Win32 API. Глобальные функции Windows

Подробное описание формата глобальных функций Windows

_CrtCheckMemory

int _CrtCheckMemory(void);

Возвращаемое значение

В случае успешного завершения проверки возвращает значение TRUE. В противном случае возвращает значение FALSE.

Описание

Данная функция производит проверку целостности блока памяти в куче отладки (действует только в отладочной версии приложения). При обнаружении ошибок в целостности кучи отладки или в ее заголовке, а также записи информации за пределами выделенных буферов, функция `_CrtCheckMemory` выдает отладочное сообщение с указанием выявленной ошибки. Если символ `_DEBUG` не определен, вызов данной функции удаляется препроцессором. Поведением функции `_CrtCheckMemory` можно управлять, устанавливая различные поля во флаге `_crtDbgFlag` с использование функции `_CrtSetDbgFlag`. Установка флага `_CRTDBG_CHECK_ALWAYS_DF` приводит к тому, что функция `_CrtCheckMemory` вызывается при каждом запросе на выделение памяти. Хотя это и замедляет выполнение приложения, зато ускоряет процесс выявления ошибок. Сброс флага `_CRTDBG_ALLOC_MEM_DF` отменяет проверку кучи и заставляет функцию `_CrtCheckMemory` всегда возвращать значение TRUE. Поскольку данная функция возвращает логическое значение, ее вызов может быть помещен в макрос _ASSERT, что позволит быстро локализовать возникшие ошибки. Описание данной функции содержится в файле заголовка crtdbg.h.

_CrtDumpMemoryLeaks

int _CrtDumpMemoryLeaks(void);

Возвращаемое значение

В случае обнаружения утечек памяти возвращает значение TRUE. В противном случае возвращает значение FALSE.

Описание

Данная функция производит распечатку информации по всем блокам памяти при обнаружении утечек памяти (действует только в отладочной версии приложения). Функция `_CrtDumpMemoryLeaks` проверяет, не возникло ли утечек памяти с момента запуска приложения. При обнаружении утечек памяти производится распечатка содержимого заголовков всех объектов, расположенных в куче, в удобочитаемой форме. Если символ `_DEBUG` не определен, вызов данной функции удаляется препроцессором. Данная функция, как правило, вызывается в конце работы приложения для проверки того, что вся выделенная в нем память освобождена. Для автоматического запуска этой функции в конце работы приложения достаточно с использованием функции `_CrtSetDbgFlag` установить поле `_CRTDBG_LEAK_CHECK_DF` во флаге `_crtDbgFlag`. Для получения информации о текущем состоянии кучи отладки функция `_CrtDumpMemoryLeaks` вызывает функцию `_CrtMemCheckpoint`. После этого она производит поиск не освобожденных блоков памяти. При обнаружении такого блока она вызывает функцию `_CMemDumpAllObjectsSince`, выводящую информацию обо всех блоках памяти, выделенных с момента запуска приложения на исполнение. По умолчанию блоки `_CRT_BLOCK` не включаются в эту распечатку. Для их включения в распечатку достаточно с использованием функции `_CrtSetDbgFlag` установить поле `_CRTDBG_CHECK_CRT_DF` во флаге `_crtDbgFlag`. Описание данной функции содержится в файле заголовка crtdbg.h.

_CrtSetDbgFlag

int _CrtSetDbgFlag(int newFlag);

Возвращаемое значение

Возвращает предыдущее состояние флага `_crtDbgFlag`.

Аргументы

`newFlag` - новое значение флага `_crtDbgFlag`. В данном флаге могут быть установлены следующие поля:

`_CRTDBG_ALLOC_MEM_DF` - разрешает добавление в кучу отладки нового блока и использование для него идентификаторов типа таких, как _CLIENT_BLOCK. Если он сброшен, добавляет блок в список кучи, но устанавливает для него тип `_IGNORE_BLOCK`. По умолчанию этот флаг установлен. Совместим с любыми макросами, задающими режим проверки кучи отладки;

`_CRTDBG_CHECK_ALWAYS_DF` - устанавливает режим вызова функции `_CrtCheckMemory` при каждом запросе на выделение и освобождение памяти в куче. Если он сброшен, функцию `_CrtCheckMemory` нужно вызывать в явном виде. По умолчанию этот флаг сброшен. При установке данного флага любые макросы, задающие режим проверки кучи отладки, игнорируются;

`_CRTDBG_CHECK_CRT_DF` - включает блоки типа `_CRT_BLOCK` в распечатку при обнаружении утечек памяти и несоответствия состояния памяти. Если он сброшен, внутренняя память библиотек времени исполнения приложения не распечатывается. По умолчанию этот флаг сброшен. Совместим с любыми макросами, задающими режим проверки кучи отладки;

`_CRTDBG_DELAY_FREE_MEM_DF` - сохраняет освобожденные блоки памяти в связном списке кучи, назначая им тип `_FREE_BLOCK` и заполняя их байты значениями `0xDD`. Если он сброшен, освобожденные блоки памяти не хранятся в связном списке кучи. По умолчанию этот флаг сброшен. Совместим с любыми макросами, задающими режим проверки кучи отладки;

`_CRTDBG_LEAK_CHECK_DF` - производит автоматическую проверку на отсутствие утечек памяти при завершении работы приложения, используя для этого вызов функции `_CrtDumpMemoryLeaks`. Если он сброшен, автоматическая проверка не осуществляется. По умолчанию этот флаг сброшен. Совместим с любыми макросами, задающими режим проверки кучи отладки;

`_CRTDBG_REPORT_FLAG` - сохраняет предыдущее значение флага. Используется при модификации флага для сохранения его текущего состояния во временной переменной.

