NAME

dpv - dialog progress view library

LIBRARY

library "libdpv"

SYNOPSIS

#include <dpv.h>

int
dpv(*struct dpv_config *config, struct dpv_file_node *file_list*);

void
dpv_free(*void*);

DESCRIPTION

The **dpv** library provides an interface for creating complex "gauge" widgets for displaying progress on various actions. The **dpv** library can display progress with one of dialog(3), dialog(1), or Xdialog(1) (*ports/x11/xdialog*).

The **dpv**() *config* argument properties for configuring global display features:

```
struct dpv_config {
    uint8_t         keep_tite;     /* Cleaner exit for scripts */
    enum dpv_display display_type;  /* Def. DPV_DISPLAY_LIBDIALOG */
    enum dpv_output  output_type;   /* Default DPV_OUTPUT_NONE */
    int             debug;         /* Enable debug on stderr */
    int             display_limit; /* Files/page. Default -1 */
    int             label_size;    /* Label size. Default 28 */
    int             pbar_size;     /* Mini-progress size */
    int             dialog_updates_per_second; /* Default 16 */
    int             status_updates_per_second; /* Default 2 */
    uint16_t        options;       /* Default 0 (none) */
    char            *title;        /* Widget title */
    char            *backtitle;    /* Widget backtitle */
    char            *aprompt;      /* Append. Default NULL */
    char            *pprompt;      /* Prefix. Default NULL */
    char            *msg_done;     /* Default ‘Done’ */
    char            *msg_fail;     /* Default ‘Fail’ */
    char            *msg_pending;  /* Default ‘Pending’ */
```

```
    char            *output;       /* Output format string */
    const char      *status_solo;  /* dialog(3) solo-status format.
                                      * Default DPV_STATUS_SOLO */
    const char      *status_many;  /* dialog(3) many-status format.
                                      * Default DPV_STATUS_MANY */

    /*
     * Function pointer; action to perform data transfer
     */
    int (*action)(struct dpv_file_node *file, int out);
};

enum dpv_display {
    DPV_DISPLAY_LIBDIALOG = 0, /* Use dialog(3) (default) */
    DPV_DISPLAY_STDOUT,        /* Use stdout */
    DPV_DISPLAY_DIALOG,        /* Use spawned dialog(1) */
    DPV_DISPLAY_XDIALOG,       /* Use spawned Xdialog(1) */
};

enum dpv_output {
    DPV_OUTPUT_NONE = 0, /* No output (default) */
    DPV_OUTPUT_FILE,     /* Read 'output' member as file path */
    DPV_OUTPUT_SHELL,    /* Read 'output' member as shell cmd */
};
```

The *options* member of the **dpv**() *config* argument is a mask of bit fields indicating various processing options. Possible flags are:

DPV_TEST_MODE — Enable test mode. In test mode, the **action**() callback of the *config* argument is not called but instead simulated-data is used to drive progress. Appends "[TEST MODE]" to the status line (to override, set the *status_format* member of the **dpv**() *config* argument; for example, to DPV_STATUS_DEFAULT).

DPV_WIDE_MODE — Enable wide mode. In wide mode, the length of the *aprompt* and *pprompt* members of the **dpv**() *config* argument will bump the width of the gauge widget. Prompts wider than the maximum width will wrap (unless using Xdialog(1) (*ports/x11/xdialog*); see BUGS section below).

DPV_NO_LABELS — Disables the display of labels associated with each transfer (*label_size* member of **dpv**() *config* argument is ignored).

DPV_USE_COLOR Force the use of color even if the *display_type* does not support color (USE_COLOR environment variable is ignored).

DPV_NO_OVERRUN When enabled, callbacks for the current *dpv_file_node* are terminated when **action**() returns 100 or greater (alleviates the need to change the *status* of the current *dpv_file_node* but may also cause file truncation if the stream exceeds expected length).

The *file_list* argument to **dpv**() is a pointer to a "linked-list", described in <*dpv.h*>:

```
struct dpv_file_node {
   enum dpv_status    status; /* status of read operation */
   char               *msg;   /* display instead of "Done/Fail" */
   char               *name;  /* name of file to read */
   char               *path;  /* path to file */
   long long          length; /* expected size */
   long long          read;   /* number units read (e.g., bytes) */
   struct dpv_file_node *next;/* pointer to next (end with NULL) */
};
```

For each of the items in the *file_list* "linked-list" argument, the **action**() callback member of the **dpv**() *config* argument is called. The **action**() function performs a "nominal" action on the file and return. The return value of *int* represents the current progress percentage (0-100) for the current file.

The **action**() callback provides two variables for each call. *file* provides a reference to the current *dpv_file_node* being processed. *out* provides a file descriptor where the data goes.

If the *output* member of the **dpv**() *config* argument was set to DPV_OUTPUT_NONE (default; when invoking **dpv**()), the *out* file descriptor of **action**() will be zero and can be ignored. If *output* was set to DPV_OUTPUT_FILE, *out* will be an open file descriptor to a file. If *output* was set to DPV_OUTPUT_SHELL, *out* will be an open file descriptor to a pipe for a spawned shell program. When *out* is greater than zero, write data that has been read back to *out*.

