EDI Threads for Windows API Help Index

The Index contains a list of all Help topics available for EDI Threads API. For information on how to use the Help system, press F1 or choose Using Help from the Help menu.

This help file contains an in-depth reference for all the EDI Threads constants and functions. Along with the sample files ***threads.pas*** and ***threads.c*** you should be able to add threads to your current applications quickly and efficiently.

Introduction

EDI Threads is a complete multi-threading library for Windows. With it you can easily create applications that use background printing, complex redrawing, repagination, serial port polling, text searches, and more. There is no need to wait for Windows NT or switch to OS/2. With EDI Threads, writing background processes for Windows applications becomes a snap.

Here are just **some** of the features of EDI Threads:

- Easy to use, even when converting existing applications.
- Almost any function can be turned into a thread.
- Speeds up application's response time.
- Automatic system-wide scheduler.
- Unlimited number of threads.
- Each thread uses it's own stack.
- Adjustable priorities and time slices.
- Pause or stop a thread at any time.
- There is practically no limit to what a thread can do, including Windows and DOS calls.
- NO ROYALTIES!

The threads operate independently of the main program. As soon as a thread is started, control is returned to the application. By applying this technique to a background operation you will drastically speed up the application's user response time. Instead of waiting for a task to finish, you can start it and return to the user without delay. No more coffee breaks during lengthy operations!

A thread does not need to be used only for specific tasks. For example, you can create a thread that does serial polling and exists for the duration of the application. By using a thread you can make the function linear rather than timer based, which is much more accurate yet doesn't impede performance.

To see a simple demonstration of EDI Threads, run **threads.exe**. It demonstrates how simple using EDI Threads is, as well as how impressive the results can be. Each ball and line is a separate thread, completely independant of the program. Examine the demos and read the API reference, then give it a try. You'll love how easy it is to use, and your customers will love how snappy your application becomes!

Constants

The following are constants which may be used with EDI Threads.

Constant	Description
TM_USER	The start of a user specified return code. Equivalent to TM_CONTINUE + a user action.
TM_QUIT	The thread has ended or should end immediately.
TM_CONTINUE	The thread can continue or is continuing.
TM_PAUSED	The thread is currently paused.
TS_DEFTIMESLICE	The default time slice that EDI Threads uses to execute threads. It is specified in milliseconds between executions. The smaller the number, the more often EDI Threads executes.
TS_DEFPRIORITY	The default priority assigned to a new thread. You can use this to reset a thread's priority to it's original value.

Types

The following are constants which may be used with EDI Threads.

PThreadRec

A pointer type to the private structure used by EDI Threads.

PThreadFunc

Thread function type. Each thread function to be called by EDI Threads must have this format. For Turbo Pascal for Windows:

```
Procedure ThreadFunc (Thread : PThreadRec;  Wnd : HWnd;  wParam : Word;  lParam :
  LongInt);
```

For Borland C:

```
VOID FAR PASCAL _export ThreadFunc (PThreadRec Thread,  HWND Wnd, WORD wParam,
  LONG lParam)
```

You must use the Windows API function **MakeProcInstance** to create the proper prolog before you use CreateThread or StartThread.

Thread system routines

Routines

Low level thread routines

Routines

High level thread routines

Routines

Usage and recommendations

Included on the disk you will find demo files showing the usage of EDI Threads. The TPW version is called ***threads.pas***, and the "C" version is called ***threads.c***. To see the demo, run ***threads.exe*** with File Manager.

We recommend that you carefully examine the API to understand what EDI Threads can do. Along with the demo programs, you should be able to quickly convert existing applications to use EDI Threads.

Common problems

The most common problem you will have with EDI Threads has to do with the fact that the *Data segment* does not equal the *stack segment*. For TPW this is not generally a problem. With "C" however, this can cause lots of strange problems. As a basic rule, anything that is restricted in a callback function or a DLL function, is also restricted in a thread function. If you understand that DS doesn't equal SS, and you work around it, you won't have such problems.

