# RunAsUser DLL

*Version 1.08*

*Software Documentation*




www.runasuser.com
support@southsoftware.com

# 1 Contents



# 2 Abstract

RunAsUser DLL runs programs with supplied user account credentials on the desktop where another user is logged on. RunAsUser DLL performs all the necessary actions required to run the program such as logging on a user into the system, providing access to desktop, loading user profile, creating a process on the specified terminal session, etc. The program also supports passwordless logging into the system (ZwCreateToken/NtCreateToken), running under LocalSystem and running with a username of a user that is working in the system at the moment. RunAsUser DLL supports all 32 and 64-bit platforms based on Windows NT including Windows NT 4.0, 2000, XP, 2003, 2008, Vista, 7, 2012, 8.

## 3 Registration

Your trial of RunAsUser DLL will expire in 30 days after installation. When your trial period is expired you will get error 1931 (ERROR_CONTEXT_EXPIRED) as a result of main RunAsUser DLL functions.
If you wish to continue using RunAsUser DLL or if you wish to redistribute it with your own application then you have to register your copy of RunAsUser DLL with us.
To register RunAsUser DLL follow this link:
http://www.runasuser.com/purchase.html
After registering you will receive full version of RunAsUser DLL and your registration key. The full version of RunAsUser DLL has the same interface as trial version but it is smaller in size and works faster. The source code below demonstrates how to apply your registration key.

```
RUNASUSERCONFIGURATION runasuser;
runasuser.pszTempServiceName = _T("Temp service name here");
runasuser.pszTempServiceExe = NULL;
runasuser.pszRegistrationKey = _T("Your registration key here");
RunAsUser_Initialize(&runasuser);
```

## 4 How to use

RunAsUser DLL package includes the following files:

```
include\
  runasuser.h – C/C++ header file
  runasuser.inc – Delphi include file
dll\
  runasuserx32a.dll – ANSI DLL
  runasuserx32a.lib – ANSI LIB
  runasuserx32w.dll – UNICODE DLL
  runasuserx32w.lib – UNICODE LIB
  runasuserx64a.dll – ANSI DLL
  runasuserx64a.lib – ANSI LIB
  runasuserx64w.dll – UNICODE DLL
  runasuserx64w.lib – UNICODE LIB
runasuser.pdf – this documentation
runasuser.url – URL to the webpage
files.txt – list of files
license.txt – license agreement
readme.txt – readme file
register.txt – registration information
samples\
  application\
    vc6\ - Microsoft Visual C++ 6 GUI sample
```

| |
|---|
| vc9\ – Microsoft Visual C++ 2008 GUI sample<br>vb9\ – Microsoft Visual Basic 2008 GUI sample<br>cs9\ – Microsoft C Sharp 2008 GUI sample<br>delphi\ – Delphi 6 GUI sample<br>console\<br>vc6\ – Microsoft Visual C++ 6 console sample<br>vc9\ – Microsoft Visual C++ 2008 console sample<br>vb9\ – Microsoft Visual Basic 2008 console sample<br>cs9\ – Microsoft C Sharp 2008 console sample<br>delphi\ – Delphi 6 console sample |

To use any functions from RunAsUser DLL you need to link your application with one of the following files:
runasuserx32a.dll - 32-bit ASCII DLL;
runasuserx32w.dll - 32-bit UNICODE DLL;
runasuserx64a.dll - 64-bit ASCII DLL;
runasuserx64w.dll - 64-bit UNICODE DLL.
RunAsUser DLL has three main functions.

RunAsUser_Initialize() – Initializes the library. This function should be called prior to using other functions.

RunAsUser_CommandLine() – Checks the command line and performs all the internal operations connected with it. It should be called when you start the program that uses RunAsUser DLL. If the function carries out internal operations then the program that called the function should be terminated immediately with the exit code returned by this function.

RunAsUser_CreateProcess() – Main function that is actually start process as user.

All these functions are declared in the runasuser.h file. The library also exports many additional functions that can be used in the program (they do not influence the work of the library, we just made some of the internal functions available).

## *5 Building samples*

Samples from the samples folder of the package demonstrate how the library works. The samples use the runasuserx32a.dll or runasuserx32w.dll library so make sure that the appropriate library file is properly located in the same folder as the executable file of the sample that you are building.

If the "This application has failed to start because runasuserx32a.dll was not found. Re-installing the application may fix this problem." message is displayed then you do not have the runasuserx32a.dll file in the same folder as the executable file. The "StartService error 1053" message means the same.

