Bookreader Design Document

This document explains Bookreader terminology and lists the routines you must use to interface with the Bookreader.

Revision/Update Information: First draft.
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAPTER 1</strong> BOOK WRITER INTERFACE OVERVIEW</td>
</tr>
<tr>
<td>1.1 ORGANIZATION OF DATA</td>
</tr>
<tr>
<td>1.1.1 Topics</td>
</tr>
<tr>
<td>1.1.2 Types of Data Chunks</td>
</tr>
<tr>
<td>1.1.2.1 Ftext packets</td>
</tr>
<tr>
<td>1.1.2.2 Subchunks</td>
</tr>
<tr>
<td>1.2 TYPICAL ROUTINES USED TO BUILD A BOOK</td>
</tr>
<tr>
<td><strong>CHAPTER 2</strong> BWI ROUTINES</td>
</tr>
<tr>
<td>BWI_BOOK_ABORT</td>
</tr>
<tr>
<td>BWI_BOOK_CLOSE</td>
</tr>
<tr>
<td>BWI_BOOK_COPYRIGHT</td>
</tr>
<tr>
<td>BWI_BOOK_CREATE</td>
</tr>
<tr>
<td>BWI_BOOK_FIRST_TOPIC</td>
</tr>
<tr>
<td>BWI_BOOK_FONT</td>
</tr>
<tr>
<td>BWI_BOOK_LICENSE</td>
</tr>
<tr>
<td>BWI_BOOK_LICENSE_ALT</td>
</tr>
<tr>
<td>BWI_BOOK_SYMBOL</td>
</tr>
<tr>
<td>BWI_BOOK_SYMBOL_ALT</td>
</tr>
<tr>
<td>BWI_DIRECTORY_CLOSE</td>
</tr>
<tr>
<td>BWI_DIRECTORY_CREATE</td>
</tr>
<tr>
<td>BWI_DIRECTORY_ENTRY</td>
</tr>
<tr>
<td>BWI_SYMBOL_DEFINE</td>
</tr>
<tr>
<td>BWI_TOPIC_CLOSE</td>
</tr>
<tr>
<td>BWI_TOPIC_CREATE</td>
</tr>
<tr>
<td>BWI_TOPIC_DATA_CHUNK</td>
</tr>
<tr>
<td>BWI_TOPIC_DATA_SUBCHUNK</td>
</tr>
<tr>
<td>BWI_TOPIC_REFERENCE_RECT</td>
</tr>
<tr>
<td>BWI_TOPIC_EXTENSION_RECT</td>
</tr>
<tr>
<td>BWI_TOPIC_REFERENCE_POLY</td>
</tr>
<tr>
<td>BWI_TOPIC_EXTENSION_POLY</td>
</tr>
<tr>
<td>BWI_TOPIC_NEXT</td>
</tr>
<tr>
<td>BWI_TOPIC_PREVIOUS</td>
</tr>
<tr>
<td>EXAMPLES</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>1–1 Structure of Ftext Packets</td>
</tr>
<tr>
<td>1–2 Ftext Packet Containing Text</td>
</tr>
<tr>
<td>1–3 Ftext Packet Containing Rule</td>
</tr>
<tr>
<td>1–4 Text Packet Example</td>
</tr>
<tr>
<td>1–5 Structure for FSE Image Information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLES</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1 Text Packet Fields in Example</td>
<td>1–3</td>
</tr>
</tbody>
</table>
1 Book Writer Interface Overview

An authoring tool links to the Bookreader through the Book Writer Interface (BWI). The authoring tool must provide information to the BWI about the book to be displayed by calling routines defined in the BWI.

The information describing the book to be displayed must be defined in terms of the structural elements described below.

1.1 Organization of Data

A book is the central object of the BWI. A book consists of a series of topics. Each topic in turn consists of a series of data chunks. These objects are described in detail in the following sections.

1.1.1 Topics

A topic is a collection of related units of information. The authoring tool determines the specific structure of a topic, and must present information describing each topic to the BWI in the form of data chunks.

1.1.2 Types of Data Chunks

A data chunk is a discrete unit of information. Data chunks that contain symbolic names can be referenced. A data chunk consists of one or more of any of the following:

- Ftext packets, for storage of text and rules.
- Subchunks, for storage of graphical objects.
- Extensions, for defining an area in which the background is shaded.
- References, for defining an area around which to draw a rectangle or polygon. The bounded area contains a pointer to other data chunks. References are also known as hotspots.

Refer to the file metaformat.h in Appendix A to see the valid data type values.

1.1.2.1 Ftext packets

The authoring tool passes text data to the BWI by a pointer to its address in the bwi_topic_data_chunk routine. This data must be in the form of ftext packets, which can contain either text or a rule. A new ftext packet is created whenever there is:

- a font change
- vertical movement
- horizontal movement
- a hot spot
An ftext packet contains information in the form "tag-length-value". The tag indicates that the information to follow is an ftext packet. The length is the number of bytes in the packet. The maximum packet length is 256 bytes. Value is either text at some resolution, or a rule. The format is shown in Example 1–1.

**Example 1–1 Structure of Ftext Packets**

```c
typedef struct _FTEXT{
    unsigned char tag
    unsigned char len
    char value[1]
} FTEXT;
```

The value member can contain data describing text or a rule. The values are defined in metaformat.h, in Appendix A. The contents of the text and rule structures are different (Example 1–2 and Example 1–3). If the value is of type text, the fields describe the x and y coordinates at which to begin the character, the font to be used, and the contents of the data. If the value is of type rule, the fields describe the x and y coordinates at which to begin the rule, and the width and height of the rule:

**Example 1–2 Ftext Packet Containing Text**

```c
typedef struct _TEXT{
    short int x;
    short int y;
    unsigned short int font_num;
    char data[1];
} TEXT;
```

**Example 1–3 Ftext Packet Containing Rule**

```c
typedef struct _RULE{
    short int x;
    short int y;
    short int width;
    short int height;
} RULE;
```

Within text, the font for the packet is passed in by the font_id argument to the *bwi_book_font* routine.

The data consists of one or more words, each of which begins with a one-byte delta x, which is the offset from the beginning of the word, the number of characters in the word, and the actual character data.
Example 1–4 shows an ftext packet containing text, in this case two words. The first four fields of the packet contain the packet header information. Table 1–1 lists the contents and provides a description of each field in the example.

Example 1–4  Text Packet Example

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
+------------------------------------------------------+
| 3, 50, 200, 1, 0, 3, t, h, e, 150, 5, b, o, o, k s |
+------------------------------------------------------+
```

Table 1–1  Text Packet Fields in Example

<table>
<thead>
<tr>
<th>Field</th>
<th>Contents</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>ftext400—resolution is 400dpi (required by Bookreader)</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>x coordinate</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>y coordinate</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>font</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>delta x, offset of first character</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>number of characters, first word</td>
</tr>
<tr>
<td>7</td>
<td>t</td>
<td>character</td>
</tr>
<tr>
<td>8</td>
<td>h</td>
<td>character</td>
</tr>
<tr>
<td>9</td>
<td>e</td>
<td>character</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>delta x, offset of next character</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>number of characters, next word</td>
</tr>
<tr>
<td>12</td>
<td>b</td>
<td>character</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>character</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>character</td>
</tr>
<tr>
<td>15</td>
<td>k</td>
<td>character</td>
</tr>
<tr>
<td>16</td>
<td>s</td>
<td>character</td>
</tr>
</tbody>
</table>

The authoring tool must build packets according to this format, and pass the data to the BWI by `bwi_topic_data_chunk`.

1.1.2.2  Subchunks

Subchunks are nonaddressable units within data chunks, used for storage of graphical objects. The information describing subchunks is passed to the BWI by the `bwi_topic_data_subchunk` routine. The data must be of type FSE, RAGS, or display PostScript (DPS).
Example 1–5  Structure for FSE Image Information

typedef struct _IMAGE {
    short int res_x;    /* horizontal resolution */
    short int res_y;    /* vertical resolution */
    short int pix_width; /* width of image in pixels */
    short int pix_height; /* height of image in pixels */
    char data [1];      /* start of image data */
} IMAGE;

For FSE (bitmapped images), the image structure must be filled in according to the format in metaformat.h (Example 1–5), and the bitmapped data must then be passed in.

For a RAGS or DPS file, the data can be passed in as is, without filling in the image structure in metaformat.h. The dimensions of the file are read from the file itself.

1.2 Typical Routines Used to Build a Book

The following is a typical sequence of the routines used to build a book. These routines are described in detail in Chapter 2.

1  Provide a book name and get a unique identifier, using bwi_book_create. This unique book_id is used in all subsequent BWI calls, and is part of the identifier for all data chunks in the book.

2  If the book contains a copyright, provide copyright information consisting of the book_id, chunk_id, and topic_id using bwi_book_copyright.

   If the book does not contain a copyright, use bwi_book_first_topic, listed below, to get the first topic. You must use one of these routines.

3  Provide optional LMF licensing information, using bwi_book_license and bwi_book_license_alt.

4  Provide an optional book title and book symbol to override the defaults, which use the root filename. Use bwi_book_title and bwi_book_symbol, respectively.

5  Define fonts, using bwi_book_font.

6  Create topics.

   a.  Provide the book_id, a topic_type (standard for an inline topic or reference for a popup topic), and get a topic_id for each new topic, using bwi_topic_create.

   b.  Provide a chunk_id for each data chunk within the topic, using bwi_topic_data_chunk. Write the data to the page according to the format in metaformat.h.

   c.  Define reference rectangles and polygons for references and extensions, using:

          bwi_topic_reference_rect
          bwi_topic_reference_poly
          bwi_topic_extension_rect
          bwi_topic_extension_poly
7 Close topics, using `bwi_topic_close`.

8 Create directories.
   a. Create a directory entry page, by providing a directory name and id, and a flag indicating the type of directory, using `bwi_directory_create`.
   b. Add entries to the directory entry page, using `bwi_directory_entry`.

9 Close directories, using `bwi_directory_close`.


This chapter lists the arguments to and the error messages returned by the BWI routines. The routines are available in an object library.
bwi_book_abort

Prematurely terminates a bookbuild.

**FORMAT**

\[
\text{BWI\_ERROR bwi\_book\_abort (unsigned int *book\_id);}\]

**ARGUMENTS**

*book\_id*

The longword identifier returned from bwi_book_create.

**DESCRIPTION**

Aborts the bookbuild upon detection of a fatal error, by closing and deleting the current book file.

**RETURNS**

- BwiOk: Successful completion
- BwiInvalidBookId: Book identifier is invalid.
- Any valid RMS file close error
bwi_book_close

Closes a book.

**FORMAT**