Another common problem occurs when you mix high level and low level routines carelessly. As a basic rule, if a thread is created using a low level routine, it should be terminated and disposed of using a low level routine. Same thing for a high level routine. For example, if you create a thread using StartThread, you should end it with EndThread. This will make sure that the thread is terminated, removed from the system scheduler, and finally disposed of. If you used the low level functions TerminateThread and DisposeThread, the thread would not be removed from the system scheduler, and EDI Threads would crash the next time it tried to execute a thread.

Other than that, programming a thread function is just like any other Windows function. As long as you remember to be nice and yield from time to time, you won't have any problems.

Copyright, disclaimer, license agreement

COPYRIGHT

DISCLAIMER

THIS INFORMATION IS PROVIDED "AS IS" WITHOUT REPRESENTATION OR WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY REPRESENTATIONS OR ENDORSEMENTS REGARDING THE USE OF, THE RESULTS OF, OR PERFORMANCE OF THE INFORMATION, ITS APPROPRIATENESS, ACCURACY, RELIABILITY, OR CORRECTNESS. THE ENTIRE RISK AS TO THE USE OF THIS INFORMATION IS ASSUMED BY THE USER. IN NO EVENT WILL ROBERT SALESAS, ESCHALON DEVELOPMENT INC. OR IT'S EMPLOYEES BE LIABLE FOR ANY DAMAGES, DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL, RESULTING FROM ANY DEFECT IN THE INFORMATION, EVEN IF ROBERT SALESAS OR ESCHALON DEVELOPMENT INC. HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS DISCLAIMER SHALL SUPERSEDE ANY VERBAL OR WRITTEN STATEMENT TO THE CONTRARY. IF YOU DO NOT ACCEPT THESE TERMS YOU MUST CEASE AND DESIST USING THIS PRODUCT IMMEDIATELY.

LICENSE AGREEMENT

Upon registration, your use of this package indicates your acceptance of the following terms and conditions:

1. Copyright: These programs and the related documentation are copyright. The sole owner is Robert Salesas. You may not use, copy, modify, or transfer the programs, documentation, or any copy except as expressly provided in this agreement.

2. License: You have the non-exclusive right to use any enclosed package only on a single computer at a time. You may load the program into your computers temporary memory (RAM). You may physically transfer the program from one computer to another, provided that the program is used on only one computer at a time. You may not distribute copies of the complete package or the accompanying documentation to others. You may not decompile, disassemble, reverse engineer, modify, or translate the program or the documentation. You may not attempt to unlock or bypass any copy protection utilized with the program. All other rights and uses not specifically granted in this license are reserved by Robert Salesas and/or Eschalon Development Inc.

3. Distribution: You may distribute **thrdutls.dll** with your product, without any royalties or restrictions as long as its intended use is as a supplement and compliment to your own package, and not for itself.

4. Back-up and Transfer: You may make one (1) copy of the program solely for back-up purposes. You must reproduce and include the copyright notice on the back-up copy. You may transfer the product to another party only if the other party agrees to the terms and conditions of this agreement and completes and returns a registration card to Eschalon Development Inc. If you transfer the program you must at the same time transfer the documentation and back-up copy or transfer the documentation and destroy the back-up copy.

5. Terms: This license is effective until terminated. You may terminate it by destroying the program, the documentation and copies thereof. This license will also terminate if you fail to comply with any terms or

conditions of this agreement. You agree upon such termination to destroy all copies of the program and of the documentation.

EDI Threads for Windows registration

If you like EDI Threads and you choose to use it, you **must** register it. You are not allowed to keep, use, or distribute EDI Threads with your applications unless you register. When you register, you receive the latest version of the package. It will not have the opening "ad box" that you get with the unregistered version and you will be allowed to redistribute it as per the license agreement.

Select **PRINT TOPIC** from the **FILE** menu to print a registration order form.

For information, orders, or site licenses, call or fax EschalonDevelopmentInc. at (604)520-1543, or leave an EMail message on Compuserve (76625,1320).