To abort **dpv**(), either from the **action**() callback or asynchronously from a signal handler, two globals are provided via <*dpv.h*>:

```
extern int dpv_interrupt; /* Set to TRUE in interrupt handler */
extern int dpv_abort;     /* Set to true in callback to abort */
```

These globals are not automatically reset and must be manually maintained. Do not forget to reset these

globals before subsequent invocations of **dpv**() when making multiple calls from the same program.

In addition, the *status* member of the **action**() *file* argument can be used to control callbacks for the current file. The *status* member can be set to any of the below from <*dpv.h*>:

```
enum dpv_status {
        DPV_STATUS_RUNNING = 0, /* Running (default) */
        DPV_STATUS_DONE,        /* Completed */
        DPV_STATUS_FAILED,      /* Oops, something went wrong */
};
```

The default *status* is zero, DPV_STATUS_RUNNING, which keeps the callbacks coming for the current **file**(). Setting 'file->status' to anything other than DPV_STATUS_RUNNING will cause **dpv**() to loop to the next file, effecting the next callback, if any.

The **action**() callback is responsible for calculating percentages and (recommended) maintaining a **dpv** global counter so **dpv**() can display throughput statistics. Percentages are reported through the *int* return value of the **action**() callback. Throughput statistics are calculated from the below global *int* in <*dpv.h*>:

```
extern int dpv_overall_read;
```

Set this to the number of bytes that have been read for all files. Throughput information is displayed in the status line (only available when using dialog(3)) at the bottom of the screen. See DPV_DISPLAY_LIBDIALOG above.

Note that *dpv_overall_read* does not have to represent bytes. For example, the *status_format* can be changed to display something other than "bytes" and increment *dpv_overall_read* accordingly (for example, counting lines).

When **dpv**() is processing the current file, the *length* and *read* members of the **action**() *file* argument are used for calculating the display of mini progress bars (if enabled; see *pbar_size* above). If the *length* member of the current *file* is less than zero (indicating an unknown file length), a humanize_number(3) version of the *read* member is used instead of a traditional progress bar. Otherwise a progress bar is calculated as percentage read to file length. **action**() callback must maintain these member values for mini-progress bars.

The **dpv_free**() function performs free(3) on private global variables initialized by **dpv**().

ENVIRONMENT

The below environment variables are referenced by **dpv**:

DIALOG Override command string used to launch dialog(1) (requires DPV_DISPLAY_DIALOG) or Xdialog(1) (*ports/x11/xdialog*) (requires DPV_DISPLAY_XDIALOG); default is either ‘dialog’ (for DPV_DISPLAY_DIALOG) or ‘Xdialog’ (for DPV_DISPLAY_XDIALOG).

DIALOGRC If set and non-NULL, path to ‘.dialogrc’ file.

HOME If ‘$DIALOGRC’ is either not set or NULL, used as a prefix to ‘.dialogrc’ (that is, ‘$HOME/.dialogrc’).

USE_COLOR If set and NULL, disables the use of color when using dialog(1). Does not apply to Xdialog(1) (*ports/x11/xdialog*).

msg_done msg_fail msg_pending
Internationalization strings for overriding the default English strings ‘Done’, ‘Fail’, and ‘Pending’ respectively. To prevent their usage, explicitly set the *msg_done*, *msg_fail*, and *msg_pending* members of **dpv**() *config* argument to default macros (DPV_DONE_DEFAULT, DPV_FAIL_DEFAULT, and DPV_PENDING_DEFAULT) or desired values.

FILES

$HOME/.dialogrc

SEE ALSO

dialog(1), Xdialog(1) (*ports/x11/xdialog*), dialog(3)

HISTORY

The **dpv** library first appeared in FreeBSD 10.2.

AUTHORS

Devin Teske <dteske@FreeBSD.org>

BUGS

Xdialog(1) (*ports/x11/xdialog*), when given both ‘**--title** *title*’ (see above ‘*title*’ member of *struct dpv_config*) and ‘**--backtitle** *backtitle*’ (see above ‘*backtitle*’ member of *struct dpv_config*), displays the backtitle in place of the title and vice-versa.

Xdialog(1) (*ports/x11/xdialog*) does not wrap long prompt texts received after initial launch. This is a known issue with the ‘--gauge’ widget in Xdialog(1) (*ports/x11/xdialog*). Embed escaped newlines within prompt text to force line breaks.

dialog(1) does not display the first character after a series of escaped escape-sequences (for example, ‘‘\\n’’ produces ‘‘\’’ instead of ‘‘\n’’). This is a known issue with dialog(1) and does not affect dialog(3) or Xdialog(1) (*ports/x11/xdialog*).

If an application ignores USE_COLOR when set and NULL before calling **dpv**() with color escape sequences anyway, dialog(3) and dialog(1) may not render properly. Workaround is to detect when USE_COLOR is set and NULL and either not use color escape sequences at that time or use unsetenv(3) to unset USE_COLOR, forcing interpretation of color sequences. This does not effect Xdialog(1) (*ports/x11/xdialog*), which renders the color escape sequences as plain text. See "embedded "\Z" sequences" in dialog(1) for additional information.