```c
BWI_ERROR bwi_book_close (unsigned int *book_id,
UNDEFSYM **undef_symbol);
```

**ARGUMENTS**

- **book_id**: The longword identifier returned from `bwi_book_create`.
- **undef_symbol**: A pointer to a linked list of structures.

**DESCRIPTION**

`bwi_book_close` checks for unresolved references. If one is found, it returns the BwiUndefSymbol status code and `undef_symbol` points to a linked list of structures of type `UNDEFSYM`, which is defined in `metaformat.h`.

`bwi_book_close` writes out the topic index and bookheader, and closes the file. It automatically creates a single-entry bookshelf file.

**RETURNS**

- `BwilInvalidBookId`: Book identifier is invalid.
- `BwilInvalidTopicId`: Topic identifier is invalid.
- `BwilInvalidUpdate`: Error encountered when updating header.
- `BwilFailMalloc`: Generic memory allocation failure.
- `BwilFailFileRead`: Error when reading file.
- `BwilFailFileWrite`: Error when writing to output file.
- `BwilOk`: Successful completion
- `BwilUndefSymbol`: Unresolved reference found.
- Any valid RMS error code.
**bwi_book_copyright**

Defines copyright links for a book.

---

**FORMAT**

```
BWI_ERROR bwi_book_copyright (unsigned int *book_id,
    BWI_OBJECT_ID chunk_id,
    BWI_OBJECT_ID topic_id);
```

---

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **chunk_id**
  The chunk id of the copyright notice. This is the identifier returned from `bwi_topic_data_chunk`.

- **topic_id**
  Reserved for future use.

---

**DESCRIPTION**

`bwi_book_copyright` displays the data chunk containing the copyright immediately upon opening a book, if the book contains a copyright. If a copyright is not present, use `bwi_book_first_topic`.

You must use one or the other. If neither routine is used, no topic window is displayed.

---

**RETURNS**

- `BwiInvalidBookId` Book identifier is invalid.
- `BwiInvalidKeyId` Topic identifier is invalid.
- `BwiOk` Successful completion
bwi_book_create

Creates a new book.

FORMAT

BWI_ERROR bwi_book_create (char *file_name,
unsigned int *book_id);

ARGUMENTS

file_name
File name is the file specification to create. It is a pointer to a null-terminated character string.

book_id
A longword identifier used in all subsequent BWI calls.

DESCRIPTION

bwi_book_create is the first call made when building a book. This routine sets up internal data structures, and sets the default book title and default book symbol to be the root filename.

The book_id argument is passed by reference.

RETURNS

BwiOk
Successful completion
BwiBadArg
Expected parameter is missing or NULL.
BwiFailFileOpen
Error encountered when opening file.
BwiFailMalloc
Generic memory allocation failure.
BwiFailHeadCreate
Error when creating the file header.
BwiErrNoMemoryNum
Internal memory allocation failure.
BwiStrTooLong
String has exceeded defined limits.
Any valid RMS error code.
bwi_book_first_topic

Defines the first topic to open.

FORMAT

BWI_ERROR bwi_book_first_topic (unsigned int *book_id,
BWI_OBJECT_ID topic_id);

ARGUMENTS

book_id
The longword identifier returned from bwi_book_create.

topic_id
The identifier returned from bwi_topic_create.

DESCRIPTION

bwi_book_first_topic displays a topic window if no copyright is present
in the book. If a copyright is present, use bwi_book_copyright.

You must use one or the other. If neither routine is used, no topic window
is displayed.

RETURNS

BwiOk Successful completion
BwiInvalidBookId Book identifier is invalid.
BwiInvalidTopicId Topic identifier is invalid.
bwi_book_font

Defines a font id for use within the book.

**FORMAT**

BWI_ERROR bwi_book_font (unsigned int *book_id,
unsigned int font_id,
char *font_name);

**ARGUMENTS**

*book_id*  
The longword identifier returned from bwi_book_create.

*font_id*  
A unique id used to reference this font within the book.

*font_name*  
A pointer to a null-terminated character string containing the name of the font.

**DESCRIPTION**

This routine defines the fontname according to standard dechwindows nomenclature. For example:

-ADOBE-New Century Schoolbook-Medium-R-Normal-*(ps)-*-P-**-
ISO8859-1.

**RETURNS**

BwiOk  
Successful completion

BwiInvalidBookId  
Book identifier is invalid.

BwiInvalidTopicId  
Topic identifier is invalid.

BwiTopicWritten  
Once a topic is closed it cannot be updated.

BwiBadArg  
Expected parameter is missing or NULL.

BwiErrInvalidMemNum  
Memory deallocation failure.

BwiErrNoMemoryNum  
Internal memory allocation failure.
bwi_book_license

Defines the (optional) licensing information required to read a book.

**FORMAT**

```
BWI_ERROR bwi_book_license (insigned int *book_id,
char *producer,
char *product_name,
unsigned int prod_date[2],
unsigned long int prod_version);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **producer**
  A pointer to a null-terminated character string, maximum of 24 characters in length, containing the License Management Facility (LMF) producer name.

- **product_name**
  The LMF product name. A pointer to a null-terminated character string, maximum of 24 characters in length, containing the LMF product name.

- **prod_date**
  The LMF product release date.

- **prod_version**
  The LMF product version.

**DESCRIPTION**

Use this routine to pass licensing information, if a license is required to read the book.

**RETURNS**

- **BwiOk**
  Successful completion

- **BwiInvalidBookId**
  Book identifier is invalid.

- **BwiStrTooLong**
  String has exceeded defined limits.

- **BwiBadArg**
  Expected parameter is missing or NULL.
# bwi_book_license_alt

Alternate LMF product names.

## FORMAT

```
BWI_ERROR bwi_book_license_alt (char *alt_name);
```

## ARGUMENTS

**alt_name**

A pointer to a null-terminated character string containing alternate product names.

## DESCRIPTION

*bwi_book_license_alt* contains alternate LMF product names.

## RETURNS

- **BwiOk**  
  Successful completion
- **BwiInvalidBookId**  
  Book identifier is invalid.
- **BwiStrTooLong**  
  String has exceeded defined limits.
- **BwiFailHeadLength**  
  Header has overflowed.
- **BwiBadArg**  
  Expected parameter is missing or NULL.
bwi_book_symbol

Defines the symbolic name of the book.

**FORMAT**

```c
BWI_ERROR bwi_book_symbol
(unsigned int *book_id,
char *symbol_name);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **symbol_name**
  A pointer to a null-terminated character string, maximum of 32 characters, containing the symbolic name.

**DESCRIPTION**

This routine, if used, overrides the default symbol name (the root filename) written by `bwi_book_create`.

**RETURNS**

- **BwiOk** Successful completion
- **BwiInvalidBookId** Book identifier is invalid.
- **BwiBadArg** Expected parameter is missing or NULL.
- **BwiStrTooLong** String has exceeded defined limits.
bwi_book_title

Defines the title of the book.

**FORMAT**

BWI_ERROR bwi_book_title (unsigned int *book_id, char *book_name);

**ARGUMENTS**

*book_id*
The longword identifier returned from bwi_book_create.

*book_name*
A pointer to a null-terminated character string, maximum of 128 characters, containing the title of the book.

**DESCRIPTION**

This routine, if used, overrides the default title (the root filename) written by bwi_book_create.

**RETURNS**

BwiOk
Successful completion

BwiInvalidBookId
Book identifier is invalid.

BwiBadArg
Expected parameter is missing or NULL.

BwiStrTooLong
String has exceeded defined limits.
bwi_directory_close

Closes a directory.

**FORMAT**