Описание

Данная функция позволяет установить или изменить состояние флага `_crtDbgFlag`, управляющего режимом работы функций слежения за кучей отладки. (Действует только в отладочной версии приложения). Если символ `_DEBUG` не определен, вызов данной функции удаляется препроцессором. Макросы, задающие режим проверки кучи отладки, определяют число вызовов функций `malloc`, `realloc`, `free` и `_msize` за которыми последует вызов функции `_CrtCheckMemory`. Определены следующие макросы, задающие режим проверки кучи отладки:

`_CRTDBG_CHECK_EVERY_16_DF` - проверка производится после 16 вызовов;

`_CRTDBG_CHECK_EVERY_128_DF` - проверка производится после 128 вызовов;

`_CRTDBG_CHECK_EVERY_1024_DF` - проверка производится после 1024 вызовов;

`_CRTDBG_CHECK_DEFAULT_DF` - эквивалентен макросу `_CRTDBG_CHECK_EVERY_1024_DF`. Для задания этих макросов используются 16 старших разрядов аргумента `newFlag`.

Описание данной функции содержится в файле заголовка crtdbg.h.

AddFontResource

int AddFontResource(LPCTSTR lpszFilename);

Возвращаемое значение
В случае успешного завершения функции возвращается количество добавленных шрифтов, в противном случае возвращается нулевое значение. В Windows NT более подробную информацию об ошибке можно получить, вызвав функцию GetLastError.
Аргументы
lpszFilename - указатель на заканчивающуюся нулем текстовую строку, содержащую корректное имя файла шрифта. Имя файла шрифта имеет расширение .fon для файлов ресурса шрифта, расширение .fnt для файлов шрифта, содержащих битовые образы символов, расширение ttf для файлов шрифтов TrueType и расширение .fot для файлов ресурсов шрифтов TrueType.
Описание
Функция AddFontResource позволяет добавить ресурсы шрифта из указанного файла в системную таблицу шрифтов. После этого данный шрифт может быть использован для вывода текста любым приложением Win32. Любые приложения, добавляющие или удаляющие шрифты из системной таблицы шрифтов, извещают об этом другие приложения посылкой сообщения WM_FONTCHANGE всем окнам верхнего уровня в операционной системе. Для посылки этого сообщения приложение должно использовать функцию SendMessage, в аргументе hWnd которой должно стоять значение HWND_BROADCAST. Как только приложение перестает использовать ресурс шрифта, загруженный функцией AddFontResource, оно должно удалить его функцией **RemoveFontResource.**
AfxBeginThread
CWinThread* AfxBeginThread(AFX_THREADPROC pfnThreadProc, LPVOID pParam, int nPriority = THREAD_PRIORITY_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);
CWinThread* AfxBeginThread(CRuntimeClass* pThreadClass, int nPriority = THREAD_PRIORITY_NORMAL, UINT nStackSize = 0, DWORD dwCreateFlags = 0, LPSECURITY_ATTRIBUTES lpSecurityAttrs = NULL);
Возвращаемое значение
Указатель на созданный данной функцией объект класса потока.
Аргументы
pfnThreadProc - указатель на исполняющую функцию рабочего потока. Не может иметь нулевого значения. Исполняющая функция потока имеет следующий формат: UINT MyControllingFunction(LPVOID pParam); pThreadClass - указатель на структуру CRuntimeClass для пользовательского класса, производного от класса CWinThread.
pParam - аргумент, передаваемый исполняющей функции потока, как показано в описании аргумента pfnThreadProc.
nPriority - приоритет потока. Если эта величина равна нулю, то у создаваемого потока устанавливается такой же приоритет, как и у вызывающего потока. Полный список значений приоритетов приведен в описании функции SetThreadPriority.
nStackSize - определяет размер стека нового потока в байтах. Если эта величина равна нулю, то у создаваемого потока создается стек того же размера, что и у вызывающего потока.
dwCreateFlags - определяет дополнительный флаг, управляющий процессом создания потока. Этот флаг может иметь два значения:
CREATE_SUSPENDED - создает поток, в котором счетчик остановки установлен в единицу. Этот поток не будет исполняться, пока не будет вызвана функция ResumeThread;
0 - начать работу потока непосредственно после его создания.
lpSecurityAttrs - указатель на объект структуры SECURITY_ATTRIBUTES, определяющий атрибуты безопасности данного потока. Если эта величина равна нулю, то

