原文はこちらをご覧下さい。

# プロセス制御関数

The functions in this section are used to launch or terminate the other processes.

- ::wxExecute
- ::wxExit
- ::wxKill
- ::wxGetProcessId
- ::wxShell
- ::wxShutdown

## ::wxExecute

long wxExecute(const wxString& command, int sync = wxEXEC_ASYNC, wxProcess *callback = NULL)

wxPerl note: In wxPerl this function is called Wx::ExecuteCommand

long wxExecute(char **argv, int flags = wxEXEC_ASYNC, wxProcess *callback = NULL)

wxPerl note: In wxPerl this function is called Wx::ExecuteArgs

long wxExecute(const wxString& command, wxArrayString& output, int flags = 0)

wxPerl note: In wxPerl this function is called Wx::ExecuteStdout and it only takes the command argument, and returns a 2-element list ( status, output ), where output is an array reference.

long wxExecute(const wxString& command, wxArrayString& output, wxArrayString& errors, int flags = 0)

wxPerl note: In wxPerl this function is called Wx::ExecuteStdoutStderr and it only takes the command argument, and returns a 3-element list ( status, output, errors ), where output and errors are array references.

Executes another program in Unix or Windows.

The first form takes a command string, such as "emacs file.txt".

The second form takes an array of values: a command, any number of arguments, terminated by NULL.

The semantics of the third and fourth versions is different from the first two and is described in more details below.

If flags parameter contains wxEXEC_ASYNC flag (the default), flow of control immediately returns. If it contains wxEXEC_SYNC, the current application waits until the other program has terminated.

In the case of synchronous execution, the return value is the exit code of the process (which terminates by the moment the function returns) and will be -1 if the process couldn't be started and typically 0 if the process terminated successfully. Also, while waiting for the process to terminate, wxExecute will call wxYield. Because of this, by default this function disables all application windows to avoid unexpected reentrancies which could result from the users interaction with the program while the child process is running. If you are sure that it is safe to not disable the program windows, you may pass wxEXEC_NODISABLE flag to prevent this automatic disabling from happening.

For asynchronous execution, however, the return value is the process id and zero value indicates that the command could not be executed. As an added complication, the return value of -1 in this case indicates that we didn't launch a new process, but connected to the running one (this can only happen in case of using DDE under Windows for command execution). In particular, in this, and only this, case the calling code will not get the notification about process termination.

If callback isn't NULL and if execution is asynchronous, wxProcess::OnTerminate will be called when the process finishes. Specifying this parameter also allows you to redirect the standard input and/or output of the process being launched by calling Redirect. If the child process IO is redirected, under Windows the process window is not shown by default (this avoids having to flush an unnecessary console for the processes which don't create any windows anyhow) but a wxEXEC_NOHIDE flag can be used to prevent this from happening, i.e. with this flag the child process window will be shown normally.

Under Unix the flag wxEXEC_MAKE_GROUP_LEADER may be used to ensure that the new process is a group leader (this will create a new session if needed). Calling wxKill passing wxKILL_CHILDREN will kill this process as well as all of its children (except those which have started their own session).

Finally, you may use the third overloaded version of this function to execute a process (always synchronously, the contents of flags is or'd with wxEXEC_SYNC) and capture its output in the array output. The fourth version adds the possibility to additionally capture the messages from standard error output in the errors array.

NB: Currently wxExecute() can only be used from the main thread, calling this function from another thread will result in an assert failure in debug build and won't work.

wxShell, wxProcess, Exec sample.

command

The command to execute and any parameters to pass to it as a single string.

argv

The command to execute should be the first element of this array, any additional ones are the command parameters and the array must be terminated with a NULL pointer.

flags

Combination of bit masks wxEXEC_ASYNC, wxEXEC_SYNC and wxEXEC_NOHIDE

callback

An optional pointer to wxProcess

<wx/utils.h>

## ::wxExit

void wxExit()

Exits application after calling wxApp::OnExit. Should only be used in an emergency: normally the top-level frame should be deleted (after deleting all other frames) to terminate the application. See wxCloseEvent and wxApp.

<wx/app.h>

## ::wxKill

int wxKill(long pid, int sig = wxSIGTERM, wxKillError *rc = NULL, int flags = 0)

Equivalent to the Unix kill function: send the given signal sig to the process with PID pid. The valid signal values are

```
enum wxSignal
{
    wxSIGNONE = 0,  // verify if the process exists under Unix
    wxSIGHUP,
    wxSIGINT,
    wxSIGQUIT,
    wxSIGILL,
    wxSIGTRAP,
    wxSIGABRT,
    wxSIGEMT,
    wxSIGFPE,
    wxSIGKILL,      // forcefully kill, dangerous!
    wxSIGBUS,
    wxSIGSEGV,
    wxSIGSYS,
    wxSIGPIPE,
    wxSIGALRM,
    wxSIGTERM       // terminate the process gently
};
```

wxSIGNONE, wxSIGKILL and wxSIGTERM have the same meaning under both Unix and Windows but all the other signals are equivalent to wxSIGTERM under Windows.

Returns 0 on success, -1 on failure. If rc parameter is not NULL, it will be filled with an element of

wxKillError enum:

```
enum wxKillError
{
    wxKILL_OK,              // no error
    wxKILL_BAD_SIGNAL,      // no such signal
    wxKILL_ACCESS_DENIED,   // permission denied
    wxKILL_NO_PROCESS,      // no such process
    wxKILL_ERROR            // another, unspecified error
};
```

The flags parameter can be wxKILL_NOCHILDREN (the default), or wxKILL_CHILDREN, in which case the child processes of this process will be killed too. Note that under Unix, for wxKILL_CHILDREN to work you should have created the process by passing wxEXEC_MAKE_GROUP_LEADER to wxExecute.

wxProcess::Kill, wxProcess::Exists, Exec sample

<wx/utils.h>

## ::wxGetProcessId

unsigned long wxGetProcessId()

Returns the number uniquely identifying the current process in the system.

If an error occurs, 0 is returned.

<wx/utils.h>

## ::wxShell

bool wxShell(const wxString& command = NULL)

Executes a command in an interactive shell window. If no command is specified, then just the shell is spawned.

参考：wxExecute, Exec sample.

<wx/utils.h>

## ::wxShutdown

bool wxShutdown(wxShutdownFlagsflags)

This function shuts down or reboots the computer depending on the value of the flags. Please notice that doing this requires the corresponding access rights (superuser under Unix, SE_SHUTDOWN privelege under Windows NT) and that this function is only implemented under Unix and Win32.

flags

Either wxSHUTDOWN_POWEROFF or wxSHUTDOWN_REBOOT

true on success, false if an error occurred.

<wx/utils.h>