```
BWI_ERROR bwi_directory_close (unsigned int *book_id,
    BWI_OBJECT_ID dir_id);
```  

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **dir_id**
  The identifier returned from `bwi_directory_create`.

**DESCRIPTION**

Once a directory is closed it cannot be reopened or reused.

**RETURNS**

- BwiOk: Successful completion
- BwiInvalidDirId: Directory identifier is invalid.
- BwiInvalidBookId: Book identifier is invalid.
- BwiInvalidTopicId: Topic identifier is invalid.
- BwiFailFileRead: Error encountered when reading file.
- BwiFailFileWrite: Error when writing to output file.
- BwiInvalidUpdate: Error encountered when updating header.
- Any valid RMS file close error: VMS only.
bwi_directory_create

Creates a directory.

FORMAT

BWI_ERROR bwi_directory_create (unsigned int *book_id,
char *dir_name,
unsigned int flags,
BWI_OBJECT_ID *dir_id);

ARGUMENTS

book_id
The longword identifier returned from bwi_book_create.

dir_name
The name of the directory. dir_name is a pointer to a null-terminated character string.

flags
Indicate attributes of the directory. All flags default to False.

• BWI_CONTENTS_MASK
  True if the directory is the Table of Contents for the book. This flag may only be set for one directory.

• BWI_INDEX_MASK
  True if the directory is the primary index for the book. This flag may only be set for one directory.

• BWI_DEFAULT_MASK
  True if this is the primary directory of the book. The intended use of this flag is to tell the user interface to open this directory when opening the book. This flag may only be set for one directory. Usually this will be true for the Table of Contents.

• BWI_MULTI_VALUED_MASK
  True if directory entries may point to multiple objects (for example, Index). False if entries may point to at most one object (for example, Table of Contents).

dir_id
An identifier of the created directory.

DESCRIPTION

The flag values are defined in metaformat.h (Appendix A).
bwi_directory_create

**RETURNS**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiStrTooLong</td>
<td>String has exceeded defined limits.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiFailHeadLength</td>
<td>The header has overflowed its fixed length.</td>
</tr>
</tbody>
</table>
bwi_directory_entry

Adds an entry to a directory.

FORMAT

BWI_ERROR bwi_directory_entry
( unsigned int *book_id,
  BWI_OBJECT_ID *dir_id,
  unsigned int ref_count,
  unsigned int *ref_list,
  unsigned int level,
  unsigned int width,
  unsigned int height,
  char *data_addr,
  unsigned int data_len,
  unsigned int data_type,
  char *title,
  BWI_OBJECT_ID *entry_id );

ARGUMENTS

book_id
The longword identifier returned from bwi_book_create.

dir_id
The identifier returned from bwi_directory_create.

ref_count
The number of references in the ref_list

ref_list
The address of a list of object identifiers of the referenced data chunk(s).

level
Directory entry level (1 = top). Level jumps must be in increments of one.
bwi_directory_entry

**width**
The width of the entry.

**height**
The height of the entry.

**data**
The address of the beginning of the data.

**data_len**
The number of bytes of data.

**data_type**
The format of the data.

**title**
A pointer to a null-terminated character string, maximum of 255 characters, containing the title of the directory entry.

**entry_id**

---

**DESCRIPTION**
The ref_count and ref_list arguments are the number of data chunks referenced and a list of chunk_ids themselves. Width, height, data, data_len, and data_type describe the displayable entry itself.

---

**RETURNS**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiInvalidDirId</td>
<td>Directory identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiTopicWritten</td>
<td>Once a topic is closed it cannot be updated.</td>
</tr>
<tr>
<td>BwiStrTooLong</td>
<td>String has exceeded defined limits.</td>
</tr>
<tr>
<td>BwiDirLevelSkip</td>
<td>Directory level increment was greater than one.</td>
</tr>
</tbody>
</table>
bwi_symbol_define

Assigns a symbolic name to a object_id that has been returned from bwi_topic_data_chunk.

**FORMAT**

BWI_ERROR bwi_symbol_define (unsigned int *book_id, BWI_OBJECT_ID object_id, char *symbolic_name, BMD_BOOLEAN flag);

**ARGUMENTS**

*book_id*
The longword identifier returned from bwi_book_create.

*object_id*
The identifier returned from bwi_topic_data_chunk.

*symbolic_name*
Symbolic name being assigned to the chunk_id.

*flag*
A TRUE value means that the symbol is a non-generated static symbol, a false value means that the symbol is generated. Generated symbols will not be stored in the symbol table.

**DESCRIPTION**

This routine stores the symbol name and object_id parameters in the book's symbol table.

**RETURNS**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiOk</td>
<td>Successful completion</td>
</tr>
<tr>
<td>BwiInvalidBookId</td>
<td>Book identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidChunkId</td>
<td>Chunk identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiTopicWritten</td>
<td>Topic has already been written to disk.</td>
</tr>
<tr>
<td>BwiErrNoMemoryNum</td>
<td>Internal memory allocation error.</td>
</tr>
</tbody>
</table>
bwi_topic_close

Closes a topic.

**FORMAT**

```
BWI_ERROR bwi_topic_close (unsigned int *book_id,
BWI_OBJECT_ID topic_id);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **topic_id**
  The identifier returned from `bwi_topic_create`.

**DESCRIPTION**

**RETURNS**

- BwiOk
  Successful completion
- BwiInvalidBookId
  Book identifier is invalid.
- BwiInvalidTopicId
  Topic identifier is invalid.
- BwiFailFileRead
  Error encountered when reading file.
- BwiFailFileWrite
  Error when writing to output file.
- BwiInvalidUpdate
  Error encountered when updating header.
- Any valid RMS file close error.
  VMS only.
bwi_topic_create

Creates a new topic.

FORMAT

BWI_ERROR bwi_topic_create
(unsigned int *book_id,
unsigned int topic_type,
BWI_OBJECT_ID *topic_id);

ARGUMENTS

book_id
The longword identifier returned from bwi_book_create.

topic_type
Indicates the type of topic, either standard or reference.

topic_id
An identifier of the created topic.

DESCRIPTION

Given the book_id and topic type, bwi_topic_create opens a topic and puts its identifier in topic_id.

RETURNS

BwiOk
Successful completion

BwiInvalidBookId
Book identifier is invalid.

BwiInvalidTopicId
Topic identifier is invalid.

BwiErrNoMemoryNum
Internal memory allocation error.
bwi_topic_data_chunk

Creates an addressable data chunk.

**FORMAT**

```
BWI_ERROR bwi_topic_data_chunk( unsigned int *book_id,
    BWI_OBJECT_ID topic_id,
    char *title,
    unsigned int width,
    unsigned int height,
    char *data_addr,
    unsigned int data_len,
    unsigned int data_type
    BWI_OBJECT_ID *chunk_id);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **topic_id**
  The identifier returned from `bwi_topic_create`.

- **title**
  A pointer to a null-terminated character string, containing the text displayed in the title bar.

- **width**
  The width of the data chunk.

- **height**
  The height of the data chunk.

- **data_addr**
  The address of the beginning of the data.

- **data_len**
  The number of bytes of data.
**data_type**  
The format of the data.

**chunk_id**  
The identifier of this data chunk.

### DESCRIPTION
A chunk is the main addressable unit within a topic. Width and height describe the dimensions of the chunk.

### RETURNS

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiOk</td>
<td>Successful completion</td>
</tr>
<tr>
<td>BwiInvalidBookId</td>
<td>Book identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiTopicWritten</td>
<td>Once a topic is closed it cannot be updated.</td>
</tr>
</tbody>
</table>
bwi_topic_data_subchunk

Creates a non-addressable data sub-chunk.

**FORMAT**

```c
BWI_ERROR bwi_topic_data_subchunk
(unsigned int *book_id,
BWI_OBJECT_ID parent_chunk_id,
unsigned int x_offset,
unsigned int y_offset,
unsigned int width,
unsigned int height,
char *data_addr,
unsigned int data_len,
unsigned int data_type);
```

**ARGUMENTS**

- **book_id**
The longword identifier returned from `bwi_book_create`.

- **parent_chunk_id**
The id of the parent chunk that this sub-chunk belongs to.

- **x_offset**
The horizontal offset from the left side of the parent chunk.

- **y_offset**
The vertical offset from the top of the parent chunk.
**width**
The width of the data chunk.

**height**
The height of the data chunk.

**data_addr**
The address of the beginning of the data

**data_len**
The number of bytes of data.

**data_type**
The format of the data, defined in metaformat.h.

**DESCRIPTION**
Defines a nonaddressable area within a data chunk that typically holds a graphic object.

**RETURNS**
- BwiInvalidBookId: Book identifier is invalid.
- BwiInvalidChunkId: Data chunk identifier is invalid.
- BwiInvalidTopicId: Topic identifier is invalid.
- BwiTopicWritten: Once a topic is closed it cannot be updated.
bwi_topic_reference_rect

Defines a cross reference that is rectangular in shape.

**FORMAT**

```c
BWI_ERROR bwi_topic_reference_rect (unsigned int *book_id, 
    BWI_OBJECT_ID chunk_id, 
    unsigned int x_offset, 
    unsigned int y_offset, 
    unsigned int width, 
    unsigned int height, 
    char *object_symbol, 
    char *book_symbol);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **chunk_id**
  The id of the chunk that this hot spot belongs to.

- **x_offset**
  Horizontal offset from the left side of the chunk.

- **y_offset**
  Vertical offset from the top of the chunk.

- **width**
  The width of the hot spot.

- **height**
  The height of the hot spot.

- **object_symbol**
  Pointer to a null-terminated character string, maximum of 31 characters in length, containing the symbolic name of the data chunk that is the
target of the cross reference. The symbolic name is defined in the `bwi_symbol_define` call for the target chunk.

*book_symbol*
Point to a null-terminated character string, maximum of 31 characters in length, containing the symbolic name of the book that is the target of the cross reference.

**DESCRIPTION**
*bwi_topic_reference_rect* defines a region within a chunk that has the option of being outlined or "hot". The x and y offset define the location, and the width and height define the size of the rectangle to draw. The targeted symbol name does not have to be defined at this point. An object symbol with no book symbol is a local cross reference, a book symbol with no object symbol is an external cross book reference that will point to another book. When both object and book symbols are present the reference will point to the specified chunk in another book.

**RETURNS**
- *BwiInvalidBookId*: Book identifier is invalid.
- *BwiInvalidTopicId*: Topic identifier is invalid.
- *BwiTopicWritten*: Once a topic is closed it cannot be updated.
- *BwiInvalidChunkId*: Data chunk identifier is invalid.
- *BwiOk*: Successful completion
- *BwiBadArg*: Expected parameter is missing or NULL.
bwi_topic_extension_rect

**bwi_topic_extension_rect**

Defines a body of extension text that is rectangular in shape.

