```c
/*
 * AKFAvatar library - for giving your programs a graphical Avatar
 * Copyright (c) 2007,2008,2009,2010,2011,2012,2013
 * Andreas K. Foerster <info@akfoerster.de>
 *
 * required standards: C99
 *
 * other software:
 * required:
 *  SDL1.2 (SDL1.2.11 or later (but not 1.3 or 2.0!))
 * optional/deprecated:
 *  SDL_image1.2 (support may be removed in future versions)
 *
 * This file is part of AKFAvatar
 *
 * AKFAvatar is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * AKFAvatar is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see <http://www.gnu.org/licenses/>.
 */

/*
 * Table of contents
 *
 * - definitions
 * - initialization / finalization
 * - setting an avatar image or a banner
 * - actions without or outside the balloon
 * - say or ask stuff
 * - character encodings
 * - handling of single characters
 * - key or event handling
 * - informational stuff
 * - colors
 * - settings
 * - handle coordinates inside the balloon
 * - activities inside the balloon
 * - showing images without the avatar
 * - high-level functions
 * - plumbing
 * - audio output
 * - experimental
 * - deprecated functions - only for backward comatibility
 */

#ifndef AKFAVATAR_H
#define AKFAVATAR_H

/* to get the systems definition of wchar_t */
#include <stddef.h>

#if !defined(__cplusplus) && !defined(__bool_true_false_are_defined)
#include <stdbool.h>
#endif

#define AKFAVATAR 1

/* maximum linelength */
#define AVT_LINELENGTH 80
```

```
/* for avt_start */
#define AVT_AUTOMODE -1
#define AVT_WINDOW 0
#define AVT_FULLSCREENNOSWITCH 2

#ifdef AVT_NOSWITCH
#  define AVT_FULLSCREEN AVT_FULLSCREENNOSWITCH
#else
#  define AVT_FULLSCREEN 1
#endif

/* status */
#define AVT_NORMAL 0
#define AVT_QUIT 1
#define AVT_ERROR -1
#define AVT_FAILURE -2  /* nonfatal failures */

/* for avt_set_text_delay and avt_set_flip_page_delay */
#define AVT_DEFAULT_TEXT_DELAY 75
#define AVT_DEFAULT_FLIP_PAGE_DELAY 2700

/* for avt_text_direction */
#define AVT_LEFT_TO_RIGHT 0
#define AVT_RIGHT_TO_LEFT 1

/* characters used for invalid input */
#define AVT_INVALID_CHAR '\x1A'
#define AVT_INVALID_WCHAR L'\uFFFD'

/*
 * for avt_get_key()
 * note: AKFAvatar reserves the codepoints 0xEA00-0xEFFF for function keys
 * note: F11 is normally used to toggle fullscreen mode!
 * note: most keyboards don't have all those keys
 */
#define AVT_KEY_NONE      0x0000  /* on error or quit request */
#define AVT_KEY_ENTER     0x000D
#define AVT_KEY_ESCAPE    0x001B  /* only if single keys are reserved */
#define AVT_KEY_BACKSPACE 0x0008
#define AVT_KEY_DELETE    0x007F
#define AVT_KEY_UP        0xEA00
#define AVT_KEY_DOWN      0xEA01
#define AVT_KEY_RIGHT     0xEA02
#define AVT_KEY_LEFT      0xEA03
#define AVT_KEY_INSERT    0xEA04
#define AVT_KEY_HOME      0xEA05
#define AVT_KEY_END       0xEA06
#define AVT_KEY_PAGEUP    0xEA07
#define AVT_KEY_PAGEDOWN  0xEA08
#define AVT_KEY_HELP      0xEA09
#define AVT_KEY_MENU      0xEA0A
#define AVT_KEY_F1        0xEAF1
#define AVT_KEY_F2        0xEAF2
#define AVT_KEY_F3        0xEAF3
#define AVT_KEY_F4        0xEAF4
#define AVT_KEY_F5        0xEAF5
#define AVT_KEY_F6        0xEAF6
#define AVT_KEY_F7        0xEAF7
#define AVT_KEY_F8        0xEAF8
#define AVT_KEY_F9        0xEAF9
#define AVT_KEY_F10       0xEAFA
#define AVT_KEY_F11       0xEAFB  /* may be reserved for toggling fullscreen */
#define AVT_KEY_F12       0xEAFC
#define AVT_KEY_F13       0xEAFD
#define AVT_KEY_F14       0xEAFE
#define AVT_KEY_F15       0xEAFF

/* example: avt_wait(AVT_SECONDS(2.5)) waits 2.5 seconds */
```

```c
#define AVT_SECONDS(x) ((x)*1000)

#ifdef __cplusplus
extern "C" {
#endif

#define AVT_API  extern

/*************************************************************************/
/* type definitions */

/*
 * type for single characters
 * On Windows wchar_t/wint_t is not large enough for Unicode
 * it should be compatible with char32_t in C11.