NAME/CONTACT:

COMPANY:

STREET:

CITY: STATE: ZIP:

COUNTRY: TELEPHONE: FAX:

License registration	$110.00 Cdn.	($95.00 US)	x ___ units	= $	_____
Source code - thrdutls.dll*	459.00 Cdn.	($395.00 US)	x ___ units	= $	_____
B.C. residents add 6% PST				$	_____
Canada residents add 7% GST				$	_____
Overseas orders add $5.00 US for shipping & handling				$	_____
*Does **not** include license registration number.				Total: $	_____

VISA #:* Expiration Date:

*will be charged in Canadian dollars

Name on Card: Signature:

- OR -

Mail a check or money-order payable to **Eschalon Development Inc.**, at:

110-2 Renaissance Square
New Westminster, BC
V3M 6K3 Canada

Please call for information on site licenses or volume discounts: (604) 520-1543

Eschalon Development product line as of Summer 1992

EDI Install for Windows - Version 1.3

EDI Install for Windows is the first in the EDI Install product line. It is a simple to use, complete installation utility for Windows applications. Features a nice Windows interface, prompts user for destination, creates Program Manager groups, multiple disk support, file compression included. Uses a simple .INF file that doesn't require programming or learning a new script language. You just list your files and go!

$45 US

EDI Install for MS-DOS - Version 1.0

EDI Install for MS-DOS is a complete implementation of EDI Install written specifically for MS-DOS. Features a pleasant text based windowed interface which gives your users a good first impression of your product. This versions uses the same format .INF file making it just as easy to learn as its Windows cousin. You can even share .INF files across Windows/MS-DOS platforms!

$45 US

EDI Install Pro for Windows - Version 1.0

Finally, an advanced installation utility that doesn't take days to learn. EDI Install Pro uses an enhanced version of the familiar.INF file format introduced in EDI Install for Windows. It is extremely easy to use, yet offers all the powerful features of products costing three times as much. There is very little, if anything, that you cannot accomplish with EDI Install Pro.

Here are just **some** of the features you'll find:

- Optional dithered and/or bit mapped background.
- Optional selectable components allow the user to install only what is needed or reinstall specific components.
- Program Manager group creation.
- .INI file creation and modification. Works on any .INI file.
- Display "ReadMe.Txt" file to the user.
- Smart progress bar displays percent completed by file size.
- User specified source and destination paths.
- Creates directory structure "on the fly", only as required.
- Version checking based on "newer files" and "user confirmation".
- Complete file compression support, including a redistributable graphical "unpacker".
- Multiple diskette installations supported.
- Custom DLL allows you do expand the capabilities of EDI Install Pro.
- And much, much more...

You can do almost anything you need with EDI Install Pro, just as it is. However, by allowing custom DLLs, you can expand the capabilities to include anything you need, such as custom setup information, custom component selection, advertising text, file encryption, autoexec.bat or config.sys modifications, and much more. We couldn't include every possible feature, but with a custom DLL you can add anything you need. If you are unable to write your own DLL we can usually create one for you at a reasonable cost.

$95 US

EDI Threads for Windows - Version 1.0

EDI Threads is a complete multi-threading library for Windows. With it you can easily create applications that use background printing, complex redrawing, repagination, serial port polling, text searches, and more. There is no need to wait for Windows NT or switch to OS/2. With EDI Threads, writing background processes for Windows applications becomes a snap.

Here are just **some** of the features you'll find:

- Easy to use, even when converting existing applications.
- Almost any function can be turned into a thread.
- Speeds up application's response time.
- Automatic system-wide scheduler.
- Unlimited number of threads.
- Each thread uses it's own stack.
- Adjustable priorities and time slices.
- Pause or stop a thread at any time.
- There is practically no limit to what a thread can do, including Windows and DOS calls.
- NO ROYALTIES!

The threads operate independently of the main program. As soon as a thread is started, control is returned to the application. By applying this technique to a background operation you will drastically speed up the application's user response time. Instead of waiting for a task to finish, you can start it and return to the user without delay. No more coffee breaks during lengthy operations!

$95 US/$490 US with source code

WinCLI/WinCLI Pro* - Version 3.0/1.0

WinCLI is a complete command line interface capable of running Windows applications as well as DOS applications. It replaces the standard DOS prompt on standard and enhanced mode systems. WinCLI also includes over 30 file management commands that are built-in to WinCLI Pro (ALIAS, ASSOCIATE, ATTRIB, CD, CLS, COPY, DATE, DEL, DIR, EXIT, FINDFILE, HELP, INFO, LABEL, MEM, MAKEDIR, MORE, MOVE, PATH, PROMPT, RENAME, RENDIR, RMDIR, SYSINFO, TIME, TITLE, VER, VERIFY, VOL, WHICH and many other extended commands).