**FORMAT**

```c
BWI_ERROR bwi_topic_extension_rect (unsigned int *book_id,
     BWI_OBJECT_ID chunk_id,
     unsigned int x_offset,
     unsigned int y_offset,
     unsigned int width,
     unsigned int height);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **chunk_id**
  The id of the chunk that this extension area belongs to.

- **x_offset**
  The horizontal offset from the left side of the chunk.

- **y_offset**
  The vertical offset from the top of the chunk.

- **width**
  The width of the extension.

- **height**
  The height of the extension.

**DESCRIPTION**

`bwi_topic_extension_rect` defines a shaded region within a chunk. The X and Y offsets define the location. The width and height define the size of the rectangle to shade.
<table>
<thead>
<tr>
<th>RETURNS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiInvalidBookId</td>
<td>Book identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidChunkId</td>
<td>Data chunk identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiTopicWritten</td>
<td>Topic identifier is invalid.</td>
</tr>
</tbody>
</table>
bwi_topic_reference_poly

Defines a local cross reference that is a polygon.

**FORMAT**

```c
BWI_ERROR bwi_topic_reference_poly
(unsigned int *book_id,
BWI_OBJECT_ID chunk_id,
unsigned int numpoints,
POINTS (*array_points)[],
char *object_symbol);

char *book_symbol);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **chunk_id**
  The id of the chunk that this hot spot belongs to.

- **numpoints**
  The number of points in the polygon.

- **array_points**
  The address of the array of points, numpoints long.

- **object_symbol**
  A pointer to a null-terminated character string, maximum of 31 characters, containing the symbolic name of the data chunk that is the target of the cross reference. The symbolic name is defined in the `bwi_topic_data_chunk` call for the target chunk.

- **book_symbol**
  Pointer to a null-terminating character string, maximum of 31 characters in length, containing the symbolic name of the book that is the target of the cross reference.
bwi_topic_reference_poly defines a region within a chunk that is hot. Numpoints is the number of coordinates needed to outline the region. array_points holds the coordinate pairs. The targeted symbol name does not have to be defined at this time. An object symbol with no book symbol is a local cross reference, a book symbol with no object symbol is an external cross book reference that will point to another book. When both object and book symbols are present the reference will point to the specified chunk in another book.

RETURNS

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BwiInvalidBookId</td>
<td>Book identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidTopicId</td>
<td>Topic identifier is invalid.</td>
</tr>
<tr>
<td>BwiInvalidChunkId</td>
<td>Data chunk identifier is invalid.</td>
</tr>
<tr>
<td>BwiBadArg</td>
<td>Expected parameter is missing or NULL.</td>
</tr>
<tr>
<td>BwiTopicWritten</td>
<td>Once a topic is closed it cannot be updated.</td>
</tr>
</tbody>
</table>
bwi_topic_extension_poly

Defines a body of extension text that is a polygon.

**FORMAT**

```
BWI_ERROR bwi_topic_extension_poly (unsigned int *book_id, BWI_OBJECT_ID chunk_id, unsigned int numpoints, POINTS (*array_points[]));
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **chunk_id**
  The id of the chunk that this extension text belongs to.

- **numpoints**
  The number of points in the polygon.

- **array_points**
  The address of the array of points.

**DESCRIPTION**

*bwi_topic_extension_poly* defines a shaded region within a chunk. `numpoints` is the number of coordinates needed to outline the region. `array_points` holds the coordinate pairs.

**RETURNS**

- BwiInvalidBookId  Book identifier is invalid.
- BwiInvalidChunkId  Data chunk identifier is invalid.
- BwiInvalidTopicId  Topic identifier is invalid.
- BwiTopicWritten   Once a topic is closed it cannot be updated.
- BwiOk             Successful completion
bwi_topic_next

FORMAT

BWI_ERROR bwi_topic_next (unsigned int *book_id,
BWI_OBJECT_ID topic_id,
BWI_OBJECT_ID prev_topic_id);

ARGUMENTS

book_id
The longword identifier returned from bwi_book_create.

topic_id
The identifier returned from bwi_topic_create.

prev_topic_id
The identifier returned from bwi_topic_create.

DESCRIPTION

Next and previous topic tracking is handled in the Book Writer Interface (BWI).

RETURNS

BwiOk Successful completion
BwiInvalidBookId Book identifier is invalid.
BwiInvalidTopicId Topic identifier is invalid.
bwi_topic_previous

Not implemented yet.

**FORMAT**