## 6 Required privileges

Application that uses the `RunAsUser_CreateProcess()` function must be running under an account from Administrators group or as LocalSystem. The following privileges must be enabled for that account:

| *Name* | *Constant* | *Purpose* |
|---|---|---|
| Act as part of the operating system | `SeTcbPrivilege` | For successful call of `LogonUser` and `CreateProcessAsUser` |
| Replace a process-level token | `SeAssignPrimaryTokenPrivilege` | For successful call of `CreateProcessAsUser` |
| Adjust memory quotas for a process (may be named "Increase memory quotas" or "Increase quotas" in Windows 2000 and NT 4.0) | `SeIncreaseQuotaPrivilege` | For successful call of `CreateProcessAsUser` |
| Bypass traversy checking | `SeChangeNotifyPrivilege` | For successful call of `CreateProcessAsUser` |
| Debug programs | `SeDebugPrivilege` | For successful call of `TerminateProcess` |
| Create a token object | `SeCreateTokenPrivilege` | For successful call of `ZwCreateToken/NtCreateToken` |

**Note:** In later versions of Windows (2003, Vista) the `SeCreateTokenPrivilege` privilege can be given only to the users from the Administrators group and cannot be provided for LocalSystem, while in previous versions of Windows (NT, 2000, XP) LocalSystem has `SeCreateTokenPrivilege` by default.

**Note:** Windows XP Home and Windows Vista Home do not contain secpol.msc snap-in, which provides User Rights Assignment section to the Control Panel, so if that is your case, you need some third-party tool to modify your account privileges. You can download our Polsedit tool from http://www.southsoftware.com/. It will let you modify your account privileges and it is completely free to use.

## 7 How it works

Running the program as a certain user while another user is working in the system requires a lot of preparatory operations. They can require an additional process or even an additional service to be run, depending on the initial conditions (the way of logging on into the system, the active terminal session, etc.).

Generally, the process of running a program includes the following operations:

## 7.1 Logging on into the system

Depending on the selected way of logging on different privileges should be assigned to the calling process.
To start process as LocalSystem, the calling process should be running under an account from Administrators group.
To log on into the system using a username and password, the calling process should have enough privileges for calling LogonUser API.
To be able to run a program as a user without having the user's password the calling process should have enough privileges for calling ZwCreateToken/NtCreateToken.

## 7.2 Providing access to WindowStation and Desktop

Before using CreateProcessAsUser/CreateProcessAsUserW API to run the program, you should allow the user to access WindowStation and Desktop on the selected terminal session. If you do not do it, the launched program would not be able to initialize and will be terminated with an error, most likely with a code 0xC0000142 ("The application failed to initialize properly"). This error occurs when user32.dll cannot get access to WindowStation or Desktop during initialization. If the program does not need user32.dll to be loaded, it can start and work normally in any terminal session even if the access to the desktop is forbidden for the user who runs the program.
RunAsUser DLL uses a complicated algorithm allowing to give the user access to the desktop on the required terminal session. Depending on the initial conditions this algorithm provides running additional processes that perform the necessary operations for getting access to the desktop. The current process with a special command line is used as an additional process. The RunAsUser_CommandLine() function is used for processing this command line.
Providing access to WindowStation and Desktop consists in changing the DACL of the necessary WindowStation and Desktop on the required terminal session, so that the necessary user has full access to these objects.

## 7.3 Setting the terminal session

Windows Terminal Services allow several users to log in to the system simultaneously. When a user logs in to the system, the user is given his or her own terminal session, so before calling CreateProcessAsUser/CreateProcessAsUserW API you should provide that the process is created in the required session.

## 7.4 Loading the user profile

To load the user profile the program uses functions from userenv.dll. These functions allow setting the variables of the environment correctly and loading the HKEY_CURRENT_USER registry key for the specified user.

## 7.5 Running the program

To create a process as a specified user the program uses the CreateProcessAsUser() function. RunAsUser DLL always uses the UNICODE version of this function - CreateProcessAsUserW – in both UNICODE and ANSI versions of the library, as CreateProcessAsUserA has some problems in older version of Windows.

# 8 Functions

Include file: `runasuser.h`
Note: All structures in this file are 4-byte aligned.

## 8.1 Main functions

Main functions and structures to be used in this implementation.