 */
typedef unsigned int avt_char;

/* general type for audio data */
typedef struct avt_audio avt_audio;

/* for streams (use FILE from your programs) */
typedef void avt_stream;


/*************************************************************************/
/* functions */
/* most functios return the status, unless otherwise stated */
/*************************************************************************/

/*************************************************************************/
/* initialization / finalization */

/*
 * initialize the avatar system
 *
 * mode is either AVT_WINDOW or AVT_FULLSCREEN or AVT_FULLSCREENNOSWITCH.
 * title and/or shortname may be NULL
 */
AVT_API int avt_start (const char *title, const char *shortname, int mode);

/*
 * reset almost everthing
 */
AVT_API void avt_reset (void);

/*
 * quit the avatar system
 * can be used with atexit
 * it's okay to call it more than once
 */
AVT_API void avt_quit (void);

/*
 * call avt_wait_button (); avt_move_out (); avt_quit ();
 * can be used with atexit
 */
AVT_API void avt_button_quit (void);


/*************************************************************************/
/* setting an avatar image or a banner */

/* On error these functions return AVT_FAILURE without changing the status */

/*
 * supported formats:
 * X-Pixmaps (XPM), X Bitmaps (XBM) and uncompressed BMP
```

```
 */

AVT_API int avt_avatar_image_default (void);
AVT_API int avt_avatar_image_none (void);
AVT_API int avt_avatar_image_xpm (char **);
AVT_API int avt_avatar_image_xbm (const unsigned char *bits,
                                  int width, int height, int color);
AVT_API int avt_avatar_image_data (void *img, size_t imgsize);
AVT_API int avt_avatar_image_file (const char *file);
AVT_API int avt_avatar_image_stream (avt_stream *stream);


/*************************************************************************/
/* actions without or outside the balloon */
/* see also "showing images without the avatar" */

/* show an empty screen with the background color */
AVT_API void avt_clear_screen (void);

/* show just the avatar without the balloon */
AVT_API void avt_show_avatar (void);

/* like avt_show_avatar, but the avatar is moved in */
AVT_API int avt_move_in (void);

/* move the avatar out => empty screen */
AVT_API int avt_move_out (void);

/*
 * make a short sound, when audio is initialized
 * else it is the same as avt_flash
 * same as with \a in avt_say
 * the sound is actually not a bell ;-)
 */
AVT_API void avt_bell (void);

/* visual flash of the screen */
AVT_API void avt_flash (void);

/*
 * update, ie. handle events
 * use this in a longer loop in your program
 */
AVT_API int avt_update (void);

/* wait a while */
AVT_API int avt_wait (size_t milliseconds);

/* counter, which is increased every millisecond */
AVT_API size_t avt_ticks (void);

/* returnes elapsed time since start_ticks in milliseconds */
#define avt_elapsed(start_ticks)  (avt_ticks()-(start_ticks))

/*************************************************************************/
/* say or ask stuff */

/*
 * The encoding for wchar_t can be UTF-32 or UTF-16, native endian.