создаваемый поток имеет те же атрибуты безопасности, что и вызывающий поток.
Описание
Данная функция позволяет создать новый поток. Первая версия данной функции создает рабочий поток. Вторая версия создает интерфейсный поток. Функция `AfxBeginThread` создает новый объект класса `CWinThread`, вызывает функцию `CreateThread` для начала исполнения потока, а также возвращает указатель на созданный объект класса потока. В процессе выполнения данной функции производятся все проверки, необходимые для того, чтобы гарантировать, что в случае возникновения ошибки в процессе выполнения данной функции все созданные до этого объекты будут соответствующим образом уничтожены. Для завершения работы потока следует вызвать функцию `AfxEndThread` из данного потока или выйти из исполняющей функции потока.
AfxCheckError
void AFXAPI AfxCheckError(SCODE sc); throw CMemoryException* throw COleException* Описание
Данная функция проверяет значение своего аргумента на то, что он содержит код ошибки. В случае положительного исхода этой проверки вызывается исключение. Если в качестве аргумента передан код `E_OUTOFMEMORY`, вызывается исключение `CMemoryException`. Для этого используется функция `AfxThrowMemoryException`. В противном случае, с использованием функции `AfxThrowOleException`, вызывается исключение `COleException`. Функция `AfxCheckError` одинаково работает как в отладочной, так и в распространяемой версиях приложения.
AfxCheckMemory
BOOL AfxCheckMemory();
Возвращаемое значение
Ненулевое, при отсутствии ошибок в памяти, и нулевое в противном случае.
Описание
Данная функция проверяет кучу и выводит сообщения об обнаруженных ошибках. Если ошибки отсутствуют, ничего не выводится. Производится проверка всех блоков памяти, размещенных в куче, включая блоки, выделенные операцией `new`. Блоки, выделенные на низком уровне, например, функциями `malloc` и `GlobalAlloc`, не проверяются. Список дефектных блоков памяти выводится в окно отладчика. Если в программе присутствует строка
#define new DEBUG_NEW
то при последующих вызовах функции `AfxCheckMemory` в выводимую информацию будет включено имя файла и номер строки, в которой был выделен этот блок памяти. Если приложение содержит классы, сохраняемые в потоке, то приведенная выше строка должно располагаться после последнего вызова макроса `IMPLEMENT_SERIAL`. Эта функция работает только в отладочной версии библиотеки MFC.
AfxDumpStack
void AFXAPI AfxDumpStack(DWORD dwTarget = AFX_STACK_DUMP_TARGET_DEFAULT);
Аргументы
`dwTarget` - указывает, куда будет записываться содержимое стека. Может представлять собой комбинацию следующих значений, объединяемых операцией ИЛИ (|):

`AFX_STACK_DUMP_TARGET_TRACE` - для вывода содержимого стека используется макрос `TRACE`. Этот макрос выводит информацию только в отладочном режиме. В окончательной версии никакой информации выводиться не будет. Выходной поток макроса `TRACE` может быть переназначен другому приемнику, помимо отладчика;

`AFX_STACK_DUMP_TARGET_DEFAULT` - при работе отладочной версии содержимое стека выводится макросом `TRACE`, а в окончательной версии оно помещается в буфер

обмена;

AFX_STACK_DUMP_TARGET_CLIPBOARD - помещает содержимое стека в буфер обмена. Запись производится в текстовом виде с использованием формата буфера обмена CF_TEXT;

AFX_STACK_DUMP_TARGET_BOTH - помещает содержимое стека в буфер обмена и одновременно выводит его макросом TRACE;

AFX_STACK_DUMP_TARGET_ODS - посылает содержимое стека непосредственно отладчику с использованием функции Win32 OutputDebugString. Отладчик, если он подключен к процессу, получит эту информацию как от отладочной, так и от окончательной версии. При установке флага AFX_STACK_DUMP_TARGET_ODS приемником информации может быть только отладчик (если он подключен).

Описание

Данная функция используется для создания образа стека. Каждой функции, содержащейся в стеке, соответствует строка в его образе. Каждая строка образа стека содержит адрес последнего вызова функции, полный путь к модулю, в котором произошел вызов и прототип вызываемой функции. Если вызов функции произошел не по указанному адресу, дополнительно указывается смещение вызова (в результате вызов может произойти не из указанной функции, а совсем из другой функции). Данная функция имеется как в отладочной, так и в распространяемой версиях библиотеки MFC. Эта функция всегда подключается статически. Даже в тех случаях, когда все остальные функции библиотеки MFC подключаются динамически. В этом случае ее реализация располагается в файле MFCS42.LIB или в его вариантах. Для обеспечения успешной работы данной функции необходимо обеспечить следующие условия:

приложение должно иметь доступ к каталогу, в котором расположен файл imagehlp.dll. В противном случае будет выдано сообщение об ошибке. Этот файл представляет собой библиотеку динамической компоновки, поставляемую совместно с Platform SDK и Windows; располагающиеся в стеке модули должны содержать отладочную информацию. В противном случае выдаваемая данной функцией информация будет не столь детальной.

