NAME

UI, UI_new, UI_new_method, UI_free, UI_add_input_string, UI_dup_input_string, UI_add_verify_string, UI_dup_verify_string, UI_add_input_boolean, UI_dup_input_boolean, UI_add_info_string, UI_dup_info_string, UI_add_error_string, UI_dup_error_string, UI_construct_prompt, UI_add_user_data, UI_dup_user_data, UI_get0_user_data, UI_get0_result, UI_get_result_length, UI_process, UI_ctrl, UI_set_default_method, UI_get_default_method, UI_get_method, UI_set_method, UI_OpenSSL, UI_null - user interface

SYNOPSIS

```
#include <openssl/ui.h>

typedef struct ui_st UI;

UI *UI_new(void);
UI *UI_new_method(const UI_METHOD *method);
void UI_free(UI *ui);

int UI_add_input_string(UI *ui, const char *prompt, int flags,
                        char *result_buf, int minsize, int maxsize);
int UI_dup_input_string(UI *ui, const char *prompt, int flags,
                        char *result_buf, int minsize, int maxsize);
int UI_add_verify_string(UI *ui, const char *prompt, int flags,
                         char *result_buf, int minsize, int maxsize,
                         const char *test_buf);
int UI_dup_verify_string(UI *ui, const char *prompt, int flags,
                         char *result_buf, int minsize, int maxsize,
                         const char *test_buf);
int UI_add_input_boolean(UI *ui, const char *prompt, const char *action_desc,
                         const char *ok_chars, const char *cancel_chars,
                         int flags, char *result_buf);
int UI_dup_input_boolean(UI *ui, const char *prompt, const char *action_desc,
                         const char *ok_chars, const char *cancel_chars,
                         int flags, char *result_buf);
int UI_add_info_string(UI *ui, const char *text);
int UI_dup_info_string(UI *ui, const char *text);
int UI_add_error_string(UI *ui, const char *text);
int UI_dup_error_string(UI *ui, const char *text);

char *UI_construct_prompt(UI *ui_method,
                          const char *phrase_desc, const char *object_name);
```

```
void *UI_add_user_data(UI *ui, void *user_data);
int UI_dup_user_data(UI *ui, void *user_data);
void *UI_get0_user_data(UI *ui);

const char *UI_get0_result(UI *ui, int i);
int UI_get_result_length(UI *ui, int i);

int UI_process(UI *ui);

int UI_ctrl(UI *ui, int cmd, long i, void *p, void (*f)());

void UI_set_default_method(const UI_METHOD *meth);
const UI_METHOD *UI_get_default_method(void);
const UI_METHOD *UI_get_method(UI *ui);
const UI_METHOD *UI_set_method(UI *ui, const UI_METHOD *meth);

UI_METHOD *UI_OpenSSL(void);
const UI_METHOD *UI_null(void);
```

DESCRIPTION

UI stands for User Interface, and is general purpose set of routines to prompt the user for text-based information. Through user-written methods (see **UI_create_method**(3)), prompting can be done in any way imaginable, be it plain text prompting, through dialog boxes or from a cell phone.

All the functions work through a context of the type UI. This context contains all the information needed to prompt correctly as well as a reference to a UI_METHOD, which is an ordered vector of functions that carry out the actual prompting.

The first thing to do is to create a UI with **UI_new()** or **UI_new_method()**, then add information to it with the UI_add or UI_dup functions. Also, user-defined random data can be passed down to the underlying method through calls to **UI_add_user_data()** or **UI_dup_user_data()**. The default UI method doesn't care about these data, but other methods might. Finally, use **UI_process()** to actually perform the prompting and **UI_get0_result()** and **UI_get_result_length()** to find the result to the prompt and its length.

A UI can contain more than one prompt, which are performed in the given sequence. Each prompt gets an index number which is returned by the UI_add and UI_dup functions, and has to be used to get the corresponding result with **UI_get0_result()** and **UI_get_result_length()**.

UI_process() can be called more than once on the same UI, thereby allowing a UI to have a long

lifetime, but can just as well have a short lifetime.

The functions are as follows:

UI_new() creates a new UI using the default UI method. When done with this UI, it should be freed using **UI_free()**.

UI_new_method() creates a new UI using the given UI method. When done with this UI, it should be freed using **UI_free()**.

UI_OpenSSL() returns the built-in UI method (note: not necessarily the default one, since the default can be changed. See further on). This method is the most machine/OS dependent part of OpenSSL and normally generates the most problems when porting.

UI_null() returns a UI method that does nothing. Its use is to avoid getting internal defaults for passed UI_METHOD pointers.

UI_free() removes a UI from memory, along with all other pieces of memory that's connected to it, like duplicated input strings, results and others. If **ui** is NULL nothing is done.

UI_add_input_string() and **UI_add_verify_string()** add a prompt to the UI, as well as flags and a result buffer and the desired minimum and maximum sizes of the result, not counting the final NUL character. The given information is used to prompt for information, for example a password, and to verify a password (i.e. having the user enter it twice and check that the same string was entered twice). **UI_add_verify_string()** takes and extra argument that should be a pointer to the result buffer of the input string that it's supposed to verify, or verification will fail.