 * The encoding for char must have been set with
 * avt_char_encoding() (see below)
 */

/*
 * prints a zero terminated string in the balloon
 * if there is no balloon, it is drawn
 * if there is no avatar, it is shown (not moved in)
 * interprets control chars including overstrike-text
```

```
 */
AVT_API int avt_say (const wchar_t *txt);
AVT_API int avt_say_char (const char *txt);

/*
 * writes string with given length in the balloon
 * the string needn't be terminated and can contain binary zeros
 * if there is no balloon, it is drawn
 * if there is no avatar, it is shown (not moved in)
 * interprets control characters including overstrike-text
 */
AVT_API int avt_say_len (const wchar_t *txt, size_t len);
AVT_API int avt_say_char_len (const char *txt, size_t len);

/*
 * sets the balloon size so that the text fits exactly
 * prints a zero terminated string in the balloon
 * if there is no balloon, it is drawn
 * if there is no avatar, it is shown (not moved in)
 * interprets control characters including overstrike-text
 */
AVT_API int avt_tell (const wchar_t *txt);
AVT_API int avt_tell_char (const char *txt);

/*
 * sets the balloon size so that the text fits exactly
 * writes string with given length in the balloon
 * the string needn't be terminated and can contain binary zeros
 * if len is 0 then it's the same as avt_tell()
 * if there is no balloon, it is drawn
 * if there is no avatar, it is shown (not moved in)
 * interprets control characters including overstrike-text
 */
AVT_API int avt_tell_len (const wchar_t *txt, size_t len);
AVT_API int avt_tell_char_len (const char *txt, size_t len);

/*
 * get string (just one line)
 * the maximum length is AVT_LINELENGTH+1
 * size is the size of s in bytes (not the length)
 */
AVT_API int avt_ask (wchar_t *s, size_t size);
AVT_API int avt_ask_char (char *s, size_t size);


/*************************************************************************/
/* character encodings */

/*
 * To set the encoding to UTF-8, just use this:
 *   avt_char_encoding (avt_utf8 ());
 *
 * Additional encodings are available in avtaddons...
 */

struct avt_charenc
{
  void *data;
  size_t (*decode) (const struct avt_charenc *self, avt_char *, const char *);
  size_t (*encode) (const struct avt_charenc *self, char *, size_t, avt_char);
};

/*
 * set a new character encoding
 * returns the old one
 * use NULL to just query the old one
 */
AVT_API const struct avt_charenc *
```

```
avt_char_encoding (const struct avt_charenc *);

/* character encoding for UTF-8 */
AVT_API const struct avt_charenc *avt_utf8 (void);

/* character encoding for ISO-8859-1 (ISO Latin-1) */
AVT_API const struct avt_charenc *avt_iso8859_1 (void);

/* character encoding for plain old ASCII */
AVT_API const struct avt_charenc *avt_ascii (void);

/*
 * try to recode from fromcode to tocode
 * copies as much as fits
 * result is terminated
 */
AVT_API size_t avt_recode_char (const struct avt_charenc *tocode,
                                char *dest, size_t dest_size,
                                const struct avt_charenc *fromcode,
                                const char *src, size_t src_size);

/*
 * checks for UTF-8
 *
 * size can be smaller than the actual string length
 * it is okay if size ends in the middle of a sequence
 *
 * returns false if invalid UTF-8 sequence is detected
 * returns true for valid UTF-8 or plain ASCII
 * doesn't check for overlong sequences or all invalid characters
 */
AVT_API bool avt_detect_utf8 (const char *str, size_t size);


/*************************************************************************/
/* handling of single characters */

/*
 * writes a single character in the balloon
 * if there is no balloon, it is drawn
 * if there is no avatar, it is shown (not moved in)
 * interprets control characters, but not for overstrike-text
 * UTF-16 surrogate characters are interpreted too
 *
 * You can use this directly for characters from
 * US-ASCII, ISO-8859-1, UCS-2, UTF-16 or UTF-32 (=UCS-4)
 * except that UTF-32/UCS-4 don't allow UTF-16 surrogate characters
 */
AVT_API int avt_put_char (avt_char);

/*
 * checks whether the given character is printable
 * returns false on unknown or control characters
 */
AVT_API bool avt_is_printable (avt_char);

/*
 * checks whether the given character is a combining character
 * combining characters are added to the previous character
 * they don't take space
 *
 * not all combining characters of Unicode supported
 */
AVT_API bool avt_combining (avt_char);


/*************************************************************************/
/* key or event handling */
```

```c
/*
 * get a character from the keyboard
 * waits until a key is pressed
 * see AVT_KEY_ constants for function keys
 * on error or a quit request it returns AVT_KEY_NONE
 */
AVT_API avt_char avt_get_key (void);

/*
 * check if a key was pressed (or an event happened)
 * the key should then be fetched with avt_get_key
 */