WinCLI has full clipboard support, a detailed help system, command line editing, command line history, aliases (like Doskey macros in MS-DOS 5.0), a scroll-back buffer, selectable font and adjustable colors. Also included in the package is a complete File Manager replacement, settings utility for WinCLI, password system protector, clock & screen saver and various other small utilities.

$35 US WinCLI/$80 US WinCLI Pro

*Some features are only available in WinCLI Pro.

FileApp - Version 1.1 (available May 1992)

FileApp is a "quick and dirty" file manager that's powerful enough to handle all your routine tasks. It sports a friendly graphical interface with "buttons" for all management chores. Easily copy, move, rename, delete, edit, or run files. You can also copy,

move, rename, or delete entire directories. Provides system and disk information, as well as a complete on-line help system. Three different interfaces are available.

$25 US

LZSSLib - Version 1.0

Complete compression/decompression DLL for Windows. Quickly and easily add compression support to your current applications. Supports Visual Basic, Turbo Pascal for Windows, C/C++, Actor, Object Vision 2.0 and any language product that supports DLLs.

$25 US

Switcher - Version 1.2

This neat little utility quickly allows you to switch between various Windows video modes. Supports most drivers and fonts. Works with Windows 3.0 and 3.1. Included free with most of our products.

Free!

All of our products can be registered directly through us (click here for contact information) and you can download the shareware versions from CompuServe (WINADV section usually). We cannot ship unregistered copies, sorry.

Contact

We will periodically provide bug fixes and upgrades. You may contact us for the latest version. Minor changes and/or bug fixes are free; major revisions carry an upgrade fee. You always pay for shipping and handling.

For technical assistance, orders, site licenses or information on our other products, you may contact us at:

Eschalon Development Inc.
2 Renaissance Square, Suite #110
New Westminster, BC
V3M 6K3 Canada

Telephone/Fax: (604) 520-1543
CompuServe: 76625,1320

GetThrdUtlsVersion

TPW Declaration

```
Function GetThrdUtlsVersion : Word;
```

C Declaration

```
WORD FAR PASCAL GetThrdUtlsVersion(VOID);
```

Description

Returns the version number of the current EDI Threads library.

Remarks

Returns the major version number in the high byte and the minor version number in the low byte.

GetNumThreads

TPW Declaration

```
Function GetNumThreads : Word;
```

C Declaration

```
WORD FAR PASCAL GetNumThreads(VOID);
```

Description

Returns the total number of threads running in the system.

SetThrdUtlsTimeSlice

TPW Declaration

```
Procedure SetThrdUtlsTimeSlice(ATimeSlice : Word);
```

C Declaration

```
VOID FAR PASCAL SetThrdUtlsTimeSlice(WORD ATimeSlice);
```

Description

Sets the time slice that EDI Threads uses to execute threads.

Parameters

ATimeSlice	The amount of milliseconds between thread executions. The smaller the number, the more often EDI Threads executes the scheduled threads.

Remarks

There should be no need to change this setting. However, if you need **less** processing time, and you want to make the system run more smoothly, you can increase the time slice setting.

CreateThread

TPW Declaration

```
Function CreateThread(ThreadFunc : PThreadFunc;  StackSize : Word;
   Wnd : HWnd;  wParam : Word;  lParam : LongInt) : PThreadRec;
```

C Declaration

```
PThreadRec FAR PASCAL CreateThread(PThreadFunc ThreadFunc, WORD
   StackSize, WND Wnd, WORD wParam, LONG lParam);
```

Description

Allows you do create a system thread. This is the lowest level for creating a thread. **CreateThread** returns NIL if the thread could not be created, or a valid PThreadRec if it was. If you want the thread to be automatically scheduled use StartThread instead of **CreateThread**. Otherwise you'll have to manually execute the thread using ExecThread.