UI_add_input_boolean() adds a prompt to the UI that's supposed to be answered in a boolean way, with a single character for yes and a different character for no. A set of characters that can be used to cancel the prompt is given as well. The prompt itself is divided in two, one part being the descriptive text (given through the *prompt* argument) and one describing the possible answers (given through the *action_desc* argument).

UI_add_info_string() and **UI_add_error_string()** add strings that are shown at the same time as the prompt for extra information or to show an error string. The difference between the two is only conceptual. With the built-in method, there's no technical difference between them. Other methods may make a difference between them, however.

The flags currently supported are **UI_INPUT_FLAG_ECHO**, which is relevant for **UI_add_input_string()** and will have the users response be echoed (when prompting for a password,

this flag should obviously not be used, and **UI_INPUT_FLAG_DEFAULT_PWD**, which means that a default password of some sort will be used (completely depending on the application and the UI method).

UI_dup_input_string(), **UI_dup_verify_string()**, **UI_dup_input_boolean()**, **UI_dup_info_string()** and **UI_dup_error_string()** are basically the same as their UI_add counterparts, except that they make their own copies of all strings.

UI_construct_prompt() is a helper function that can be used to create a prompt from two pieces of information: a phrase description *phrase_desc* and an object name *object_name*, where the latter may be NULL. The default constructor (if there is none provided by the method used) creates a string "Enter *phrase_desc* for *object_name*:" where the " for *object_name*" part is left out if *object_name* is NULL. With the description "pass phrase" and the filename "foo.key", that becomes "Enter pass phrase for foo.key:". Other methods may create whatever string and may include encodings that will be processed by the other method functions.

UI_add_user_data() adds a user data pointer for the method to use at any time. The built-in UI method doesn't care about this info. Note that several calls to this function doesn't add data, it replaces the previous blob with the one given as argument.

UI_dup_user_data() duplicates the user data and works as an alternative to **UI_add_user_data()** when the user data needs to be preserved for a longer duration, perhaps even the lifetime of the application. The UI object takes ownership of this duplicate and will free it whenever it gets replaced or the UI is destroyed. **UI_dup_user_data()** returns 0 on success, or -1 on memory allocation failure or if the method doesn't have a duplicator function.

UI_get0_user_data() retrieves the data that has last been given to the UI with **UI_add_user_data()** or UI_dup_user_data.

UI_get0_result() returns a pointer to the result buffer associated with the information indexed by *i*.

UI_get_result_length() returns the length of the result buffer associated with the information indexed by *i*.

UI_process() goes through the information given so far, does all the printing and prompting and returns the final status, which is -2 on out-of-band events (Interrupt, Cancel, ...), -1 on error and 0 on success.

UI_ctrl() adds extra control for the application author. For now, it understands two commands: **UI_CTRL_PRINT_ERRORS**, which makes **UI_process()** print the OpenSSL error stack as part of processing the UI, and **UI_CTRL_IS_REDOABLE**, which returns a flag saying if the used UI can be

used again or not.

UI_set_default_method() changes the default UI method to the one given. This function is not thread-safe and should not be called at the same time as other OpenSSL functions.

UI_get_default_method() returns a pointer to the current default UI method.

UI_get_method() returns the UI method associated with a given UI.

UI_set_method() changes the UI method associated with a given UI.

NOTES

The resulting strings that the built in method **UI_OpenSSL()** generate are assumed to be encoded according to the current locale or (for Windows) code page. For applications having different demands, these strings need to be converted appropriately by the caller. For Windows, if the **OPENSSL_WIN32_UTF8** environment variable is set, the built-in method **UI_OpenSSL()** will produce UTF-8 encoded strings instead.

RETURN VALUES

UI_new() and **UI_new_method()** return a valid **UI** structure or NULL if an error occurred.

UI_add_input_string(), **UI_dup_input_string()**, **UI_add_verify_string()**, **UI_dup_verify_string()**, **UI_add_input_boolean()**, **UI_dup_input_boolean()**, **UI_add_info_string()**, **UI_dup_info_string()**, **UI_add_error_string()** and **UI_dup_error_string()** return a positive number on success or a value which is less than or equal to 0 otherwise.

UI_construct_prompt() returns a string or NULL if an error occurred.

UI_dup_user_data() returns 0 on success or -1 on error.

UI_get0_result() returns a string or NULL on error.

UI_get_result_length() returns a positive integer or 0 on success; otherwise it returns -1 on error.

UI_process() returns 0 on success or a negative value on error.

UI_ctrl() returns a mask on success or -1 on error.

UI_get_default_method(), **UI_get_method()**, **UI_OpenSSL()**, **UI_null()** and **UI_set_method()** return either a valid **UI_METHOD** structure or NULL respectively.

HISTORY

The **UI_dup_user_data()** function was added in OpenSSL 1.1.1.

COPYRIGHT