AVT_API bool avt_key_pressed (void);

/* clear key buffer */
AVT_API void avt_clear_keys (void);

/* push key or event */
AVT_API void avt_push_key (avt_char);


/*****************************************************************************/
/* informational stuff */

/* is it initialized? */
AVT_API bool avt_initialized (void);

/* 0 = normal; 1 = quit-request; -1 = error */
AVT_API int avt_get_status (void);

/* set status */
AVT_API void avt_set_status (int);

/* get error message */
AVT_API char *avt_get_error (void);

/* set error message */
AVT_API void avt_set_error (const char *message);

/* which version of the linked library is used? */
AVT_API const char *avt_version (void);
AVT_API const wchar_t *avt_wide_version (void);

/* get copyright information */
AVT_API const char *avt_copyright (void);
AVT_API const wchar_t *avt_wide_copyright (void);

/* get license information */
AVT_API const char *avt_license (void);
AVT_API const wchar_t *avt_wide_license (void);


/*****************************************************************************/
/* colors */

/*
 * colors can always be used as 6-digit hexadecimal number,
 * for example 0xFFFFFF for white, 0x000000 for black
 * first 2 digits are for red, then 2 digits for green, then 2 for blue
 */

/*
 * returns a color number for the given values for red, green and blue
 * with these base colors you can mix any deliberate color
 * the values must be in the range of 0-255 (0x00-0xFF)
 */
#define avt_rgb(red, green, blue) \
```

```
   ((((red)&0xFF)<<16) | (((green)&0xFF)<<8) | ((blue)&0xFF))

/* strip the red, green or blue parts from a color number */
#define avt_red(color)    (((color) >> 16) & 0xFF)
#define avt_green(color)  (((color) >> 8) & 0xFF)
#define avt_blue(color)   ((color) & 0xFF)

/*
 * get color number from a given name
 * returns -1 on error
 */
AVT_API int avt_colorname (const char *name);

/*
 * reads a palette of predefined colors
 * returns the color name or NULL on error
 * if color is not NULL it gets the color number
 */
AVT_API const char *avt_palette (int entry, int *color);

/* get the size of the palette (number of entries) */
AVT_API size_t avt_palette_size (void);

/*
 * define the background color
 * can and should be called before avt_start
 * if the balloon is visible, it is cleared
 */
AVT_API void avt_set_background_color (int color);
AVT_API int avt_get_background_color (void);

/*
 * define the balloon color
 * can be called before avt_start
 * the text-background-color is set to the balloon-color too
 * if the balloon is visible, it is cleared
 */
AVT_API void avt_set_balloon_color (int color);
AVT_API int avt_get_balloon_color (void);

/* change the text color */
AVT_API void avt_set_text_color (int color);
AVT_API void avt_set_text_background_color (int color);

/* set text background to balloon color */
AVT_API void avt_set_text_background_ballooncolor (void);

/* set a color for loading monochrome bitmaps */
/* the background is always transparent */
AVT_API void avt_set_bitmap_color (int color);

AVT_API int avt_darker (int color, int amount);

AVT_API int avt_brighter (int color, int amount);

/* returns the maximum color value (red, green or blue) */
AVT_API int avt_brightness (int color);

/*****************************************************************/
/* settings */

/*
 * change the title and/or the shortname
 * use NULL for the unchanged part
 * if possible stick to ASCII for compatibility
 * SDL accepts UTF-8
 * backends may ignore this information
 */
```

```
AVT_API void avt_set_title (const char *title, const char *shortname);

/*
 * set name for the avatar
 * set to NULL to clear the name
 */
AVT_API int avt_set_avatar_name (const wchar_t *name);
AVT_API int avt_set_avatar_name_char (const char *name);
AVT_API int avt_set_avatar_name_mb (const char *name);

/* switch to fullscreen or window mode (if available) */
AVT_API void avt_switch_mode (int mode);

/* toggle fullscreen mode (if available) */
AVT_API void avt_toggle_fullscreen (void);

/* get screen mode */
AVT_API int avt_get_mode (void);

/*
 * set the baloon width and height in number of characters
 * 0 or less for maximum width
 * if it's actually changed, the balloon is redrawn and emptied
 * see also avt_get_max_x () and avt_get_max_y ()
 */
AVT_API void avt_set_balloon_size (int height, int width);
AVT_API void avt_set_balloon_width (int width);
AVT_API void avt_set_balloon_height (int height);

/* values for avt_set_avatar_mode */
#define AVT_SAY 0
#define AVT_THINK 1
#define AVT_HEADER 2
#define AVT_FOOTER 3

/* set the avatar mode */
AVT_API void avt_set_avatar_mode (int mode);

/* activate the text cursor? (default: no) */
AVT_API void avt_activate_cursor (bool);

/*
 * set text direction
 * the cursor is moved to start of the line
 * in a text, you might want to call avt_newline after that
 */
AVT_API void avt_text_direction (int direction);

/*
 * delay time for text-writing
 * may be AVT_DEFAULT_TEXT_DELAY