### 8.1.1 RunAsUser_GetVersion

**SUMMARY:**
Current version of the library.

**DECLARATION:**
```
DWORD WINAPI RunAsUser_GetVersion(void);
```

**RETURN VALUE:**
`MAKELONG(minor, major)`.

### 8.1.2 RunAsUser_Initialize

**SUMMARY:**
Initialization interface. Must be called in the main function when program starts.

**ARGUMENTS:**
`lprunasuser` – pointer to the configuration structure.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_Initialize(LPRUNASUSERCONFIGURATION
lprunasuser);
```

**RETURN VALUE:**
If `TRUE` then initialization has been succeeded.

### 8.1.3 RunAsUser_CommandLine

**SUMMARY:**
Checks command line arguments and executes internal tasks if necessary. Must be called in the main function when program starts.

**ARGUMENTS:**
`lpszCommandLine` – command line to examine;
`pdwExitCode` – if return value is `TRUE` then this value contains process exit code.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_CommandLine(LPCTSTR lpszCommandLine,
LPDWORD pdwExitCode);
```

**RETURN VALUE:**
If `TRUE`, process must be terminated with the returned exit code.

### 8.1.4 RunAsUser_CreateProcess

**SUMMARY:**
Creates a new process and its primary thread. The new process runs in the security context of the specified user account.

**ARGUMENTS:**
`runasType` – type of authorization (`RUNASTYPE_` constants);
`pszEXE` – the name of the module to be executed;

`pszCmdLine` – the command line to be executed;
`pszDomainName` – the domain or server whose account database contains the user account;
`pszUserName` – the name of the user;
`pszPassword` – the clear-text password for the user account specified by `pszUserName`;
`pszDesktop` – the name of the desktop (Default desktop: "`WinSta0\Default`"). The desktop must exist at the moment of the function call;
`pszStartIn` – the full path to the current directory of the process;
`nCmdShow` – the window show state. Can be any of the `SW_` constants defined in `WINUSER.H`;
`dwSession` – the identifier of the session in which the process to be created;
`fImpersonate` – impersonate user before calling the `CreateProcessAsUser()` function;
`fElevate` – create process with highest privileges (Windows Vista and later);
`lpfnRunAsUser_State` – callback function that receives the execution status and extended error log. This parameter **cannot** be `NULL`;
`lpfnRunAsUser_Wait` – callback function that receives handle and identifier of the created process. In this function you can communicate with the created process in any way. This parameter can be `NULL`;
`lParam` – a user-defined value that will be passed to `lpfnRunAsUser_State` and `lpfnRunAsUser_Wait` callback functions.

**DECLARATION:**

```
BOOL WINAPI RunAsUser_CreateProcess(
    RUNASTYPE runasType,
    LPCTSTR pszEXE,
    LPCTSTR pszCmdLine,
    LPCTSTR pszDomainName,
    LPCTSTR pszUserName,
    LPCTSTR pszPassword,
    LPCTSTR pszDesktop,
    LPCTSTR pszStartIn,
    UINT nCmdShow,
    DWORD dwSession,
    BOOL fImpersonate,
    BOOL fElevate,
    RunAsUser_StateFn lpfnRunAsUser_State,
    RunAsUser_WaitFn lpfnRunAsUser_Wait,
    LPARAM lParam);
```

**REMARKS:**
If the process is started with `RUNASTYPE_LOGONLOCALLYUSER` or `RUNASTYPE_PASSWORDLESSUSER` type, this function does not return until started process is terminated. It is very important to wait until the process is terminated before exiting this function because it is necessary to unload user profile. Make sure your application is not terminated in the `lpfnRunAsUser_State` or `lpfnRunAsUser_Wait` function. If it does then user profile will remain loaded.

**Desktop:** Although, in earlier versions of Windows the program can be run on the desktop that appears after you start the computer (when the system asks for the username and password necessary to log in), starting from Windows XP, Logon and Screen saver desktops are secure. However, it is still possible to start a process when there is no logged on user. To do so, your process must be running as service, `pszDesktop` must be "`Service-0x0-3e7$\Default`" and `dwSession` must be `0` - the process will be started on the hidden desktop.

**Impersonation:** RunAsUser_CreateProcess allows you to access the specified directory and executable image in the security context of the caller or the target user. If fImpersonate is FALSE then RunAsUser_CreateProcess accesses the directory and executable image in the security context of the caller. In this case, if the caller does not have access to the directory and executable image, the function fails. Specify TRUE to access the directory and executable image in the security context of the target user.

**Elevation:** On Windows Vista and later, this option allows creating process with highest privileges. Set this option to TRUE, if your process fails to start with error 740 (ERROR_ELEVATION_REQUIRED).