AfxEnableControlContainer

void AfxEnableControlContainer();

Описание

Данная функция вызывается функцией CWinApp::InitInstance, чтобы обеспечить в приложении поддержку элементов управления OLE.

AfxEnableMemoryTracking

BOOL AfxEnableMemoryTracking(BOOL bTrack);

Возвращаемое значение

Возвращает предыдущее значение флага разрешения проверки памяти.

Аргументы

bTrack - флаг разрешения проверки памяти.

Описание

Позволяет установить или отменить режим трассировки памяти. По умолчанию в отладочной версии приложения MFC установлен режим проверки памяти. Чтобы отменить этот режим, достаточно вызвать функцию AfxEnableMemoryTracking с аргументом FALSE. Для восстановления этого режима необходимо передать в аргументе данной функции значение TRUE. Эта функция работает только в отладочной версии библиотеки MFC.

AfxEndThread

void AfxEndThread(UINT nExitCode);

Аргументы

nExitCode - определяет код завершения потока.

Описание

Данная функция используется для завершения текущего потока. Должна вызываться из потока, который необходимо завершить.

GetMessage

BOOL GetMessage(LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin, UINT wMsgFilterMax);

Возвращаемое значение

Если данная функция извлекает сообщение отличное от `WM_QUIT`, то возвращается ненулевая величина. В противном случае возвращается ноль. Если в процессе выполнения функции произошла ошибка, то возвращаемая величина равна -1. Например, ошибка может возникнуть, если аргумент `hWnd` не является дескриптором окна или аргумент `lpMsg` не указывает на объект структуры `MSG`. Дополнительную информацию по ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`lpMsg` - указатель на объект структуры `MSG`, в который будет записана информация из очереди сообщений потока.

`hWnd` - дескриптор окна, сообщения для которого следует получить. Этот аргумент может принимать нулевое значение. Это значение свидетельствует о том, что функция `GetMessage` должна получать сообщения для любого окна, принадлежащего данному потоку, и сообщения от потока, посланные функцией `PostThreadMessage`.

`wMsgFilterMin` - определяет нижнюю границу диапазона отыскиваемых сообщений.

`wMsgFilterMax` - определяет верхнюю границу диапазона отыскиваемых сообщений.

Описание

Функция `GetMessage` получает сообщения из очереди сообщений вызвавшего ее потока и помещает их в объект специальной структуры. Данная функция позволяет получать сообщения для любого окна, принадлежащего данному потоку, и сообщения от потока, посланные функцией `PostThreadMessage`. При поиске сообщений производится проверка на нахождение их в определенном диапазоне значений. Функция `GetMessage` не может отыскивать сообщения, направленные другим окнам или приложениям. Приложение использует возвращаемое данной функцией значение для определения того, нет ли необходимости выйти из главного цикла сообщений приложения и завершить его работу. Функция `GetMessage` позволяет получить сообщения, связанные с окном, определенным в аргументе `hWnd`, или с любым его дочерним окном, как это определено в функции `IsChild`. Получаемые сообщения должны находиться в диапазоне, заданном аргументами `wMsgFilterMin` и `wMsgFilterMax`. Если аргумент `hWnd` имеет нулевое значение, то функция `GetMessage` должна получать сообщения для любого окна, принадлежащего данному потоку, и сообщения от потока, посланные функцией `PostThreadMessage`. Даже в том случае, когда аргумент `hWnd` имеет нулевое значение, функция `GetMessage` не позволяет получать сообщения, предназначенные для окон, принадлежащих другим потокам, или от других потоков, кроме вызывающего. Сообщение от потока, направляемое функцией `PostThreadMessage` имеет нулевое значение параметра `hWnd`. Если оба аргумента `wMsgFilterMin` и `wMsgFilterMax` имеют нулевое значение, то функция `GetMessage` позволяет получить все возможные сообщения (фильтрация сообщений отсутствует). В качестве границ диапазона могут использоваться значения `WM_KEYFIRST` и `WM_KEYLAST`, позволяющие выделить все сообщения, связанные с клавиатурой, и значения `WM_MOUSEFIRST` и `WM_MOUSELAST`, позволяющие выделить все сообщения, связанные с мышью. Функция `GetMessage` не удаляет сообщение `WM_PAINT` из очереди сообщений. Оно остается там пока не будет обработано. Поскольку данная функция может возвращать положительные, нулевые и отрицательные значения, то при написании программы следует избегать использования таких выражений как:

while (GetMessage(lpMsg, hWnd, 0, 0)) {
...
}
поскольку данное выражение не реагирует на величину -1, свидетельствующую о возникновении фатальной ошибки в приложении.