 * or 0 to switch it off
 */
AVT_API void avt_set_text_delay (int delay);

/*
 * delay time for page flipping
 * default: AVT_DEFAULT_FLIP_PAGE_DELAY
 */
AVT_API void avt_set_flip_page_delay (int delay);

/* set underlined mode on or off */
AVT_API void avt_underlined (bool);

/* get underlined mode */
AVT_API bool avt_get_underlined (void);

/* set bold mode on or off (not recommended) */
```

```
AVT_API void avt_bold (bool);

/* get bold mode */
AVT_API bool avt_get_bold (void);

/* set inverse mode on or off */
AVT_API void avt_inverse (bool);

/* get inverse mode */
AVT_API bool avt_get_inverse (void);

/* set default color and switch off bold, underlined, inverse */
/* also switches the markup mode off */
AVT_API void avt_normal_text (void);

/*
 * switch markup mode on or off
 *
 * in markup mode the character "_" toggles the underlined mode
 * and the character "*" toggles the bold mode on or off
 */
AVT_API void avt_markup (bool);

/*
 * set scroll mode
 * -1 = off, 0 = page-flipping, 1 = normal
 */
AVT_API void avt_set_scroll_mode (int mode);
AVT_API int avt_get_scroll_mode (void);

/* set newline mode (default: on) */
AVT_API void avt_newline_mode (bool mode);

/* get newline mode */
AVT_API bool avt_get_newline_mode (void);

/* set auto-margin mode (default: on) */
AVT_API void avt_set_auto_margin (bool mode);
AVT_API void avt_auto_margin (bool mode);

/* get auto-margin mode */
AVT_API bool avt_get_auto_margin (void);

/*
 * origin mode
 * false: origin (1,1) is always the top of the textarea
 * true:  origin (1,1) is the top of the viewport
 */
AVT_API void avt_set_origin_mode (bool mode);
AVT_API bool avt_get_origin_mode (void);

/* with this you can switch the mouse pointer on or off */
AVT_API void avt_set_mouse_visible (bool);


/*****************************************************************/
/* handle coordinates inside the balloon */

/*
 * the coordinates start with 1, 1 in the upper left corner
 * and are independent from the text direction
 */

/* get position in the viewport */
AVT_API int avt_where_x (void);
AVT_API int avt_where_y (void);

/*
```

```
 * is the cursor in the home position?
 * (also works for right-to-left writing)
 */
AVT_API bool avt_home_position (void);

/* maximum positions (whole text-field) */
AVT_API int avt_get_max_x (void);
AVT_API int avt_get_max_y (void);

/* put cusor to specified coordinates */
AVT_API void avt_move_x (int x);
AVT_API void avt_move_y (int y);
AVT_API void avt_move_xy (int x, int y);

/* save and restore current cursor position */
AVT_API void avt_save_position (void);
AVT_API void avt_restore_position (void);

/* set a viewport (sub-area of the textarea) */
AVT_API void avt_viewport (int x, int y, int width, int height);


/*************************************************************************/
/* activities inside the balloon */

/* new line - same as \n in avt_say */
AVT_API int avt_new_line (void);

/* wait a while and then clear the textfield - same as \f in avt_say */
AVT_API int avt_flip_page (void);

/*
 * clears the viewport
 * if there is no balloon yet, it is drawn
 */
AVT_API void avt_clear (void);

/*
 * clears from cursor position down the viewport
 * if there is no balloon yet, it is drawn
 */
AVT_API void avt_clear_down (void);

/*
 * clears from cursor position up the viewport
 * if there is no balloon yet, it is drawn
 */
AVT_API void avt_clear_up (void);

/*
 * clear end of line
 * depending on text direction
 */
AVT_API void avt_clear_eol (void);

/*
 * clear beginning of line
 * depending on text direction
 */
AVT_API void avt_clear_bol (void);

/* clear line */
AVT_API void avt_clear_line (void);

/*
 * forward one character position
 * ie. print a space
 */
```

```
AVT_API int avt_forward (void);

/* delete last caracter */
AVT_API void avt_backspace (void);

/* insert spaces at current position (move rest of line) */
AVT_API void avt_insert_spaces (int num);

/* delete num characters at current position (move rest of line) */
AVT_API void avt_delete_characters (int num);

/*
 * erase num characters from current position
 * don't move the cursor or the rest of the line
 */
AVT_API void avt_erase_characters (int num);

/* go to next or last tab stop */
AVT_API void avt_next_tab (void);
AVT_API void avt_last_tab (void);

/* reset tab stops to every eigth column */
AVT_API void avt_reset_tab_stops (void);

/* clear all tab stops */
AVT_API void avt_clear_tab_stops (void);

/* set or clear a tab in position x */
AVT_API void avt_set_tab (int x, bool);

/*
 * delete num lines, starting from line
 * the rest ist scrolled up
 */
AVT_API void avt_delete_lines (int line, int num);

/*
 * insert num lines, starting at line
 * the rest ist scrolled down
 */
AVT_API void avt_insert_lines (int line, int num);

/*
 * lock or unlock updates of the balloon-content
 * can be used for speedups
 * when set to false, the textarea gets updated
 * when set to true, the text_delay is set to 0
 * use with care!