```c
BWI_ERROR bwi_topic_previous(unsigned int *book_id, BWI_OBJECT_ID topic_id, BWI_OBJECT_ID next_topic_id);
```

**ARGUMENTS**

- **book_id**
  The longword identifier returned from `bwi_book_create`.

- **topic_id**
  The identifier returned from `bwi_topic_create`.

- **next_topic_id**
  The identifier returned from `bwi_topic_create`.

**DESCRIPTION**

Next and previous topic tracking is handled in the Book Writer Interface (BWI).

**RETURNS**

- **BwiOk**
  Successful completion

- **BwiInvalidBookId**
  Book identifier is invalid.

- **BwiInvalidTopicId**
  Topic identifier is invalid.
A

Metaformat.h

****************************************************************************
/**
/** Copyright (c) 1989
/** by DIGITAL Equipment Corporation, Maynard, Mass.
/**
/** This software is furnished under a license and may be used and copied
/** only in accordance with the terms of such license and with the
/** inclusion of the above copyright notice. This software or any other
/** copies thereof may not be provided or otherwise made available to any
/** other person. No title to and ownership of the software is hereby
/** transferred.
/**
/** The information in this software is subject to change without notice
/** and should not be construed as a commitment by DIGITAL Equipment
/** Corporation.
/**
/** DIGITAL assumes no responsibility for the use or reliability of its
/** software on equipment which is not supplied by DIGITAL.
/**
****************************************************************************
/****************************************************************************
/****************************************************************************
/*** MODULE $metaformat ***/
/*
*/
/* FTEXT -- Tag Length Value format for generically coding information */
/*
*/
/* format: */
/* +--------+--------+-------- +*/
/* | Tag | Length | Value | +*/
/* +--------+--------+-------- +*/
/* */
/* (length includes the tag, length, and value fields) */
/* Constant FTEXT_LENGTH equals the total size of the type and length */
/* fields. */
/* */
/* Length must be less than 256 */
/* */
#define FTEXT_LENGTH 2 /* length of header */
typedef struct _FTEXT {
  unsigned char tag; /* "type" (opcode) identifier */
  unsigned char len; /* length of total packet */
  char value[1]; /*start of data (generic) */
} FTEXT;
/*
*/
/* Tag values for electrodoc internal data format (FTEXTs) */
/*
*/
#define FTEXT_MIN_TAG 1
#define FTEXT_RULE 1 /* draw a rule */
#define FTEXT_TEXT300 2 /* draw text stored in 300 dpi */
#define FTEXT_TEXT400 3 /* draw text stored in 400 dpi */
#define FTEXT_MAX_TAG 3
/*
*/
/* RULE -- packet value contents for RULE */
/*
*/
#define RULE_LENGTH 8 /* length of RULE packet */
typedef struct _RULE {
/*
*/
short int x; /* x coord. of upper left corner */
short int y; /* y coord. of upper left corner */
short int width; /* width of rule */
short int height; /* height of rule */
} RULE;
/* */
/* TEXT -- Text packet (text length is obtained by subtracting the */
/* FTEXT$K_LENGTH from the total packet length. */
/* */
/* The format of the data field is "text_words", where each */
/* text_word starts with a 1 byte "delta" value which */
/* is the amount of space to put before the word, followed */
/* by a 1 byte count, followed by that many letters, */
/* */
#define TEXT_LENGTH 6 /* length of TEXT packet header */
typedef struct _TEXT {
  short int x; /* x coord. of first char */
  short int y; /* y coord. of baseline */
  unsigned short int font_num; /* number assoc w/font by DEFINE_FONT */
  char data [1]; /* start of data */
} TEXT;
#define WORD_LENGTH 2 /* length of WORD packet */
typedef struct _WORD {
  unsigned char delta; /* horizontal offset to start of word */
  unsigned char count; /* number of char’s in word */
  char chars [1]; /* start of text */
} WORD;
/* */
/* IMAGE -- format for image information */
/* */
#define IMAGE_LENGTH 8 /* length of WORD packet */
typedef struct _IMAGE {
  short int res_x; /* horizontal resolution created at */
  short int res_y; /* vertical resolution created at */
  short int pix_width; /* width of image in pixels */
  short int pix_height; /* height of image in pixels */
  char data [1]; /* start of image data */
} IMAGE;
/* */
/* UNDEF -- format for undefined symbols */
/* */
typedef struct _UNDEFSYM {
  struct _UNDEFSYM *next;
  unsigned ckid;
  char name[32];
} UNDEFSYM;
/* */
/* Data type values for electrodoc internal data chunks */
/* */
#define MIN_CHUNK_TYPE 1
#define CHUNK_ASCII 1 /* ASCII text */
#define CHUNK_FTEXT 2 /* DOCUMENT Formatted Text */
#define CHUNK_RAGS 3 /* RAGS Graphics Editor format */
#define CHUNK_IMAGE75 4 /* Bitmap Image -- */
#define CHUNK_IMAGE 5 /* Bitmap Image */
#define NEVER_USED 6 /* CHUNK_IMAGE100 -- */
75 dpi resolution */

A–2
#define CHUNK_DDIF 7 /* DDIF format */
#define CHUNK_POSTSCRIPT 8 /* Postscript format */
#define MAX_CHUNK_TYPE 8 /* */
/* Data chunk types */
/* */
#define MIN_DATA_CHUNK 18
#define MAIN_CHUNK 18 /* main level data chunk */
#define SUB_CHUNK 19 /* data sub chunk */
#define REFERENCE_CHUNK 20 /* cross reference */
#define MAX_DATA_CHUNK 20 /* */
/* */
/* Topic types */
/* */
#define MIN_TOPIC_TYPE 1
#define STANDARD 1 /* mainline topic */
#define REFERENCE 2 /* formal reference topic */
#define MAX_TOPIC_TYPE 2 /* */
/* */
/* Directory flags */
/* */
#define CONTENTS_MASK 1 /* True for Table of Contents */
#define INDEX_MASK 2 /* True for main Index */
#define DEFAULTMASK 4 /* True for default directory */
#define MULTI_VALUED_MASK 8 /* True if multiple hits allowed */
/* bit mask for standard table of contents */
#define TOC_FLAGS 5 /* bitmask for standard index */
#define INDEX_FLAGS 10 /* */
/* returned value type */
typedef unsigned long int BWI_OBJECT_ID;
/* error code type */
typedef unsigned long int BWI_ERROR;
/* BWI Error Codes */
#define BwiOK 1 /* Successful completion */
#define Bwi.UndefSymbol 2 /* A symbol was referenced but never defined */
#define BwiInvalidBookId 3 /* Book Identifier is Invalid */
#define BwiInvalidTopicId 4 /* Topic Identifier is Invalid */
#define BwiInvalidUpdate 5 /* An error was encountered when updating the file */
#define BwiFailFileOut 6 /* OBSOLETE */
#define BwiFailFileRead 7 /* Error encountered when reading the file */
#define BwiTopicWritten 8 /* Once a Topic is closed it can't be updated */
#define BwiInvalidChunkId 9 /* Chunk Identifier is Invalid */
#define BwiFailHeadCreate 10 /* Error when creating the file header */
#define BwiFailHeadLength 11 /* Indicates the header has overflowed its fixed */
#define BwiBadArg 12 /* Expected parameter is missing or NULL */
#define BwiErrNoMemoryNum 13 /* Memory allocation failure (Internal) */
#define BwiErrInvalidMemNum 14 /* Memory de-allocation failure */
#define BwiStrTooLong 15 /* String has exceeded defined limits */
#define BwiFailMalloc 16 /* Memory allocation failure (generic) */
#define BwiFailFileOpen 17 /* Error encountered when opening the file */
#define BwiFailFileWrite 18 /* Error encountered when writing to the file */
#define BwiDirLevelSkip 19 /* Directory levels MUST be in single ascending */
#define BwiInvalidDirId 20 /* Directory Identifier is Invalid */
/*
 * *********************************************************************************
 * COPYRIGHT (c) 1989 BY
 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
 * ALL RIGHTS RESERVED
 * This software is furnished under a license and may be used and copied
 * only in accordance with the terms of such license and with the
 * inclusion of the above copyright notice. This software or any other
 * copies thereof may not be provided or otherwise made available to any
 * other person. No title to and ownership of the software is hereby
 * transferred.
 * The information in this software is subject to change without notice
 * and should not be construed as a commitment by Digital Equipment
 * Corporation.
 * Digital assumes no responsibility for the use or reliability of its
 * software on equipment which is not supplied by Digital.
 * *********************************************************************************/

/*
 *>Title:  BWI-TEST
 *>Abstract:  This module fakes what the device converter does
 *>Author:  Mike Fitzell
 **/

/* PROGRAM-WIDE STUFF*/

/* include <stdlib.h> */
#include <stdio.h>
#include <ctype.h>
#include "bwi_incl.h"
#include "metaformat.h"

#define WIDTH 50
#define Y_INCR 200

main()
{
  char *filename="psuedobook.decw_book";
  int book_id;
  char *font_name="-ADOBE-Helvetica-Medium-R-Normal--*-140-*-*-P--*

...
int word_length,x,y,i;
char *copyright="copyright";
char *DEC=" 1990 Digital Equipment Corporation";
char *ARR=" All Rights Reserved";
char *bookname="A Very Short Book";
char *contents="contents";
char *title_page="Title Page";
char *data="GOOD MORNIIING NEW HAMPSHIRE";
int pagerefarray[20];
int pagect,entlevel,width,height;
char *header="GREETINGS";
char *help="I’m NOT making this up !";
char *sqrt="sqrt";
int status;

book_id = book$create(filename);

book$copyright(1,0);

book$font(0,font_name);
topic$create(0);
chunk$create();
y=252;
x=200;
text$new_word(x,y);

word_length = strlen(copyright);
for(i=0;i<word_length;i++) {
x += WIDTH;
text$addchar(copyright[i],x);
}

text$new_word(x,y);
word_length = strlen(DEC);
for(i=0;i<word_length;i++) {
x += WIDTH;
text$addchar(DEC[i],x);
}

chunk$close("1","",2266,445);
/*348: eop */
/*349: beginning of page 2 */

y = 813;
x = 146;
chunk$create();

book$title(bookname); /* VWI_BOOK_TITLE(MUTANT RADICAL) */
book$symbol(bookname); /* VWI_BOOK_SYMBOL(MUTANTS)*/
text)new_word(x,y);
word_length = strlen(bookname);
for(i=0;i<word_length;i++) {
x += WIDTH;
text$addchar(bookname[i],x);
}
chunk$close("2","",2266,838);
/*607: eop */
/* let's fake a contents file */

directory$create(DIR_CONTENTS, contents);
directory$startentry();
pagerefarray[0] = 0;
entlevel = 1;
width = 1428;
height = 169;
pagect = 0;
x = 2;
y = 170;
text$new_word(x, y);
word_length = strlen(contents);
