSize*, **long** *blockId*)

void GetBlockText(**char** **buffer*, **int** *maxSize*, **wxNode** **node*, **long** *blockId*)

Gets the plain text bounded by the given block, stripping out any block codes. The second form is more efficient since it takes a node containing a pointer to the **wxTextChunk**, and doesn't have to search for the block.

wxHyperTextWindow::GetBlockType

int GetBlockType(**long** *blockId*)

Gets the type of the given block.

wxHyperTextWindow::GetCurrentSectionNumber

int GetCurrentSectionNumber(**void**)

Gets the number of the currently-displayed section, starting from 1. Zero is returned if there are no section markers.

wxHyperTextWindow::GetEditMode

Bool GetEditMode(**void**)

Returns TRUE if the hypertext window is editable.

wxHyperTextWindow::GetFirstSelection

long GetFirstSelection(**void**)

Gets the first block selected. Use **GetNextSelection** for subsequent blocks. Returns -1 if no more selections.

wxHyperTextWindow::GetLinkTable

wxHashTable * GetLinkTable(**void**)

Returns the hypertext window's hash table used for storing links between blocks. Objects of type **HypertextItem** are stored in the table, containing a destination filename and destination block id; these objects must be indexed by the source block id, to store a link between a source block and destination block.

This is only relevant if using the built-in index facility, rather than implementing your own index. You need to put and get explicitly, and writing to a file will use this table for saving the index. For example:

```
if (GetLinkTable()->Get(block_id))
  MainFrame->SetStatusText("This block already linked!");
else if (hySelection->block_id > -1)
{
  GetLinkTable()->Put(block_id,
    new HypertextItem(hySelection->filename, hySelection->block_id));
  modified = TRUE;
  SelectBlock(hySelection->block_id, FALSE);
  Compile();
  DisplayFile();
}
```

wxHyperTextWindow::GetNextSelection

long GetFirstSelection(void)

Gets the next block selected (use **GetFirstSelection** to start. Returns -1 if no more selections.

wxHyperTextWindow::GetTitle

char * GetTitle(void)

Returns NULL or the title (pointer to the hypertext window's local memory).

wxHyperTextWindow::LoadFile

Bool LoadFile(char ****file*)**

Loads the named file.

wxHyperTextWindow::Modified

Bool Modified(void)

Returns true if the user has modified the text.

wxHyperTextWindow::OnBeginDragLeft

void OnBeginDragLeft(float *x*, **float** *y*, **long** *blockId*, **int** *keys***)**

Called when the user starts to left-drag. Overrideable.

wxHyperTextWindow::OnBeginDragRight

void OnBeginDragRight(float *x*, **float** *y*, **long** *blockId*, **int** *keys***)**

Called when the user starts to right-drag. Overrideable.

wxHyperTextWindow::OnDragLeft

void OnDragLeft(Bool *draw*, **float** *x*, **float** *y*, **long** *blockId*, **int** *keys*)

Called when the user is in the middle of a drag operation; called once with *draw* equal to FALSE and with x and y equal to the old values, then again with *draw* equal to TRUE and updated x and y (to allow erase/draw operations).

wxHyperTextWindow::OnDragRight

void OnDragRight(Bool *draw*, **float** *x*, **float** *y*, **long** *blockId*, **int** *keys*)

Called when the user is in the middle of a drag operation; called once with *draw* equal to FALSE and with x and y equal to the old values, then again with *draw* equal to TRUE and updated x and y (to allow erase/draw operations).

wxHyperTextWindow::OnEndDragLeft

void OnEndDragLeft(float *x*, **float** *y*, **long** *blockId*, **int** *keys*)

Called when the user finishes left-dragging. Overrideable.

wxHyperTextWindow::OnEndDragRight

void OnEndDragRight(float *x*, **float** *y*, **long** *blockId*, **int** *keys*)

Called when the user finishes right-dragging. Overrideable.

wxHyperTextWindow::OnLeftClick

void OnLeftClick(float *x*, **float** *y*, **int** *charPos*, **int** *linePos*, **long** *blockId*, **int** *keys*)

Called when the user left-clicks. Overrideable. The default behaviour when SHIFT is held down is to select or deselect the mouse-over block.

wxHyperTextWindow::OnRightClick

void OnRightClick(float *x*, **float** *y*, **int** *charPos*, **int** *linePos*, **long** *blockId*, **int** *keys*)

Called when the user right-clicks. Overrideable.

wxHyperTextWindow::OnSelectBlock

void OnSelectBlock(long *blockId*, **Bool** *select*)

Called whenever a block is selected or deselected. Overridable.

wxHyperTextWindow::RestoreSection

void RestoreSection(void)