 */
AVT_API void avt_lock_updates (bool);


/*****************************************************************/
/* showing images without the avatar */
/* you should call avt_wait or avt_wait_button or avt_get_key thereafter */

/* On error these functions return AVT_FAILURE without changing the status */

/*
 * supported formats:
 * X-Pixmaps (XPM), X Bitmaps (XBM) and uncompressed BMP
 */

AVT_API int avt_image_max_width (void);
AVT_API int avt_image_max_height (void);

AVT_API int avt_show_image_file (const char *file);
AVT_API int avt_show_image_stream (avt_stream *stream);
```

```
AVT_API int avt_show_image_data (void *img, size_t imgsize);
AVT_API int avt_show_image_xpm (char **xpm);
AVT_API int avt_show_image_xbm (const unsigned char *bits,
                                int width, int height, int color);


/*****************************************************************/
/* high-level functions */

/* wait for a keypress while displaying a button */
AVT_API int avt_wait_button (void);

/*
 * show positive or negative buttons
 * keys for positive: + 1 Enter
 * keys for negative: - 0 Backspace
 *
 * returns the result as boolean
 * on error or quit request false is returned and the status is set
 * you should check the status with avt_get_status()
 */
AVT_API bool avt_decide (void);

/*
 * navigation bar
 *
 * buttons is a string with the following characters
 * 'l': left
 * 'r': right (play)
 * 'd': down
 * 'u': up
 * 'x': cancel
 * 'f': (fast)forward
 * 'b': (fast)backward
 * 'p': pause
 * 's': stop
 * 'e': eject
 * '*': circle (record)
 * '+': plus (add)
 * '-': minus (remove)
 * '?': help
 * ' ': spacer (no button)
 *
 * Pressing a key with one of those characters selects it.
 * For the directions you can also use the arrow keys,
 * The [Help] key or [F1] return '?'.
 *
 * the function returns the letter for the selected option
 * or AVT_ERROR or AVT_QUIT
 *
 * If audio output ends while this function is active, it automatically
 * pushes either 's' (stop) or 'f' (forward).  If both are given, then
 * the rightmost.
 *
 * example:
 *   r = avt_navigate ("lxr");
 *   if (r < 32) exit (0);
 *   switch (r) ...
 */
AVT_API int avt_navigate (const char *buttons);

/*
 * avt_choice - use for short menus
 * result:        result item, first item is 1
 * start_line:    line, where choice begins
 * items:         number of items/lines
 * key:           first key, like '1' or 'a', 0 for no keys
 * back, forward: whether first/last entry is a back/forward function
```

```
 *
 * returns AVT_FAILURE or AVT_ERROR on error
 */
AVT_API int
avt_choice (int *result, int start_line, int items, avt_char key,
            bool back, bool forward);

/*
 * avt_menu - use for long menus
 * result:        result item, first item is 1
 * items:         number of items/lines
 * show:          function for showing an entry with given nr in one line
 * data:          data passed to the function show (may be NULL)
 *
 * returns AVT_FAILURE or AVT_ERROR on error
 */
AVT_API int
avt_menu (int *result, int items,
          void (*show) (int nr, void *data),
          void *data);

/*
 * show longer text with a text-viewer application
 * if len is 0, assume 0-terminated string
 * startline is only used, when it is greater than 1
 */
AVT_API int avt_pager (const wchar_t *txt, size_t len, int startline);
AVT_API int avt_pager_char (const char *txt, size_t len, int startline);

AVT_API int avt_pager_mb (const char *txt, size_t len, int startline);

/* show final credits */
AVT_API int avt_credits (const wchar_t *text, bool centered);
AVT_API int avt_credits_char (const char *txt, bool centered);
AVT_API int avt_credits_mb (const char *text, bool centered);


/*****************************************************************************/
/* plumbing */

/*
 * set a function for the bell
 * this is for avt_bell(), or "\a", also used internally
 *
 * set to avt_flash to get a visual bell
 * set to NULL to disable the bell
 */
AVT_API void avt_bell_function (void (*func) (void));

/* reserve single keys (Esc, F11) */
AVT_API void avt_reserve_single_keys (bool);

/*
 * returns pointer to character definition of given codepoint
 * either defined as unsigned char or unsigned short,
 * depending on the fonts width
 */