GetWindowContextHelpId

DWORD GetWindowContextHelpId(HWND hwnd);

Возвращаемое значение

Возвращает контекстный идентификатор справки, если такой идентификатор связан с данным окном, и ноль в противном случае.

Аргументы

`hwnd` - дескриптор окна, для которого необходимо получить контекстный идентификатор справки.

Описание

Позволяет получить контекстный идентификатор справки для указанного окна, если с данным окном связан контекстный идентификатор справки.

GetWindowRect

BOOL GetWindowRect(HWND hWnd, LPRECT lpRect);

Возвращаемое значение

Ненулевое, если функция завершается успешно, и нулевое в противном случае. Дополнительную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`hWnd` - дескриптор окна.

`lpRect` - указатель на объект структуры `RECT`, содержащий экранные координаты верхнего левого и нижнего правого углов окна.

Описание

Функция `GetWindowRect` позволяет получить координаты прямоугольника, описывающего указанное окно. Координаты окна измеряются в экранных координатах, нулевая точка которых расположена в левом верхнем углу экрана.

HtmlHelp

HWND HtmlHelp(HWND hwndCaller, LPCSTR pszFile, UINT uCommand, DWORD dwData);

Возвращаемое значение

Дескриптор окна, в котором будет выводиться справочная информация.

Аргументы

`hwndCaller` - дескриптор окна, для которого вызывается справочная система.

`pszFile` - определяет файл HTML, URL, компилированный файл справочной системы HTML или определение окна (предваряемое символом '>'). Если используемая команда не предполагает использования файла или URL, может принимать нулевое значение.

`uCommand` - содержит выполняемую команду. Список команд приведен в примечании.

`dwData` - содержит информацию, используемую командой, определенной в аргументе `uCommand`.

Описание

Данная функция используется для вывода справочной информации в формате HTML. Создаваемое ею окно справочной системы является дочерним окном того окна, дескриптор которого передан в качестве аргумента данной функции, автоматически отображается поверх родительского окна и закрывается вместе с ним. Если справочная система будет посылать сообщения, они будут направляться родительскому окну. Для создаваемого окна могут быть заданы стили, координаты, заголовок и режим

отображения. Как уже говорилось выше, аргумент `uCommand` функции `HtmlHelp` определяет операцию, производимую данной функцией. Эта операция определяет тип файла, передаваемого в аргументе `pszFile`, и информацию, передаваемую в аргументе `dwData` данной функции. Список команд и содержимое аргументов приведены в таблице П2.1.

Таблица П2.1. Команды функции HtmlHelp

Команда	Описание	Содержимое аргумента pszFile	Содержимое аргумента dwData
HH_DISPLAY_TOPIC	Выводит файл HTML. Если тип окна не определен, используется окно, определенное по умолчанию. Если окно уже выведено, файл HTML заменяет его содержимое.	Файл, URL или скомпилированный файл HTML. Если в этом аргументе присутствует символ '>' за ним следует определение типа окна, в которое будет выводиться справочная информация.	Может содержать указатель на файл, URL, скомпилированный файл HTML или имя файла в имя скомпилированном файле HTML, на который указывает аргумент pszFile. Указатель может быть нулевым.
HH_DISPLAY_TEXT_POPUP	Выводит текст из строкового ресурса, текстовой строки или текстового файла во всплывающее окно.	Имя текстового файла или нулевое значение, если текст содержится в строковом ресурсе или объекте структуры HH_POPUP.	Указатель на объект структуры HH_POPUP.
HH_SET_WIN_TYPE	Создает новый или изменяет существующий тип окна.	Игнорируется.	Указатель на объект структуры HH_WINTYPE.
HH_GET_WIN_TYPE	Позволяет получить объект структуры HH_WINTYPE, связанный с типом окна. Если указанный тип окна не определен, возвращает значение 1. Если указанный тип окна определен, возвращает дескриптор окна (если окно не создано, возвращает нулевое	Содержит имя окна. Это имя должно начинаться с символа '>', перед которым может располагаться имя скомпилированного файла HTML.	Адрес указателя на объект структуры PHH_WINTYPE. В полученный объект структуры нельзя вносить изменения.

	значение).		
HH_GET_WIN_HANDLE	Получает дескриптор окна, связанного с указанным типом окна. Если указанный тип окна не определен, возвращает нулевое значение.	Игнорируется	Указатель на строку, содержащую имя типа окна.
HH_SET_INFO_TYPES	Устанавливает все типы информации, которые будут выводиться в данном окне. В окне, содержащем три панели, это приведет к перерисовке навигационной панели (если она отображается).	Игнорируется	Указатель на объект структуры HH_WINTYPE. В этом объекте структуры должны быть заполнены переменные cbStruct и ainfoTypes.
HH_SYNC	Синхронизирует оглавление в окне, содержащем три панели, с указанным URL. Эта команда используется только в окнах, не поддерживающих автоматическую синхронизацию.	Содержит имя синхронизируемого окна.	Содержит синхронизирующий URL, который может в настоящее время и не выводиться.
HH_ADD_NAV_UI	Добавляет окно навигации в навигационную панель окна, содержащего три панели.	Указатель на функцию HhWinCallBack, поддерживающую новый UI.	Уникальный числовой идентификатор нового UI. Переключение между UI осуществляется изменением значения переменной curNavType объекта структуры HH_WINTYPE.
HH_ADD_BUTTON	Добавляет кнопку в панель инструментов окна,	Указатель на функцию HhWinCallBack, поддерживающую	Уникальный числовой идентификатор новой кнопки.