for (i = 0; i < word_length; i++) {
    x += WIDTH;
text$addchar(contents[i], x);
}
status = directory$endentry(pagect, &pagerefarray[0], entlevel, width, height, contents);
directory$startentry();
x = 69;
y = 91;
text$new_word(x, y);

word_length = strlen(title_page);
for (i = 0; i < word_length; i++) {
    x += WIDTH;
text$addchar(title_page[i], x);
}
pagect = 1;
pagerefarray[0] = 1;
height = 136;

status = directory$endentry(pagect, &pagerefarray[0], entlevel, width, height, title_page);
directory$startentry();
pagerefarray[0] = 4;
height = 103;
x = 36;
y = 75;
text$new_word(x, y);

word_length = strlen(data);
for (i = 0; i < word_length; i++) {
    x += WIDTH;
text$addchar(data[i], x);
}
status = directory$endentry(pagect, &pagerefarray[0], entlevel, width, height, data);
directory$close(1);

/*/686: xxx'bookr:dvifile MSQR_CONTENTS */

chunk$create();
chunk$close("3", ",", 0, 0);
topic$close(1); /* topic id should be 5 */
topic$create(0);

/*/724: beginning of page 4 */
chunk$create();
y = 252;
x = 79;
text\$new\_word(x,y);
word\_length = strlen(header);
for(i=0;i<word\_length;i++) {
  x += WIDTH;
  text\$addchar(header[i],x);
}
y += Y\_INCR;

chunk\$close(sqrt, data, 2266, 465);
topic\$close(1);
book\$close();
/*
   ************************************************************************
   COPYRIGHT (c) 1989 BY
   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
   ALL RIGHTS RESERVED
   *
   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
   ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
   INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
   COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
   OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
   TRANSFERRED.
   *
   THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
   AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
   CORPORATION.
   *
   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
   SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
   *
   ************************************************************************
*/

/*
   TITLE:
   XVWINF
   ABSTRACT:
   This module contains the routines and data storage used
   to interact with the VWI.
   AUTHOR:
   Mark DeVries
   */

nero

#include <stdio.h>
#include <ctype.h>
#include "bwi_incl.h"
#include "metaformat.h"

#define TRUE 1
#define FALSE 0
#define MAXINT (65535)

globaldef struct build_status STATE;

typedef struct _TOPW {
    unsigned topid; /*TOPIC OR DIRECTORY ID*/
} TOPWORK;

globaldef TOPWORK towpkarea[NUM_TOPIC_TYPES];
globaldef int curtoptyp; /*THIS IS INDEX TO TOPICS*/
globaldef unsigned lost_page_ending[20]; /* keeps track of lost page endings */
globaldef int ipe_count;
globaldef unsigned intfbookid;

C-1
int chunk$init_storage();
void chunk$free_buffers();

void book$abort()
{/*----------------*/
  VVI_BOOK_ABORT(intfbookid);
}

int book$create(outfilename)
  char *outfilename;
{/*---------------------*/
  int status,i;
  intfbookid = VVI_BOOK_CREATE(outfilename);
  if(intfbookid == NULL) {
    book$abort();
    return(dvc$_bookabort); /*
  }
  if((status = chunk$init_storage()) != TRUE) {
    book$abort();
    return(status);
  }
  for (i=0; i<NUM_TOPIC_TYPES; i++)
    topwkarea[i].topid = 0;
  for(i=0; i<20;i++)
    lost_page_ending[i] = 0;
  lpe_count = 0;
  return(status);
}

int book$symbol(symbol)
  char *symbol;
{/*---------------------*/
  VVI_BOOK_SYMBOL(intfbookid,symbol);
  return(TRUE);
}

int book$close()
{/*----------------*/
  int status;
  status = VVI_BOOK_CLOSE(intfbookid);
  (void)chunk$free_buffers();
  return(status);
}

void book$copyright(cid, TBS)
  unsigned cid, TBS;
{/*---------------------*/
  VVI_BOOK_COPYRIGHT(intfbookid, cid, TBS);
}

void book$font(ftnum, fname)
  unsigned ftnum;
  char *fname;
{/*---------------------*/
  VVI_BOOK_FONT(intfbookid, ftnum, fname);
}
void book$license(producer, product, dayte, version)
    char *producer;
    char *product;
    unsigned dayte;
    int version;
    /*-----------------------------*/
    { unsigned date[2] = {0, 0};
        VWI_BOOK_LICENSE (intfbookid, producer, product, date, version);
    }
void book$licensealt(alt_name)
    char *alt_name;
    /* this is for LMF alternate names */
    VWI_BOOK_LICENSE_ALT(intfbookid, alt_name );
}
void book$title(bkname)
    char *bkname;
    /*-----------------------------*/
    { VWI_BOOK_TITLE(intfbookid, bkname); }
int shelf$create(filespec, symbolname, title)
    char *filespec;
    char *symbolname;
    char *title;
    { return(VWI_SHELF_CREATE(filespec, symbolname, title)); }
int shelf$entry(shelfid, type, filespec, symbolname, title, shelflevel)
    int shelfid;
    int type;
    char *filespec;
    char *symbolname;
    char *title;
    int shelflevel;
    { return(VWI_SHELF_ENTRY(shelfid, type, filespec, symbolname, title, shelflevel)); }
void shelf$close(shelfid)
    int shelfid;
    { VWI_SHELF_CLOSE(shelfid); }
/*********************************************
/* BOOKSHELF STUFF */
/*********************************************
*/ TOPIC-LEVEL STUFF */
/*************************************/
/* The term "TOPIC" in the document is equivalent to "PAGE" in VWI. */
/*************************************/
/* TOPIC WORK BUFFERS */
/* There is a separate work area for each topic type: */
/* TOPIC_MAINLINE */
/* TOPIC_FORMAL */
/* TOPIC_DIRECTORY */
/* That will allow us to retain mainline topic info, such as ID, */
/* after an interruption for formal or directory chunks. */
unsigned topic$create(ttype)
    unsigned ttype;
    /*---------------------------*/
    /* Create a VWI page in the current book, initialize local storage, */
    /* and return the VWI page id.                                */
    /*---------------------------*/
    {
        int topicid;
        unsigned vwipgtype;

        if (ttype == TOPIC_FORMAL)
            vwipgtype = REFERENCE;
        else
            vwipgtype = STANDARD;

        topicid = VWI_TOPIC_CREATE(intfbookid, vwipgtype);
        curtoptyp = ttype;
        if(topwkarea[curtoptyp].topid != 0) /*page hasn't been closed yet */
            lost_page_ending[++lpe_count] = topwkarea[curtoptyp].topid;
        topwkarea[curtoptyp].topid = topicid;
        return(topicid);
    }

void topic$close(endflag)
    int endflag;
    /*---------------------------*/
    {
        TOPWORK *tptr = &topwkarea[curtoptyp];
        if(!tptr->topid) { /* if 0 it's already closed */
            VWI_TOPIC_CLOSE(intfbookid, tptr->topid);
            curtoptyp = TOPIC_MAINLINE;
            tptr->topid = 0;
        }
        if(endflag) { /* cleans up pages that never got closed */
            while(lost_page_ending[lpe_count]) {
                VWI_TOPIC_CLOSE(intfbookid, lost_page_ending[lpe_count--]);
            }
        }
    }

void topic$next(curtid, nexttid)
    unsigned curtid, nexttid;
    /*---------------------------*/
    {
        VWI_TOPIC_NEXT(intfbookid, curtid, nexttid);
    }

void topic$previous(curtid, prevtid)
    unsigned curtid, prevtid;
    /*---------------------------*/
    {
        VWI_TOPIC_PREVIOUS(intfbookid, curtid, prevtid);
    }

/*********************************************************************/
/* CHUNK-LEVEL STUFF                                               */
/* The term "PAGE" in the document is equivalent to "CHUNK" in VWI. */
/*********************************************************************/
/*EXTERN int PCpage_counter; */
#define CHUNKALLOCMIN (64 * 1024)
#define CHUNKALLOCINCR (64 * 1024)
globaldef unsigned cbid;
globaldef int cbtype;
globaldef char *cbufaddr;
globaldef int cbuflen;
globaldef int cballocsize;
globaldef int cbcurX;
globaldef int cbcurY;
globaldef int cbcurf;
globaldef int cbxoffset;
globaldef int cbyoffset;
globaldef int cbstartpkt;
globaldef int cbstartword;
globaldef char *direntrytitle;
globaldef int direntryalloc;

int chunk$create()
/*
/*====================*/
/* Allocate or reuse a chunk buffer; init key variables. */
/* There may be no more than one chunk at a time for each type of topic. */
/*--------------------*/
{
    cbtype = CHUNK_FTEXT; /*ASSUME DEFAULT TYPE*/
    if (cbufaddr == NULL) {
        cbufaddr = malloc(CHUNKALLOCMIN);
        if(cbufaddr == NULL)
            return(FALSE);
        cballocsize = CHUNKALLOCMIN;
    }
    cbuflen = 0;
    cbcurX = cbcurY = MAXINT;
    cbcurf = -1;
    cbxoffset = cbyoffset = 0;
    cbstartpkt = cbstartword = 0;
    return(TRUE);
}

int chunk$close(sym, title, width, height)
char *sym, *title;
unsigned width, height;
/*--------------------*/
{
    int topic_idnum;
    char message[256];
    TOPWORK *tptr = &topwkarea[curtoptyp];
    unsigned char *offender;
    if(!tptr->topid && lpe_count)
        topic_idnum = lost_page_ending[lpe_count];
    else
        topic_idnum = tptr->topid;
    /* if(!topic_idnum) {
        offender = CTITLE[0];
        (void) sprintf(message,"TeXpage = %d, ChunkID = %d Last meaningful text = %s",
                        PCpage_counter, cbid, offender );
        (void) error_parse(dvc$_pagecoordlost,0,message,TRUE);
        return(dvc$_bookabort);
    } */
    /* if(*title == 0)
        (void)sprintf(title,"%d",cbid); */
    cbid = VWI_TOPIC_DATA_CHUNK(intfbookid, topic_idnum, sym, title,
                                width, height, cbufaddr, cbuflen, cbtype);
    /* if (cbid != PCpage_counter) {
        (void) sprintf(message,"TeXpage = %d, ChunkID = %d\n",
                        PCpage_counter, cbid);
        printf("%s\n",message); */
/* (void) error_parse(dvc$_pagecoordlost,0,message,TRUE); 
  return(dvc$_bookabort); */
return(TRUE);
}

void chunk$reference(numpoints,array_points,targetsym,type,shape)
unsigned numpoints;
struct POINTS (*array_points)[];
char *targetsym;
int type;
int shape;
{
  unsigned xoff, yoff, chwidth, chheight;
  struct POINTS *pt1,*pt2;
  if(shape == RECTANGLE) {
    pt1 = array_points; /* using the upper left and */
    *pt1++; /* lower right points of the */
    pt2 = pt1 + 2; /* box compute the width and */