AVT_API void *avt_get_font_char (int);

/*
 * get height, width and baseline of a character
 * the font is a fixed width font
 */
AVT_API void avt_get_font_dimensions (int *width, int *height,
                                      int *baseline);


/* free memory allocated by this library */
```

```
AVT_API void avt_free (void *);

/*************************************************************************/
/* audio output */

/* Note: The interface for audio output may change in future versions */

/* for playmode parameters */
#define AVT_LOAD 0
#define AVT_PLAY 1
#define AVT_LOOP 2

/* must be called AFTER avt_start! */
AVT_API int avt_start_audio (void);

/*
 * quit audio system
 * (automatically called by avt_quit())
 */
AVT_API void avt_quit_audio (void);

/*
 * supported audio formats:
 * AU or Wave (*.au, *.snd, *.wav)
 * linear PCM with up to 32Bit, mu-law, A-law
 * mono or stereo
 *
 * the current implementation can only play sounds with
 * up to 16Bit precision, but files with more bits can
 * be imported.
 */

/*
 * loads an audio file in AU or Wave format
 * not for headerless formats
 */
AVT_API avt_audio *avt_load_audio_file (const char *filename, int playmode);

/*
 * loads audio in AU or Wave format from a stream
 * if maxsize is >0 it reads no more than this size
 * the stream will not be closed
 * not for headerless formats
 */
AVT_API avt_audio *avt_load_audio_part (avt_stream *stream, size_t maxsize, int playmode);

/*
 * loads audio in AU or Wave format from a stream
 * not for headerless formats
 * the stream will not be closed
 */
AVT_API avt_audio *avt_load_audio_stream (avt_stream *stream, int playmode);

/*
 * loads audio in AU or Wave format from memory
 * must still be freed with avt_free_audio!
 */
AVT_API avt_audio *avt_load_audio_data (const void *data, size_t datasize, int playmode);

/* values for audio_type */
#define AVT_AUDIO_UNKNOWN   0  /* doesn't play */
#define AVT_AUDIO_U8        1  /* unsigned 8 Bit */
#define AVT_AUDIO_S8        2  /* signed 8 Bit */
#define AVT_AUDIO_S16LE     3  /* signed 16 Bit little endian */
#define AVT_AUDIO_S16BE     4  /* signed 16 Bit big endian */
#define AVT_AUDIO_S16SYS    5  /* signed 16 Bit system's endianess */
#define AVT_AUDIO_S24LE     6  /* signed 24 Bit little endian */
#define AVT_AUDIO_S24BE     7  /* signed 24 Bit big endian */
```

```
#define AVT_AUDIO_S24SYS    8  /* signed 24 Bit system's endianess */
#define AVT_AUDIO_S32LE     9  /* signed 32 Bit little endian */
#define AVT_AUDIO_S32BE    10  /* signed 32 Bit big endian */
#define AVT_AUDIO_S32SYS   11  /* signed 32 Bit system's endianess */
#define AVT_AUDIO_MULAW   100  /* 8 Bit mu-law (u-law) */
#define AVT_AUDIO_ALAW    101  /* 8 Bit A-Law */

/* for channels */
#define AVT_AUDIO_MONO      1
#define AVT_AUDIO_STEREO    2

/*
 * prepares raw audio
 * you can use avt_add_raw_audio_data to add data
 * and avt_finalize_raw_audio to finalize it
 *
 * capacity is the amount of bytes you want to add
 * capacity may be 0 if unknown, but you should avoid that
 * audio_type is one of the AVT_AUDIO_* constants
 * channels is AVT_MONO or AVT_STEREO
 *
 * must be freed with avt_free_audio! (even if empty)
 */
AVT_API avt_audio *avt_prepare_raw_audio (size_t capacity,
                      int samplingrate, int audio_type, int channels);

/*
 * add raw audio data to an audio type
 * the audio type must have been created with avt_prepare_raw_audio
 * data should be a larger buffer
 */
AVT_API int avt_add_raw_audio_data (avt_audio *snd, void *data, size_t data_size);

/* eventually do some cleanups after several avt_add_raw_audio_data */
AVT_API void avt_finalize_raw_audio (avt_audio *snd);

/* frees memory of a loaded sound */
AVT_API void avt_free_audio (avt_audio *snd);

/*
 * plays a sound
 * playmode is one of AVT_PLAY or AVT_LOOP
 * on error it returns AVT_ERROR without changing the status
 */
AVT_API int avt_play_audio (avt_audio *snd, int playmode);

/*
 * wait until the sound ends
 * this stops a loop, but still plays to the end of the sound
 */
AVT_API int avt_wait_audio_end (void);

/* stops audio immediately */
AVT_API void avt_stop_audio (void);

/* pause/resume audio */
AVT_API void avt_pause_audio (bool);

/*
 * Is this sound currently playing?