Parameters

ThreadFunc	Procedure instance of a thread function which is called by the thread scheduler. This is the actual function to get called as a separate thread. You **must** use the Windows API function **MakeProcInstance** to create the proper prolog code.
StackSize	The amount of stack space to allocate, in bytes. A stack of 5000 to 8000 bytes is generally enough, unless you do a lot of recursion.
Wnd	A window that you wish to associate with the thread. This value gets passed to your thread function. Optional.
wParam	A 16-bit value to pass to your thread function. Optional.
lParam	A 32-bit value to pass to your thread. Optional.

Remarks

You should note that each thread gets its own stack, therefore the same limitations and conditions that apply to DLLs, apply to threads. Be careful not to assume that DS=SS.

Note that the thread is not executed until you call AddThread to automatically schedule the thread, or ExecThread to specifically execute the thread.

DisposeThread

TPW Declaration

```
Procedure DisposeThread(Var Thread : PThreadRec);
```

C Declaration

```
VOID FAR PASCAL DisposeThread(PThreadRec *Thread);
```

Description

Disposes of a thread created by CreateThread.

Parameters

Thread	The thread to dispose of.

Remarks

This function will dispose of a thread even if it has not terminated. Such procedure is recommended, since your thread might have allocated memory or resources that it didn't get a chance to free. You should always use IsThreadFinished and/or TerminateThread before you call **DisposeThread**.

ExecThread

TPW Declaration

```
Function ExecThread(Thread : PThreadRec) : Word;
```

C Declaration

```
WORD FAR PASCAL ExecThread(PThreadRec Thread);
```

Description

Executes the specified thread. If this is the first time the thread is executed, EDI Threads prepares the stack and calls your function. If it has already been executed before and your thread yielded, the execution is continued right after the YieldThread command. If the thread has terminated, execution is ignored and TM_QUIT is returned, otherwise TM_CONTINUE or TM_PAUSED is returned.

Parameters

Thread	The thread to execute.

Remarks

Unless you need precise control over when your threads get executed, it is best to avoid this function and use the internal scheduler through AddThread.

YieldThread

TPW Declaration

```
Function YieldThread : Word;
```

C Declaration

```
WORD FAR PASCAL YieldThread(VOID);
```

Description

Yields control back to the system from the current thread. If the thread must terminate, TM_QUIT is returned, otherwise TM_CONTINUE is returned. Normally, you can terminate a thread whenever you need to, **but you absolutely must terminate if YieldThread returns TM_QUIT**.

Remarks

This function can only be called from within a thread.

Occasionally you will have to yield control from your thread because Windows does not use preemptive multitasking. If you never yield, other tasks will never get a chance to execute. Your thread function should look something like this:

```
Procedure ThreadFunc(Thread : PThreadRec;  Wnd : HWnd;  wParam : Word;  lParam :
            LongInt);  Export;
Begin

  { Initiate stuff... }

  Repeat

    { Do thread work...  polling serial port maybe? }

  Until (YieldThread = tm_Quit);

  { Clean up stuff... }

  ExitThread;
End;
```

ExitThread

TPW Declaration

```
Procedure ExitThread;
```

C Declaration

```
VOID FAR PASCAL ExitThread(VOID);
```

Description

Exits and terminates the current thread.

Remarks

This function can only be called from within a thread. See YieldThread.

TerminateThread

TPW Declaration

```
Procedure TerminateThread(Thread : PThreadRec);
```

C Declaration

```
VOID FAR PASCAL TerminateThread(PThreadRec Thread);
```

Description

Terminates a thread from outside the thread. This is used to force a thread to finish, even if it has not completed its task. Upon receiving our request, the thread will clean up and exit. See YieldThread.

Parameters

Thread	The thread to terminate.

Remarks

This is the proper way to force a thread to terminate. You should call this function before DisposeThread in order to make sure that it gets a chance to clean up properly.

SetThreadPriority

TPW Declaration

```
Procedure SetThreadPriority(Thread : PThreadRec;  Priority : Word);
```

C Declaration

```
VOID FAR PASCAL SetThreadPriority(PThreadRec Thread, WORD
   Priority);
```

Description

Sets the thread priority relative to all the other threads in the system.