**RUNASTYPE_CURRENTUSER:** When Terminal Services is running, the process will be started on the dwSession session as user that is logged on in the dwSession session. If Terminal Services is not running then process will be started as user that is working in the system for the moment.

**RUNASTYPE_LOGONLOCALLYUSER:** Because of clear-text password is used for this function, you may wish to clear the password from memory by calling the SecureZeroMemory function.
If you wish to logon a user with blank password, then you will need to disable the restriction against logging on to the network without a password. You can do so by Local Security Policy: Control Panel\Administrative Tools\Local Security Policy\Security Settings\Local Policies\Security Options. The name of the policy is "Accounts: Limit local account use of blank passwords to console logon only". By default, this option is enabled.

This example shows how to start process with RUNASTYPE_CURRENTUSER type:

```
RunAsUser_CreateProcess(
    RUNASTYPE_CURRENTUSER,
    _T("cmd.exe"),
    _T("/C dir"),
    _T(""),
    _T(""),
    _T(""),
    _T("WinSta0\\Default"),
    _T("c:\\"),
    SW_SHOWNORMAL,
    dwSessionId,
    FALSE, FALSE,
    RunAsUser_State,
    RunAsUser_Wait,
    NULL);
```

This example shows how to start process with RUNASTYPE_THISPROCESSUSER type:

```
RunAsUser_CreateProcess(
    RUNASTYPE_THISPROCESSUSER,
    _T("cmd.exe"),
    _T("/C dir"),
    _T(""),
    _T(""),
    _T(""),
    _T("WinSta0\\Default"),
    _T("c:\\"),
```

```
    SW_SHOWNORMAL,
    dwSessionId,
    FALSE, FALSE,
    RunAsUser_State,
    RunAsUser_Wait,
    NULL);
```

This example shows how to start process with `RUNASTYPE_LOGONLOCALLYUSER` type:

```
RunAsUser_CreateProcess(
    RUNASTYPE_LOGONLOCALLYUSER,
    _T("cmd.exe"),
    _T("/C dir"),
    szDomainName,
    szUserName,
    szPassword,
    _T("WinSta0\\Default"),
    _T("c:\\"),
    SW_SHOWNORMAL,
    dwSessionId,
    FALSE, FALSE,
    RunAsUser_State,
    RunAsUser_Wait,
    NULL);
```

This example shows how to start process with `RUNASTYPE_PASSWORDLESSUSER` type:

```
RunAsUser_CreateProcess(
    RUNASTYPE_PASSWORDLESSUSER,
    _T("cmd.exe"),
    _T("/C dir"),
    szDomainName,
    szUserName,
    _T(""),
    _T("WinSta0\\Default"),
    _T("c:\\"),
    SW_SHOWNORMAL,
    dwSessionId,
    FALSE, FALSE,
    RunAsUser_State,
    RunAsUser_Wait,
    NULL);
```

This example shows how to start process with `RUNASTYPE_LOCALSYSTEM` type:

```
RunAsUser_CreateProcess(
    RUNASTYPE_LOCALSYSTEM,
    _T("cmd.exe"),
    _T("/C dir"),
    _T(""),
    _T(""),
    _T(""),
    _T("WinSta0\\Default"),
    _T("c:\\"),
    SW_SHOWNORMAL,
    dwSessionId,
```

```
    FALSE, FALSE,
    RunAsUser_State,
    RunAsUser_Wait,
    NULL);
```

**RETURN VALUE:**
Returns `TRUE` if function succeeded, otherwise returns `FALSE`.
`GetLastError()` can be used for extended error information.

## 8.2 Extra functions

Extra common functions that we made available from this implementation.

### 8.2.1 RunAsUser_CreateUUID

**SUMMARY:**
Create a Universally Unique Identifier (UUID). A UUID is a 16-octet (128-bit) number represented by 32 hexadecimal digits, displayed in five groups separated by hyphens, in the following form: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.

**ARGUMENTS:**
`lpszUuid` – buffer for UUID string;
`cUuid` – size, in `TCHAR`s, of the `lpszUuid` buffer.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_CreateUUID(LPTSTR lpszUuid, DWORD cUuid);
```

**RETURN VALUE:**
Returns `TRUE` if function succeeded, otherwise returns `FALSE`.
`GetLastError()` can be used for extended error information.

### 8.2.2 RunAsUser_GetCurrentUserNonWTS

**SUMMARY:**
Get current user and domain name (when Terminal Services is not running).

**ARGUMENTS:**
`pszSamUserName` – buffer for the SAM (Windows NT) account name;
`dwBufferLen` – size, in `TCHAR`s, of the `pszSamUserName` buffer.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_GetCurrentUserNonWTS(LPTSTR
pszSamUserName, DWORD dwBufferLen);
```

**RETURN VALUE:**
Returns `TRUE` if function succeeded, otherwise returns `FALSE`.
`GetLastError()` can be used for extended error information.