 * Use NULL to check for any sound
 */
AVT_API bool avt_audio_playing (avt_audio *snd);

/*
 * send a key when audio ends
 * set to AVT_KEY_NONE to stop it
 * it returns the previous key
```

```
 */
AVT_API avt_char avt_set_audio_end_key (avt_char key);


/*****************************************************************/
/* experimental */
/* these functions will likely change or being removed in later versions */
/* so use them with care, don't rely on them */

/*
 * send a key when the window is resized
 * set to AVT_KEY_NONE to stop it
 * it returns the previous key
 */
AVT_API avt_char avt_set_resize_key (avt_char key);

/*
 * send a key when one of the buttons of the pointer (mouse) is pressed
 * set to AVT_KEY_NONE to stop it
 * it returns the previous key
 */
AVT_API avt_char avt_set_pointer_buttons_key (avt_char key);

/*
 * send a key when the pointer (mouse) is moved
 * set to AVT_KEY_NONE to stop it
 * it returns the previous key
 */
AVT_API avt_char avt_set_pointer_motion_key (avt_char key);

/*
 * get the current pointer position (absolute position)
 */
AVT_API void avt_get_pointer_position (int *x, int *y);

/*
 * get a string with a default text
 *
 * position can be -1 for at the end or 0 for at the beginning,
 * 1 for behind first character and so on
 *
 * if mode is 0 input is ended with Enter
 * if mode is 1 input is also ended by up and down arrow keys
 *
 * returns the key which ended input, or AVT_KEY_NONE on quit request
 */
AVT_API avt_char avt_input (wchar_t *result, size_t size,
                            const wchar_t *default_text,
                            int position, int mode);

AVT_API avt_char avt_input_char (char *result, size_t size,
                                 const char *default_text,
                                 int position, int mode);

AVT_API avt_char avt_input_mb (char *s, size_t size,
                               const char *default_text,
                               int position, int mode);

/*
 * show raw image
 * only 4 Bytes per pixel supported (0RGB)
 */
AVT_API int avt_show_raw_image (void *image_data, int width, int height);

/*
 * put an image onto a raw image
 * only 4 Bytes per pixel supported (0RGB)
 */
```

```
AVT_API int avt_put_raw_image_file (const char *file, int x, int y,
                                    void *image_data, int width, int height);

AVT_API int avt_put_raw_image_stream (avt_stream * stream, int x, int y,
                                      void *image_data, int width, int height);

AVT_API int avt_put_raw_image_data (void *img, size_t imgsize, int x, int y,
                                    void *image_data, int width, int height);

AVT_API int avt_put_raw_image_xpm (char **xpm, int x, int y,
                                   void *image_data, int width, int height);


/*****************************************************************************/
/* deprecated functions - only for backward comatibility */
/* don't use them for new programs! */

#ifndef DISABLE_DEPRECATED

/* macro for marking deprecated functions in this header */
#if (defined(__GNUC__) || defined(__clang__)) && !defined(_AVT_USE_DEPRECATED)
#  define AVT_DEPRECATED  __attribute__ ((__deprecated__))
#else
#  define AVT_DEPRECATED
#endif /* __GNUC__ */

AVT_API int avt_key (avt_char *ch) AVT_DEPRECATED;

#endif /* DISABLE_DEPRECATED */

#ifdef __cplusplus
}
#endif

#endif /* AKFAVATAR_H */
```