    chwidth = (pt2->x - pt1->x) + 2; /* height */
    chheight = (pt2->y - pt1->y) + 4;
    xoff = pt1->x;
    yoff = pt1->y - 2;
    if(type == HOT_END)
      VWI_TOPIC_REFERENCE_RECT(intfbookid, cbid, xoff, yoff, chwidth,
                                chheight, targetsym);
    else
      VWI_TOPIC_EXTENSION_RECT(intfbookid, cbid, xoff, yoff, chwidth,
                                chheight);
  } else {
    if(type == HOT_END)
      VWI_TOPIC_REFERENCE_POLY(intfbookid, cbid, numpoints,
                              array_points, targetsym);
    else
      VWI_TOPIC_EXTENSION_POLY(intfbookid, cbid, numpoints,
                              array_points);
  }
}

void chunk$free_buffers()
/*
 */
/*------------------------*/
{ if (cbufaddr != NULL) {
    free(cbufaddr);
    cbufaddr = NULL;
  } if (direntrytitle != NULL) {
    free(direntrytitle);
    direntrytitle = NULL;
  }
}

int chunk$init_storage()
/*
 */
/*------------------------*/
{ cbufaddr = NULL;
  direntryalloc = 256;
  direntrytitle = malloc(direntryalloc);
  if(direntrytitle == NULL)
    return(0);
  /* return(dvc$_vmovrflow); */
  return(TRUE);
}
int chunk$makeroomfor(whatchuneed)
    unsigned whatchuneed;
    /*---------------------*/
    /* Allow an extra 100 bytes for sloppiness. */
    /*---------------------*/
    {
        unsigned i, sizeavail;
        char *memptr;

        #define SLOP 100
        sizeavail = cballocsize - (cbuflen + SLOP);
        if (sizeavail < whatchuneed) {
            for (i=1; (sizeavail+(i*CHUNKALLOCINCR))<whatchuneed; i++)
                cballocsize += i * CHUNKALLOCINCR;
            memptr = realloc(cbufaddr, cballocsize);
            if (memptr == NULL) {
                return(FALSE);
            }
            cbufaddr = memptr;
        }
        return(TRUE);
    }

    /***** ROUTINES TO CONTROL SUBCHUNKS *****/
    void subchunk$create(xoffs, yoffs)
        unsigned xoffs, yoffs;
        /*---------------------*/
        /* Allocate or reuse a chunk buffer as a subchunk; init key variables.
        * Subchunk will always use the same buffer as its parent chunk (since
        * it is in the same topic), and will expect to find the parent chunk ID
        * preserved in the variable cbid. And, of course,
        * cbufaddr will never be NULL, since the buffer would have been
        * allocated when the parent chunk was defined (or perhaps by an earlier
        * chunk).
        */
        /*---------------------*/
    {
        cbtype = CHUNK_FTEXT; /*ASSUME DEFAULT TYPE*/
        cbuflen = 0;
        cbcurX = cbcurY = MAXINT;
        cbcurf = -1;
        cbxoffset = xoffs;
        cbyoffset = yoffs;
        cbstartpkt = cbstartword = 0;
    }

    void subchunk$close(width, height)
        unsigned width, height;
        /*---------------------*/
    {
        VWI_TOPIC_DATA_SUBCHUNK(intfbookid, cbid, cbxoffset, cbyoffset,
            width, height, cbufaddr, cbuflen, cbtype);
    }

    /***** ROUTINES FOR TEXT PACKETS IN *FTEXT* CHUNKS *****/
    #define MINMOVE 1
    #define MAXINLINEMOVE 255
    #define abs(x) ((x >= 0) ?x:- x)
    #define MAXPACKETSIZE 255

    int text$startpkt(exx, wye, font)
        unsigned exx, wye, font;
        /*---------------------*/
    {
        FTEXT *fptr;
        TEXT *txptr;
if(!chunk$makeroomfor(MAXPACKETSIZE))
return(FALSE);

    cbstartpkt = cbflen;
    fptr = &cbufaddr[cbstartpkt];
    fptr->tag = FTEXT_TEXT400;
    txptr = &fptr->value;
    txptr->x = exx;
    txptr->y = wye;
    txptr->font_num = font;
    fptr->len = FTEXT_LENGTH + TEXT_LENGTH;
    cbuflen += fptr->len;
    return(TRUE);
}

int text$new_word(newX, newY)
    int newX, newY;
    /*====================*/
    /* If movement is only horizontal (and small), and the font has not changed:
    /* Insert short, positive horizontal offset.
    /* Else
    /* Define new font, if needed.
    /* Start new text packet with explicit position and font.
    /*====================*/
    {
        TEXT *tptr;
        FTEXT *fptr;
        WORD *wptr;
        int textft = 0; /*currentfont(); */
        int deltaX, NEWFONT;
        deltaX = newX - cbcurX;
        NEWFONT = (textft != cbcurf);
        if ((abs(deltaX) < MINMOVE) && (! NEWFONT) && (newY == cbcurY))
        goto cnw_out; /*IGNORE A MINISCULE MOVE*/
        if ((cbstartword == 0) /*TEXT PACKET UNDERWAY*/
            (newY != cbcurY) /*VERTICAL MOVE*/
            (deltaX > MAXINLINEMOVE) /*LARGE FORWARD HORIZ MOVE*/
            (deltaX < 0) /*BACKWARD HORIZ MOVE*/
            (STATE.hotpacket)
            (NEWFONT)) { /*FONT CHANGE*/
                if(!text$startpkt(newX, newY, textft))
                    return(FALSE); /*START A NEW TEXT PACKET*/
            deltaX = 0; /*DELTA=0 FOR FIRST WORD*/
                    if(STATE.hotpacket)
            STATE.hotpacket = 0; /* force new FTEXT PACKET for hotspots */
        } cbstartword = cbflen;
    wptr = &cbufaddr[cbstartword];
    wptr->delta = (unsigned char) deltaX; /*SET FIRST DELTA = 0*/
    wptr->count = 0; /*INIT WORD LENGTH*/
    cbuflen += WORD_LENGTH;
    fptr = &cbufaddr[cbstartpkt];
    fptr->len = WORD_LENGTH; /*ADD NEW BYTES TO PKTLEN*/
    cbcurX = newX; /*STORE NEW TEXT STATE*/
    cbcurY = newY;
    cbcurf = textft;
        cnw_out:
    return(TRUE);
int text$addchar(texch, newX)
    unsigned char texch;
    int newX;
/*--------------------------*/
/* Store the new character, and update horizontal position beyond it. */
/*--------------------------*/
{
    FTEXT *fptr;
    WORD *wptr;
    if ((cbuflen - cbstartpkt) == MAXPACKETSIZE) {
        cbstartword = 0; /*SIGNAL NEW PACKET NEEDED*/
        if (!text$new_word(cbcurX, cbcurY))
            return(FALSE);
    }
    cbufaddr[cbuflen++] = texch; /*STORE THE CHAR*/
    cbcurX = newX; /*STORE NEW XPOS*/
    wptr = &cbufaddr[cbstartword];
    wptr->count++; /*INCR WORD LENGTH*/
    fptr = &cbufaddr[cbstartpkt];
    fptr->len++; /*INCR PKT LENGTH*/
    return(TRUE);
}

/***** ROUTINE FOR RULE PACKETS IN "FTEXT" CHUNKS *****/
int rule$store(rx0, ry0, rx1, ry1)
    int rx0, ry0, rx1, ry1;
/*--------------------------*/
/* RULE *rptr; */
/* FTEXT *fptr; */

if (! chunk$makeroomfor(MAXPACKETSIZE))
    return(0);
/* return(dvc$_vmovrflow); */
 cbstartpkt = cbuflen;
 fptr = &cbufaddr[cbstartpkt];
 fptr->tag = FTEXT_RULE;
 fptr->len = FTEXT_LENGTH + RULE_LENGTH;
 rptr = &fptr->value;
 rptr->x = rx0;
 rptr->y = ry0;
 rptr->height = ry1 - ry0;
 rptr->width = rx1 - rx0;
 cbuflen += fptr->len;
 cbcurX = cbcurY = MAXINT;
 cbstartword = 0;
 return(TRUE);
}

/***** ROUTINE FOR "RAGS" CHUNKS *****/
int rags$store(rptr, rlen)
    FILE *rptr;
    unsigned int rlen;
/*--------------------------*/
/* STORE A RAGS GRAPHIC! (Which means store its filename only.) */
/*--------------------------*/
{
    if (! chunk$makeroomfor(rlen))
        return(0);
/* return(dvc$_vmovrflow); */
    cbuflen = fread(cbufaddr,1,(int)rlen,rptr);
    if (!cbuflen)
        return(0);
/* return(dvc$_fignotfnd); */
    cbtype = CHUNK_RAGS;
    return(TRUE);
}
return(TRUE);
}

/***** ROUTINES FOR "IMAGE..." CHUNKS *****/

void image$start(gtype,pixwid,pixheight)
    int gtype;
    short pixwid;
    short pixheight;
    /*------------------------*/
    /* START AN IMAGE RECORD.
    /*------------------------*/
    {
        IMAGE *imag;
        short resolution;
        cbtype = CHUNK_IMAGE;
        if (gtype == PLOT_IMAGE)
            resolution = 75;
        else if (gtype == PLOT_IMAGE75)
            resolution = 75;
        else if (gtype == PLOT_IMAGE100)
            resolution = 100;
        imag = &cbufaddr[cbuflen];
        imag->res_x = resolution;
        imag->res_y = resolution;
        imag->pix_width = pixwid;
        imag->pix_height = pixheight;
        cbuflen += IMAGE_LENGTH;
    }

int image$store(bufptr, buflen)
    char *bufptr;
    unsigned int buflen;
    /*------------------------*/
    /* STORE A BLOCK OF IMAGE DATA.
    /*------------------------*/
    {
        int i;
        if(!chunk$makeroomfor(buflen))
            return(0);
        /* return(dvc$_vmovrflow); */
        for (i=0; i<buflen; i++)
            cbufaddr[cbuflen++] = bufptr[i];
        return(TRUE);
    }

/**************************************************************************/
/* DIRECTORY-LEVEL STUFF
/* The term "DIRECTORY" refers to TABLE OF CONTENTS, INDEX, ETC.
/* Directories are handled here much the same as topics.
/**************************************************************************/

globaldef int tempdirid = 0;
void directory$create(dirtype, dirnamstr)
    unsigned dirtype;
    char *dirnamstr;
    /*------------------------*/
    {
        int dirid;
        unsigned dirflags;

if (dirtype == DIR_CONTENTS) {
    /* dirnamstr = "TABLE OF CONTENTS"; /* TEMP TEMP TEMP*/
    dirflags = CONTENTS_MASK | DEFAULT_MASK;
} else if (dirtype == DIR_INDEX) {
    /* dirnamstr = "INDEX"; /* TEMP TEMP TEMP*/
    dirflags = INDEX_MASK | MULTI_VALUED_MASK;
} else {
    dirflags = 0;
}

dirid = VWL_DIRECTORY_CREATE(intfbookid, dirnamstr, dirflags);
if (curtoptyp == TOPIC_DIRECTORY)
    tempdirid = topwkarea[curtoptyp].topid;
curtoptyp = TOPIC_DIRECTORY;
    topwkarea[curtoptyp].topid = dirid;
}

void directory$close(end)
int end; /* true means end of toc */
/*
/*====================*/
{
    VWL_DIRECTORY_CLOSE(intfbookid, topwkarea[curtoptyp].topid);
    if (tempdirid) {
        topwkarea[curtoptyp].topid = tempdirid;
        tempdirid = 0;
        if (end) {
            VWL_DIRECTORY_CLOSE(intfbookid, topwkarea[curtoptyp].topid);
            curtoptyp = TOPIC_MAINLINE;
        }
    } else
        curtoptyp = TOPIC_MAINLINE;
}

int directory$startentry()
/*====================*/
/* (A directory entry uses a data buffer in the same way a chunk does.
/* This routine is analogous to (and probably identical to) chunk$create.)
/* Allocate or reuse a chunk buffer; init key variables.
/* There may be no more than one directory entry at a time.
/*====================*/
{
    cbtype = CHUNK_FTEXT; /* ASSUME DEFAULT TYPE*/