When a call is made to **Compile**, the current pointer to the current section becomes invalid, since all sections are recalculated. You need to **SaveSection** before **Compile**, followed by **RestoreSection** after the **Compile**, in order to restore the display to the previous state.

wxHyperTextWindow::SaveFile

Bool SaveFile(char **file*)

Saves the hypertext file and index.

wxHyperTextWindow::SaveSection

void SaveSection(void)

When a call is made to **Compile**, the current pointer to the current section becomes invalid, since all sections are recalculated. You need to call this before **Compile**, followed by **RestoreSection** after the **Compile**, in order to restore the display to the previous state.

wxHyperTextWindow::SelectBlock

void SelectBlock(wxTextChunk * *block*, **Bool** *select = TRUE*)

void SelectBlock(long *blockId*, **Bool** *select = TRUE*)

If *select* is TRUE, select the existing block, marking it in cyan (colour screens) or in inverse video (monochrome screens). If *select* is FALSE, deselect the block. The first form is more efficient since no search need be done for the block.

Note that **Compile** must be called before this call has any visible effect.

wxHyperTextWindow::SetBlockType

void SetBlockType(long *blockId*, **int** *blockType*)

Set the specified block to have the given type.

wxHyperTextWindow::SetEditMode

void SetEditMode(Bool *editable*)

Specifies whether the user should be able to mark up the text or not.

wxHyperTextWindow::SetIndexWriting

void SetIndexWriting(**Bool** *indexWriting*)

Specifies whether the built-in index and title should be written when **SaveFile** is called. The default is FALSE.

wxHyperTextWindow::SetMapping

void SetMapping(**wxHyperTextMapping** **mapping*)

Specify the set of block mappings for this window; this must be called.

wxHyperTextWindow::SetMargins

void SetMargins(**int** *left*, **int** *top*)

Sets the margins to leave to the left and top of the canvas when displaying text.

wxHyperTextWindow::SetTitle

void SetTitle(**char** **title*)

Sets the title of the hypertext window (allocates its own memory), to be written to the index file if index writing mode is on.

2.4 wxTextChunk: wxObject

This class is used for storing a text string which has all the same font and colour attributes. The entire hypertext file is broken up into a list of these fragments, and the **Compile** function assigns actual font and colour attributes to each chunk. A text chunk may also mark the start of a line (each line has a special start line text chunk).

If a chunk represents the start of a block, the **block_id** is this block. For chunks within a block, the **block_id** is always the id of the block currently in scope. A text chunk which marks the end of a block has **block_id** set to the *next* block's id, but **end_block** set to the ending block's id. This is because a text chunk contains the *next* fragment of text, and an end block chunk has two purposes: to end one block, and continue another.

wxTextChunk::wxTextChunk

void wxTextChunk(**int** *chunkType*, **int** *lineNumber*, **char** **text*, **wxFont** **font*, **wxColour** **colour*, **int** *blockType*, **long** *blockId*, **int** *attribute*, **Bool** *visibility*)

Constructor. Used only internally.

wxTextChunk::~wxTextChunk

void ~wxTextChunk(**void**)

Destructor. Used only internally.

wxTextChunk::background_colour

wxColour * *background_colour*

The background colour allocated for the chunk by **Compile**.

wxTextChunk::block_id

long *block_id*

Id of the block associated with the text in the chunk.

wxTextChunk::block_type

int *block_type*

Block type, an integer defined by the application using a **wxHyperTextMapping** object.

wxTextChunk::chunk_type

int *chunk_type*

The **chunk_type** data member may be one of:

- CHUNK_START_BLOCK
- CHUNK_START_UNRECOGNIZED_BLOCK
- CHUNK_END_BLOCK
- CHUNK_START_BLOCK

wxTextChunk::colour

wxColour * *colour*

The foreground colour allocated for the chunk by **Compile**.

wxTextChunk::end_id

long *end_id*

Id of the block which has just ended, if the type of this chunk is CHUNK_END_BLOCK. **block_id** is the id of block which has come into scope, and which starts with the text stored in the chunk.

wxTextChunk::font

wxFont * *font*

The font allocated for the chunk by **Compile**.

wxTextChunk::line_no

int *line_no*

The line number for this chunk.

wxTextChunk::logical_op

int *logical_op*

The logical operator for this chunk.

wxTextChunk::selected

Bool *selected*

For chunks which start a block, TRUE if the block is currently selected.

wxTextChunk::special_attribute

int *special_attribute*

For a block-starting chunk, specifies one or more special attributes ORed together. There is currently only one such attribute, wxHYPER_SECTION, which if present indicates that the block starts a new section.

wxTextChunk::text

char * *text*

The actual text in the chunk.

wxTextChunk::visibility

Bool *visibility*

For a block-starting chunk, determines whether the chunk is visible.

Index