Parameters

Thread	The thread for which you wish to change priority.
Paused	The new priority, relative to the other threads in the system.

Remarks

When you specify a priority, you are determining how CPU resources are allocated to the threads. The numbers you specify set the priority of the thread relative to the other threads that are running. (Therefore, these numbers cannot be translated into a fixed percentage of CPU time.) The higher the priority a thread has, the more CPU resources are allocated to it. Priorities range from 10 to 1000. The default priority is 100.

SetThreadPause

TPW Declaration

```
Procedure SetThreadPause(Thread : PThreadRec;  Paused : Bool);
```

C Declaration

```
VOID FAR PASCAL SetThreadPause(PThreadRec Thread, BOOL Paused);
```

Description

Pauses or unpauses a thread. A paused thread stays paused until **SetThreadPause** unpauses it, or TerminateThread forces the thread to quit.

Parameters

Thread	The thread to pause/unpause.
Paused	True to pause thread, false to unpause it.

Remarks

Generally, **SetThreadPause** will be used to suspend a CPU intensive thread, while the user selects something that requires immediate feedback. For example, you might pause background re pagination in a word processor if the user brings up the font requester. This would speed up the font generation in True Type and ATM, for example. Since your thread doesn't even notice it's paused, when you unpause it, it simply continues with whatever it was doing.

Note that if you pause a serial port polling thread for too long you will probably lose incoming data.

IsThreadPaused

TPW Declaration

```
Function IsThreadPaused(Thread : PThreadRec) : Bool;
```

C Declaration

```
BOOL FAR PASCAL IsThreadPaused(PThreadRec Thread);
```

Description

Returns TRUE if the specified thread is paused and FALSE if it isn't.

Parameters

Thread	The thread to examine.

Remarks

You can use this function to verify whether a thread is executing or if it has been paused at some other point. Used in conjunction with SetThreadPause, **IsThreadPaused** can be used to toggle the state of the thread.

IsThreadFinished

TPW Declaration

```
Function IsThreadFinished(Thread : PThreadRec) : Bool;
```

C Declaration

```
BOOL FAR PASCAL IsThreadFinished(PThreadRec Thread);
```

Description

Returns TRUE if the specified thread is finished and FALSE if it isn't..

Parameters

Thread	The thread to examine.

Remarks

You can use this function to verify if a thread has completed it's task. For example, the following loop would wait until the thread was finished:

```
While Not IsThreadFinished(Thread) Do
    While PeekMessage(M, 0, 0, 0, pm_Remove) Do
   Begin
     TranslateMessage(M);
     DispatchMessage(M);
   End;
```

AddThread

TPW Declaration

```
Function AddThread(Thread : PThreadRec) : Bool;
```

C Declaration

```
BOOL FAR PASCAL AddThread(PThreadRec Thread);
```

Description

Add a thread to the system scheduling supplied by EDI Threads. The thread will automatically be executed by EDI Threads periodically. You no longer have to call ExecThread yourself.

Parameters

Thread	The thread to add to the system scheduler.

Remarks

Once a thread has been added with **AddThread** you can terminate, remove and dispose of it using EndThread instead of TerminateThread and DisposeThread.

If you wish to use **TerminateThread** and **DisposeThread** you must remember to first call RemoveThread to remove it from the system scheduler. If you don't call **RemoveThread**, the system will crash the next time the system threads get executed.

RemoveThread

TPW Declaration

```
Procedure RemoveThread(Thread : PThreadRec);
```

C Declaration

```
VOID FAR PASCAL RemoveThread(PThreadRec Thread);
```

Description

Removes a thread from the EDI Threads system scheduler. It does not terminate nor dispose of the thread, it only removes it from the automatic scheduler. After calling **RemoveThread** you can either keep executing the thread using ExecThread or terminate and dispose of it with TerminateThread and DisposeThread.

Parameters

Thread	The thread to remove from the system scheduler.

Remarks

If you created the thread using StartThread or CreateThread and AddThread, and you want to terminate, remove and dispose of the thread, you should use EndThread instead of **RemoveThread**, **TerminateThread** and **DisposeThread**.