    if (cbufaddr == NULL) {
        cbufaddr = malloc(CHUNKALLOCMIN);
        if (cbufaddr == NULL)
            return(0);
        /* return(dvc$_vmovrflow); */
        cballocsize = CHUNKALLOCMIN;
    }
    cbuflen = 0;
    cbcurX = cbcurY = MAXINT;
    cbcurf = -1;
    cbxoffset = cbyoffset = 0;
    cbstartpkt = cbstartword = 0;
    return(TRUE);
}

int directory$endentry(refct, reflistptr, level, width, height, title)
unsigned refct, *reflistptr, level, width, height;
char *title;
/*====================*/
{
    TOPWORK *tptr = &topwkarea[curtoptyp];
}
int titlen = strlen(title);
    if ((titlen+1) > direntryalloc) {
        direntryalloc = titlen + titlen;
        direntrytitle = realloc(direntrytitle, direntryalloc);
        if(direntrytitle == NULL)
            return(0);
    } /* return(dvc$_vmovrflow); */
    (void) strcpy(direntrytitle, title); /*COPY TITLE STRING*/
    VWI_DIRECTORY_ENTRY(intfbookid, tptr-
    >topid, refct, reflistptr, level,
    width, height, cbufaddr, cbuflen, cbtype, direntrytitle);
    if(tempdirid) /* building two directories */
        VWI_DIRECTORY_ENTRY(intfbookid,tempdirid,refct, reflistptr, level,
        width, height, cbufaddr, cbuflen, cbtype, direntrytitle);
    return(TRUE);
/*****************************************************************************/
/** *
** COPYRIGHT (c) 1989 BY *
** DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS. *
** ALL RIGHTS RESERVED *
** *
** THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
** ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
** INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
** COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
** OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
** TRANSFERRED. *
** *
** THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
** AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
** CORPORATION. *
** *
** DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
** SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
** *
*****************************************************************************/

#include <dvi.h>

#define ID_BYTE 2 /* current version of the dvi format */
#define SET_CHAR_0 0 /* typeset character 0 and move right */
#define SET_CHAR_127 127 /* typeset character 127 and move right */
#define SET1 128 /* typeset a character and move right */
#define SET4 131 /* typeset a character and move right */
#define SET_RULE 132 /* typeset a rule and move right */
#define PUT1 133 /* typeset a character */
#define PUT_RULE 137 /* typeset a rule */
#define NOP 138 /* no operation */
#define BOP 139 /* beginning of page */
#define EOP 140 /* ending of page */
#define PUSH 141 /* save the current positions */
#define POP 142 /* restore previous positions */
#define RIGHT1 143 /* move right */
#define W0 147 /* move right by |w| */
#define W1 148 /* move right and set |w| */
#define X0 152 /* move right by |x| */
#define X1 153 /* move right and set |x| */
#define DOWN1 157 /* move down */
#define Y0 161 /* move down by |y| */
#define Y1 162 /* move down and set |y| */
#define Z0 166 /* move down by |z| */
#define Z1 167 /* move down and set |z| */
#define FNT_NUM_0 171 /* set current font to 0 */
#define FNT_NUM_63 234 /* set current font to 63 */
#define FNT1 235 /* set current font (number is in parameter) */
#define XXX1 239 /* extension to dvi primitives \special */
#define XXX4 242 /* potentially long extension to dvi primitives */
#define FNT_DEF1 243 /* define a font */
#define PRE 247 /* preamble */
#define POST 248 /* postamble beginning */
#define POST_POST 249 /* postamble ending */
#define UNDEFINED_COMMAND 250
/***** ERROR HANDLING *****/
/* The globalvalues are ID's of user msgs stored in an external message
/* object file. The longword passes the length of a string argument to
/* such a message. */

/***** THE txf STRUCTURE STORES INFORMATION FOR ONE TEX FONT. PICTORIALLY:
/* +---+---+------+-------+-------+------+------+
/* | bc| ec| space| design| scaled| *font| *font|
/* | | | | size | size | name| width|
/* +---+---+------+-------+-------+------+------+
/**/
struct txf {unsigned bc; /*beginning char (lowest code)*/
  unsigned ec; /*ending char (highest code)*/
  int space;
  int design_size;
  int scaled_size;
  char *font_name; /*ptr to filename*/
  int *font_width; /*ptr to table of widths*/
  int *font_height; /* ptr to table of heights*/
  int *font_depth; /* ptr to table of depths*/
  int bbox_depth; /* max depth in font */
};
#define GLOBAL globaldef
#define EXTERN globalref

/******* MACRO definitions ******************************************/
#define max(x,y) (((x)>(y))?(x):(y))
#define min(x,y) (((x)<(y))?(x):(y))
#define pixel_round(x) ((int) ((x<0) ? (conv*(x)-0.5) : (conv*(x)+0.5)))

/*****VALUES TO CONTROL \special PROCESSING *****/
#define spec_execute 1
#define spec_noexecute 2
#define DEFPOINTSPECL 0
#define CONNECTSPECL 1
#define PLOTFILESPECL 2
#define RESETPOINTSSPECL 3
#define DVIFILESPECL 4
#define ONLINEOCSPSPECL 5
#define ONLINEIDXXSPECL 6
#define ONLINECHUNK 7
#define HOTSPOT 8
#define CHUNKTITLE 9
#define EXTENSION 10
#define LMF 11
#define SHELF 12
#define BOOK_SYMBOL 13
/* NOSPECIALS is, ambiguously, "number of specials" & "no specials" */
#define NOSPECIALS 14
/* SPECIALOUTPUTERROR means we got an I/O error writing to O/P file */
#define SPECIALOUTPUTERROR 65535

/***** DATATYPES FOR GRAPHICS *****/
#define PLOT_RAGS 1
#define PLOT_IMAGE 11
#define PLOT_IMAGE75 1175
#define PLOT_IMAGE100 11100
#define CHILD_REFERENCE 2
#define PIXELS_PER_INCH (400.0)
#define pts2pixels(n) (n * 400 / 72)
#define scaledptsperinch (4718592.0)
#define pixelsperinch (400.0)
#define scaledptsperpixel (scaledptsperinch / pixelsperinch)
#define centimetersperinch (2.54)

/***** TYPES OF DIRECTORIES: Need not be contiguous *****/
#define DIR_CONTENTS 1
#define DIR_INDEX 2

/***** TYPES OF TOPICS: Must be contiguous from zero *****/
#define TOPIC_MAINLINE 0
#define TOPIC_FORMAL 1
#define TOPICDIRECTORY 2
#define NUM_TOPIC_TYPES 3

/******* HOTSPOT AND EXTENSION DEFINES ****************************/
#define RECTANGLE 1
#define POLYGON 2
#define HOT_START 0
#define HOT_END 1
#define EXTENSION_START 2
#define EXTENSION_END 3

/******* (IN_prefix INCLUDE) *************/
/* things to control including DVI files */
#define IN_PRIMARYDVI 0
#define IN_INCLUDEDVI 1
#define IN_MAXDVISEQ 20

/*********** Instead of 8 different stacks we have an array of structs *****/
struct coordinates {
  int h;       /*horizontal position in TeX scaled points*/
  int v;       /*vertical position in TeX scaled points*/
  int x;
  int y;
  int z;
  int w;
  int hh;      /*pixel-rounded version of 'h'*/
  int vv;      /*pixel-rounded version of 'v'*/
  int hoff;    /*horizontal offset of origin, in pixels*/
  int voff;    /*vertical offset of origin, in pixels*/
};

/******* NEW struct for defining vector points ***********/
struct POINTS {
  int x;
  int y;
};

/** defines a region of text for hotspots or extensions *****/
struct region_t{
  int xorg;
  int yorg;
  int width;
  int height;
  int depth;
  int numpoints;
  struct POINTS pt_vec[8];
  int delete_flag;
  int shape;
  int type;
  char *symbol_name;
  struct region_t *next;
  struct region_t *prev;
};
***** DATA STRUCTURE TO HOLD DESCRIPTION OF DEFERRED RECORD *****
A DEFERRED RECORD MAY BE A PLOTFILE OR A HOTSPOT */

struct DEFERREDREC {
    struct DEFERREDREC *dfr_nextrec;
    int dfr_datatype;
    int dfr_bbox[4]; /*Xtopleft, Ytopleft, Xbotright, Ybotright */ /*in units specific to datatype */
    int dfr_offX; /*X and Y offset from start of parent chunk*/
    int dfr_offY;
    int dfr_normalwidth; /*Width and height in "standard" units */
    int dfr_normalheight;
    int dfr_crop_flag;
    unsigned char dfr_crop_value;
    int dfr_type; /* HOTSPOT or EXTENSION */
    int dfr_numpoints; /* number of x/y coord in this region */
    struct POINTS (*dfr_pt_vec)[8]; /* ptr to array of coordinates that */
    /* define a region */
    char *dfr_filnam; /*Ptr to name of file containing graphic */
};

#define MAXTEXFONTS 512 /* used to be 100 */
#define DEFAULT_CHARSET 0
#define TEXMATH_REMAP 99
#define PAGE_INCR 2000;

struct enumtable {
    int enumdef[MAXTEXFONTS+1]; /*TRUE means font defined in curr DVI*/
    int enumval[MAXTEXFONTS+1]; /*DVI's font number for this font*/
};

struct box{
    int xorg;
    int yorg;
    int width;
    int height;
    int depth;
};

/** Put all the status and mode flags in one struct so we can look **/
/** in one place to get the status or mode of the build **/

struct build_status {
    int hot_spot;
    int hot_spot_type;
    int hotpacket;
    int numpoints;
    int saveveebox;
    int setposition;
    int saveveeboxctr;
    int num_fonts; /*number of fonts defined so far*/
    int current_font; /*internal number of current font*/
    int building_dir; /* building a directory if TRUE */
    int insubentry;
    int extracttitle; /* grabbing ascii title */
    int indextitle; /* grabbing index title for deferrment*/
    int endformalpending; /* formal topic closed waiting for EOP*/
    int newchunkneeded; /* just what it says */
};
# Index

## B
- Bitmapped graphics • 1–3
- bwi_book_abort • 2–2
- bwi_book_close • 2–3
- bwi_book_copyright • 2–4
- bwi_book_create • 2–5
- bwi_book_first_topic • 2–6
- bwi_book_font • 2–7
- bwi_book_license • 2–8
- bwi_book_license_alt • 2–9
- bwi_book_symbol • 2–10
- bwi_book_title • 2–11
- bwi_directory_close • 2–12
- bwi_directory_create • 2–13
- bwi_directory_entry • 2–15
- bwi_symbol_define • 2–17
- bwi_topic_close • 2–18
- bwi_topic_create • 2–19
- bwi_topic_data_chunk • 2–20
- bwi_topic_data_subchunk • 2–22
- bwi_topic_extension_poly • 2–30
- bwi_topic_extension_rect • 2–26
- bwi_topic_next • 2–31
- bwi_topic_previous • 2–32
- bwi_topic_reference_poly • 2–28
- bwi_topic_reference_rect • 2–24

## D
- Display PostScript • 1–3

## F
- FSE • 1–3
- Ftext packets • 1–2
  - rule • 1–2
  - text • 1–2
- RAGS • 1–3
- Subchunks • 1–3