### 8.2.3 RunAsUser_SDDL2SID

**SUMMARY:**
Converts a string-format SID into a valid, functional security descriptor.

**ARGUMENTS:**
`pszSidStr` – pointer to a null-terminated string containing the SID to convert;

pSid – pointer to a variable that receives a pointer to the converted SID.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_SDDL2SID(LPCTSTR pszSidStr, PSID *pSid);
```

**RETURN VALUE:**
Returns TRUE if function succeeded, otherwise returns FALSE.
GetLastError() can be used for extended error information.
To free pSid, call the FreeSid() function.

### 8.2.4 RunAsUser_SDDL2SAM

**SUMMARY:**
Converts a string-format SID into an account name.

**ARGUMENTS:**
pszSidStr – pointer to a null-terminated string containing the SID to convert;
pszTarget – the domain or server whose account database contains the user account;
pszAccountName – pointer to a variable that receives an user account name.
dwBufferLen – size, in TCHARs, of the pszAccountName buffer.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_SDDL2SAM(LPCTSTR pszSidStr, LPCTSTR
pszTarget, LPTSTR pszAccountName, DWORD dwBufferLen);
```

**RETURN VALUE:**
Returns TRUE if function succeeded, otherwise returns FALSE.
GetLastError() can be used for extended error information.

### 8.2.5 RunAsUser_SID2SDDL

**SUMMARY:**
Converts a SID to a string format.

**ARGUMENTS:**
pSid – pointer to the security descriptor to convert;
pszSidStr – pointer to a variable that receives a null-terminated SID string;
dwBufferLen – size, in TCHARs, of the pszSidStr buffer.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_SID2SDDL(PSID pSid, LPTSTR pszSidStr,
DWORD dwBufferLen);
```

**RETURN VALUE:**
Returns TRUE if function succeeded, otherwise returns FALSE.
GetLastError() can be used for extended error information.

### 8.2.6 RunAsUser_Name2SID

**SUMMARY:**
Converts an account name to a binary SID.

**ARGUMENTS:**
pszUserName – the name of the user;

pszDomainName – the domain or server whose account database contains the user account;

**DECLARATION:**
```
PSID WINAPI RunAsUser_Name2SID(LPCTSTR pszUserName, LPCTSTR
pszDomainName);
```

**RETURN VALUE:**
Returns pointer to a binary SID if function succeeded, otherwise returns NULL.
GetLastError() can be used for extended error information.
To free the returned SID, call the LocalFree() function.

### 8.2.7 RunAsUser_AdjustPrivilege

**SUMMARY:**
Enables or disables a privilege for the calling process.

**ARGUMENTS:**
pszPrivilege – name of the privilege;
dwNewState – can be a combination of SE_PRIVILEGE_ constants defined in WINNT.H.
pdwOldState – receives the previous state of the privilege. This parameter can be NULL.

**DECLARATION:**
```
BOOL WINAPI RunAsUser_AdjustPrivilege(LPCTSTR pszPrivilege,
DWORD dwNewState, LPDWORD pdwOldState);
```

**RETURN VALUE:**
Returns TRUE if function succeeded, otherwise returns FALSE.
GetLastError() can be used for extended error information.

## 9 Structures

Include file: runasuser.h
Note: All structures in this file are 4-byte aligned.

## 9.1 RUNASUSERCONFIGURATION

**SUMMARY:**
Initialization structure.

**DECLARATION:**
```
typedef struct tagRUNASUSERCONFIGURATION {
    // temp service name
    LPCTSTR pszTempServiceName;
    // temp service executable
    LPCTSTR pszTempServiceExe;
    // your registration key
    LPCTSTR pszRegistrationKey;
} RUNASUSERCONFIGURATION, FAR *LPRUNASUSERCONFIGURATION;
```

## 10 Constants

Include file: runasuser.h

## 10.1 RUNASTYPE

**SUMMARY:**
Types of authorization.

**VALUES:**
`RUNASTYPE_CURRENTUSER` – use currently logged on user;
`RUNASTYPE_THISPROCESSUSER` – use current process' user;
`RUNASTYPE_LOGONLOCALLYUSER` – logon user locally (`pszDomainName`/`pszUserName`/`pszPassword`);
`RUNASTYPE_PASSWORDLESSUSER` – use passwordless authorization (`pszDomainName`/`pszUserName`). Additional privilege required: `SeCreateTokenPrivilege`.
`RUNASTYPE_LOCALSYSTEM` – use LocalSystem account to run a process.

**DECLARATION:**
```
typedef enum {
    RUNASTYPE_CURRENTUSER,
    RUNASTYPE_THISPROCESSUSER,
    RUNASTYPE_LOGONLOCALLYUSER,
    RUNASTYPE_PASSWORDLESSUSER,
    RUNASTYPE_LOCALSYSTEM
} RUNASTYPE;
```

## 10.2 RUNASSTATE

**SUMMARY:**
States for `RunAsUser_State()`.

**VALUES:**
`RUNASSTATE_STARTED` – function has started the application specified;
`RUNASSTATE_FINISHED` – function has finished its work.

**DECLARATION:**
```
typedef enum {
    RUNASSTATE_STARTED,
    RUNASSTATE_FINISHED
} RUNASSTATE;
```

# 11 Callback functions

Include file: `runasuser.h`

## 11.1 RunAsUser_StateFn

**SUMMARY:**
Callback function for `RunAsUser_CreateProcess()`.

**ARGUMENTS:**
`state` – state (`RUNASSTATE_` constants);
`dwError` – error code;
`pszErrorText` – debug information;
`lParam` – user-defined argument (`lParam` from `RunAsUser_CreateProcess()`).

**DECLARATION:**
```
typedef VOID (CALLBACK * RunAsUser_StateFn)(RUNASSTATE state,
DWORD dwError, LPCTSTR pszErrorText, LPARAM lParam);
```

**REMARKS:**
This example shows how to use `RunAsUser_State` function to write extended log to a file:

```
VOID CALLBACK RunAsUserState(RUNASSTATE state, DWORD dwError,
    LPCTSTR pszErrorText, LPARAM lParam)
{
    HANDLE hLogFile;
    DWORD dwWritten;
    TCHAR szLogText[256];
    hLogFile = CreateFile(szLogFile,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ, NULL, OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL, NULL);
    if (hLogFile != INVALID_HANDLE_VALUE)
    {
        SetFilePointer(hLogFile, 0, 0, FILE_END);
        _sntprintf(szLogText, sizeof(szLogText)/sizeof(TCHAR),
        _T("state = %u\r\ndwError = %u\r\npszErrorLog:\r\n"),
        state, dwError);
        WriteFile(hLogFile, szLogText,
        _tcslen(szLogText)*sizeof(TCHAR), &dwWritten, NULL);
        WriteFile(hLogFile, pszErrorText,
        _tcslen(pszErrorText)*sizeof(TCHAR), &dwWritten, NULL);
        CloseHandle(hLogFile);
    }
}
```

## *11.2 RunAsUser_WaitFn*

**SUMMARY:**
Callback function for `RunAsUser_CreateProcess()`.

**ARGUMENTS:**
`hProcess` – handle to the started process;
`dwProcessId` – process identifier (PID) of the started process;
`lParam` – **user-defined argument (**`lParam` **from** `RunAsUser_CreateProcess()`**).**

**DECLARATION:**
```
typedef VOID (CALLBACK * RunAsUser_WaitFn)(HANDLE hProcess,
DWORD dwProcessId, LPARAM lParam);
```

**REMARKS:**
The `hProcess` has full access to the started process. Do not close the `hProcess` handle in this callback function.

This example shows how to terminate the process after it is running for 10 seconds:

```
VOID CALLBACK RunAsUserWait(HANDLE hProcess,
    DWORD dwProcessId, LPARAM lParam)
{
    if (WaitForSingleObject(hProcess, 10000)==WAIT_TIMEOUT)
    {
        DWORD dwPrivDebug = SE_PRIVILEGE_USED_FOR_ACCESS;
        if (RunAsUser_AdjustPrivilege(SE_DEBUG_NAME,
            SE_PRIVILEGE_ENABLED, &dwPrivDebug))
        {
            TerminateProcess(hProcess, 0);
            RunAsUser_AdjustPrivilege(SE_DEBUG_NAME,
                dwPrivDebug, NULL);
        }
    }
}
```