StartThread

TPW Declaration

```
Function StartThread(ThreadFunc : PThreadFunc;  StackSize : Word;
   Wnd : HWnd;  wParam : Word;  lParam : LongInt) : PThreadRec;
```

C Declaration

```
PThreadRec FAR PASCAL StartThread(PThreadFunc ThreadFunc, WORD
   StackSize, HWND Wnd, WORD wParam, LONG lParam);
```

Description

Allows you do create and schedule a system thread. **StartThread** returns NIL if the thread could not be created, or a valid PThreadRec if it was. The thread is automatically added to the system, and will be scheduled by EDI Threads. You simply start the thread and forget about it. Use EndThread to terminate, remove and dispose of the thread.

Parameters

ThreadFunc	Procedure instance of a thread function which is called by the thread scheduler. This is the actual function to get called as a separate thread. You **must** use the Windows API function **MakeProcInstance** to create the proper prolog code.
StackSize	The amount of stack space to allocate, in bytes. A stack of 5000 to 8000 bytes is generally enough, unless you do a lot of recursion.
Wnd	A window that you wish to associate with the thread. This value gets passed to your thread function. Optional.
wParam	A 16-bit value to pass to your thread function. Optional.
lParam	A 32-bit value to pass to your thread. Optional.

Remarks

You should note that each thread gets its own stack, therefore the same limitations and conditions that apply to DLLs, apply to threads. Be careful not to assume that DS=SS.

Note that the thread is not executed until the first system scheduling of all threads.

EndThread

TPW Declaration

```
Procedure EndThread(Var Thread : PThreadRec);
```

C Declaration

```
VOID FAR PASCAL EndThread(PThreadRec *Thread);
```

Description

Terminates, removes and disposes of the specified thread.

Parameters

Thread	The thread to terminate, remove and dispose of.

Remarks

Note that only the threads that were created with StartThread or added with AddThread will be ended. Any thread you created with CreateThread and did not add to the system using **AddThread** will not be ended. Instead you should use TerminateThread and DisposeThread.

ExecTaskThreads

TPW Declaration

```
Procedure ExecTaskThreads(Task : THandle);
```

C Declaration

```
VOID FAR PASCAL ExecTaskThreads(HANDLE Task);
```

Description

Executes all threads that belong to the specified task. See ExecThread for details on thread execution.

Parameters

Task	The handle of the task for which to execute all threads. The task handle is obtained using the Windows API function **GetCurrentTask**.

Remarks

For most applications it will be unnecessary to call this function. However, if you need more processing power than the built in scheduler provides, you can use a *PeekMessage* loop instead of the usual *GetMessage* loop, and call **ExecTaskThreads** from the loop. This will execute the threads as often as possible, yet still leave other applications the opportunity to run. For example:

```
AllDone := False;
Repeat
  If PeekMessage(Msg, 0, 0, 0, pm_NoRemove) Then
    Begin
      If GetMessage(Msg, 0, 0, 0) Then
        Begin
          TranslateMessage(Msg);
          DispatchMessage(Msg);
        End
      Else
        AllDone := True;
    End
  Else
    ExecTaskThreads(GetCurrentTask);
Until AllDone;
```

Note that only the threads that were created with StartThread or added with AddThread will be executed. Any thread you created with CreateThread and did not add to the system using **AddThread** will not be executed. Instead you should use ExecThread on each thread you created.

EndTaskThreads

TPW Declaration

```
Procedure EndTaskThreads(Task : THandle);
```

C Declaration

```
VOID FAR PASCAL EndTaskThreads(HANDLE Task);
```

Description

Terminates, removes and disposes of all threads that belong to the specified task.

Parameters

Task	The handle of the task for which to terminate all threads. The task handle is obtained using the Windows API function **GetCurrentTask**.

Remarks

This function allows you to quickly end all the threads in your program with one call. It will call EndThread for each system thread that belongs to your task.

Note that only the threads that were created with StartThread or added with AddThread will be ended. Any thread you created with CreateThread and did not add to the system using **AddThread** will not be ended. Instead you should use TerminateThread and DisposeThread.