	содержащего три панели.	новую кнопку.	
HH_KEYWORD_LOOKUP	Производит поиск указанного ключевого слова в файле .hhk. При его обнаружении соответствующая тема выводится в указанном (или в текущем, если не указано) окне.	Если содержит нулевое значение, поиск производится в файле .hhk, связанным с текущим окном. Если содержит строку, содержащую описание окна, поиск производится в файле .hhk, связанным с данным окном. В нем же будут отображаться найденные темы. Если указан файл .hhk, то поиск будет производиться в этом файле, а отображение - в текущем окне.	Указатель на строку, содержащую одно или несколько ключевых слов, разделенных точкой с запятой (;).

AfxGetApp

CWinApp* AfxGetApp();

Возвращаемое значение

Указатель на единственный объект класса `CWinApp` данного приложения.

Описание

Указатель, возвращаемый данной функцией, может быть использован для доступа к информации о данном приложении, такой, как главная процедура обработки сообщения или объект главного окна программы.

AfxGetResourceHandle

HINSTANCE AfxGetResourceHandle();

Возвращаемое значение

Дескриптор `HINSTANCE` ресурсов, принадлежащих данному приложению по умолчанию.

Описание

Возвращаемый данной функцией дескриптор может быть использован для непосредственного доступа к ресурсам приложения, например, при вызове функции Windows `FindResource`.

AfxInitExtensionModule

BOOL AFXAPI AfxInitExtensionModule(AFX_EXTENSION_MODULE& state, HMODULE hModule);

Возвращаемое значение

Если инициализация библиотеки динамической компоновки прошла успешно, возвращает значение `TRUE`. В противном случае возвращает значение `FALSE`.

Аргументы

`state` - ссылка на объект структуры AFX_EXTENSION_MODULE, в котором будет сохранено состояние библиотеки расширения MFC после ее инициализации. Помимо прочего, в ней содержится состояние объектов классов, инициализированных

статическими конструкторами до вызова функции DllMain.
hModule - дескриптор модуля библиотеки расширения MFC.
Описание
Данная функция вызывается в функции DllMain для инициализации библиотеки расширения MFC. Функция AfxInitExtensionModule копирует HMODULE библиотеки динамической компоновки и сохраняет объекты структуры CRuntimeClass и фабрики объектов (объекты COleObjectFactory) для последующего использования при создании объекта CDynLinkLibrary. В функции DllMain библиотеки расширения MFC необходимо произвести две операции:
вызвать функцию AfxInitExtensionModule и проверить возвращаемое ею значение;
создать объект класса CDynLinkLibrary, если библиотека динамической компоновки экспортирует объекты CRuntimeClass или имеет собственные пользовательские ресурсы.
При завершении работы процесса или отключении библиотек динамической компоновки вызовом функции AfxFreeLibrary можно вызвать функцию AfxTermExtensionModule для очистки библиотеки расширения MFC.
AfxIsMemoryBlock
BOOL AfxIsMemoryBlock(const void* p, UINT nBytes, LONG* plRequestNumber = NULL);
Возвращаемое значение
Ненулевое, если блок памяти выделен и имеет указанный размер, и нулевое в противном случае.
Аргументы
p - указатель на проверяемый блок памяти.
nBytes - размер проверяемого блока в байтах.
plRequestNumber - указатель на переменную типа long, в которую будет записан номер блока в последовательности. Эта переменная заполняется только в том случае, если данная функция возвратила ненулевое значение.
Описание
Данная функция проверяет адрес в памяти, чтобы убедиться в том, что он соответствует активному блоку памяти, выделенному диагностической версией операции new. Кроме того, производится проверка соответствия указанного размера блока его истинным размерам. Если функция AfxIsMemoryBlock возвращает ненулевое значение, в переменную, на которую указывает аргумент plRequestNumber, записывается номер блока в последовательности, представляющий собой порядковый номер вызова операции new при создании данного блока.
AfxIsValidAddress
BOOL AfxIsValidAddress(const void* lp, UINT nBytes, BOOL bReadWrite = TRUE);
Возвращаемое значение
Ненулевое, если указанный блок памяти полностью расположен в области памяти, выделенной данному приложению, и нулевое в противном случае.
Аргументы
lp - указатель на проверяемый адрес в оперативной памяти.
nBytes - содержит размер проверяемой области памяти.
bReadWrite - определяет режим доступа к памяти: по чтению и записи (TRUE) или только для чтения (FALSE).
Описание
Данная функция проверяет указанный блок, чтобы убедиться в том, что он полностью расположен в области памяти, выделенной данному приложению. Эта проверка не ограничивается блоками памяти, выделенными операцией new.
AfxIsValidString

BOOL AfxIsValidString(LPCSTR lpsz, int nLength = -1);

Возвращаемое значение

Ненулевое, если передан указатель на строку указанного размера, и нулевое в противном случае.

Аргументы

`lpsz` - проверяемый указатель.

`nLength` - длина проверяемой строки в байтах. Значение -1 указывает на то, что строка завершается нулем.

Описание

Данная функция используется для определения того, что переданный ей указатель указывает на корректную строку.

AfxMessageBox

int AfxMessageBox(LPCTSTR lpszText, UINT nType = MB_OK, UINT nIDHelp = 0); int AFXAPI AfxMessageBox(UINT nIDPrompt, UINT nType = MB_OK, UINT nIDHelp = (UINT) -1);

Возвращаемое значение

Если для вывода диалогового окна недостаточно памяти, возвращает нулевое значение. В противном случае возвращает одно из следующих значений:

`IDABORT` - была нажата кнопка Abort (Прекращение);

`IDCANCEL` - была нажата кнопка Cancel (Отмена);

`IDIGNORE` - была нажата кнопка Ignore (Пропуск);

`IDNO` - была нажата кнопка No (Нет);

`IDOK` - была нажата кнопка OK;

`IDRETRY` - была нажата кнопка Retry (Повторение);

`IDYES` - была нажата кнопка Yes (Да).

Аргументы

`lpszText` - указатель на объект класса `CString` или заканчивающуюся нулем строку, содержащую выводимое в окне сообщение.

`nType` - стиль окна сообщения. Список этих стилей приведен в описании функции `MessageBox`.

`nIDHelp` - идентификатор контекстной справки данного сообщения. Если в данном аргументе передается нулевое значение или значение -1, выводится контекстное сообщение, определенное по умолчанию.

`nIDPrompt` -идентификатор ресурса строки.

Описание

Функция `AfxMessageBox` выводит на экран окно сообщения. Если данное диалоговое окно имеет кнопку Cancel (Отмена), то значение `IDCANCEL` возвращается не только при нажатии данной кнопки, но и при нажатии клавиши . В противном случае нажатие клавиши не приводит ни к каким последствиям. Для форматирования строк в окне сообщения могут использоваться функции `AfxFormatString` и `AfxFormatString2`.

AfxSetAllocHook

AFX_ALLOC_HOOK AfxSetAllocHook(AFX_ALLOC_HOOK pfnAllocHook);

Возвращаемое значение

Ненулевое, если память следует выделить, и нулевое в противном случае.

Аргументы

`pfnAllocHook` - содержит имя вызываемой функции.

Описание

Данная функция задает имя функции обратного вызова, которая будет вызываться перед каждой операцией выделения памяти. Диспетчер памяти MFC позволяет вызывать пользовательскую функцию обратного вызова, проверяющую корректность выполняемой операции выделения памяти. Пользовательская функция обратного вызова должна иметь

следующий формат: BOOL AFXAPI AllocHook(size_t nSize, BOOL bObject, LONG lRequestNumber); где: `nSize` - размер требуемой области памяти.
`bObject` - флаг, устанавливаемый в том случае, если выделяется память под объект, производный от класса `CObject`.
`lRequestNumber` - номер блока в последовательности.
Следует помнить о том, что соглашение о вызовах `AFXAPI` предполагает, что вызываемая функция должна удалить свои аргументы из стека.
AfxTermExtensionModule
void AFXAPI AfxTermExtensionModule(AFX_EXTENSION_MODULE& state, BOOL bAll = FALSE);
Аргументы
`state` - ссылка на объект структуры `AFX_EXTENSION_MODULE`, хранящий состояние модуля библиотеки расширения MFC.
`bAll` - если данный аргумент имеет значение `TRUE`, очищаются все модули библиотек расширения MFC. Если он имеет значение `FALSE`, очищается только текущий модуль.
Описание
Вызов данной функции позволяет очистить библиотеку расширения MFC при завершении работы процесса или отключении библиотек динамической компоновки вызовом функции `AfxFreeLibrary`. Функция `AfxTermExtensionModule` освобождает всю локальную память, выделенную модулю и удаляет все записи из кэша карты сообщений. Эту функцию необходимо вызывать при динамической загрузке и освобождении библиотеки расширения MFC. Поскольку большинство библиотек динамической компоновки не используют динамической загрузки, а подключаются через импортные библиотеки, использовать для них функцию `AfxTermExtensionModule` необязательно.
ChooseFont
BOOL ChooseFont(LPCHOOSEFONT lpcf);
Возвращаемое значение
Если пользователь нажимает кнопку OK в диалоговом окне Шрифт, возвращаемое значение не равно нулю. Выбор пользователя фиксируется в объекте структуры `CHOOSEFONT`. Если пользователь нажимает кнопку Cancel (Отмена) или закрывает диалоговое окно Шрифт (Font), данная функция возвращает нулевое значение. Для получения дополнительной информации используется вызов функции `CommDlgExtendedError`, возвращающей следующие значения: `CDERR_FINDRESFAILURE`, `CDERR_INITIALIZATION`, `CDERR_LOCKRESFAILURE`, `CDERR_LOADRESFAILURE`, `CDERR_LOADSTRFAILURE`, `CDERR_MEMALLOCFAILURE`, `CDERR_MEMLOCKFAILURE`, `CDERR_NOHINSTANCE`, `CDERR_NOHOOK`, `CDERR_NOTEMPLATE`, `CDERR_STRUCTSIZE`, `CFERR_MAXLESSTHANMIN` и `CFERR_NOFONTS`.
Аргументы
`lpcf` - указатель на объект структуры `CHOOSEFONT`, содержащий информацию об установках элементов управления в диалоговом окне.
Описание
Функция `ChooseFont` открывает стандартное диалоговое окно Шрифт (Font), позволяющее пользователю установить атрибуты логического шрифта. Эти атрибуты включают в себя начертание, стиль (жирный, курсив или обычный), размер шрифта, эффекты (подчеркивание, зачеркивание или цвет) и набор символов шрифта. В диалоговом окне Шрифт для обработки сообщений, посылаемых данному окну может быть использован объект класса `CFHookProc`. Для использования данного объекта установите флаг `CF_ENABLEHOOK` в переменной `Flags`, являющейся членом структуры `CHOOSEFONT`, поместите указатель на соответствующую функцию обратного вызова в переменную `lpfnHook` объект структуры `CHOOSEFONT`. Функция обратного вызова

может посылать диалоговому окну сообщения WM_CHOOSEFONT_GETLOGFONT, WM_CHOOSEFONT_SETFLAGS и WM_CHOOSEFONT_SETLOGFONT для получения от него информации о текущих установках его элементов управления.

CloseHandle

BOOL CloseHandle(HANDLE hObject);

Возвращаемое значение

Ненулевое, если функция успешно завершает свою работу, и нулевое в противном случае. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

hObject - дескриптор открытого объекта.

Описание

Функция CloseHandle уничтожает дескриптор открытого объекта. Данная функция уничтожает дескриптор объекта, уменьшает счетчик дескрипторов объекта и производит проверку этого счетчика. Как только значение счетчика становится равным нулю, объект удаляется из системы. Уничтожение дескриптора потока не приводит к завершению связанного с ним потока. Для уничтожения объекта потока необходимо сначала завершить выполнение потока, а затем уничтожить все его дескрипторы. Функция CloseHandle используется для уничтожения дескрипторов, полученных при вызове функции CreateFile. Для уничтожения дескрипторов, возвращаемых функцией FindFirstFile, используйте функцию FindClose. Уничтожение недопустимого дескриптора вызывает исключение. Это происходит, например, при повторном уничтожении дескриптора.

CreateEvent

HANDLE CreateEvent(LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName);

Возвращаемое значение

В случае успешного завершения функции возвращается дескриптор объекта события. Если именованный объект события существовал до вызова данной функции, то функция возвращает дескриптор существующего объекта, а функция GetLastError возвращает значение ERROR_ALREADY_EXISTS. Если в процессе выполнения данной функции возникает ошибка, то возвращается нулевое значение. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

lpEventAttributes - указатель на объект структуры SECURITY_ATTRIBUTES, определяющую, может ли возвращаемый дескриптор наследоваться дочерним процессом. Если данный аргумент имеет нулевое значение, то дескриптор не может наследоваться. В Windows NT переменная lpSecurityDescriptor, являющаяся членом структуры, определяет дескриптор безопасности нового объекта события. Если эта переменная равна нулю, то объект события использует дескриптор безопасности, заданный по умолчанию.

bManualReset - определяет ручной или автоматический сброс объекта события при его создании. Если этот аргумент имеет значение TRUE, то для сброса данного объекта события в неотмеченное состояние необходимо использовать функцию ResetEvent. В противном случае система автоматически сбрасывает состояние данного объекта события в неотмеченное, после освобождения единственного ждущего потока.

bInitialState - определяет начальное состояние объекта события. Если данный аргумент имеет значение TRUE, то данный объект события изначально отмечен.

lpName - указатель на заканчивающуюся нулем строку, содержащую имя объекта события. Длина данного имени ограничивается величиной MAX_PATH и может содержать любые символы кроме символа черты, наклоненной влево, используемой для разделения полей при задании пути к файлу. При сравнении имен учитывается регистр

использованных символов.
Если аргумент lpName совпадает с именем существующего именованного объекта, данная функция запрашивает доступ к данному объекту в режиме EVENT_ALL_ACCESS. В этом случае значения аргументов bManualReset и bInitialState игнорируются.
Если аргумент lpEventAttributes имеет ненулевое значение, он определяет возможность наследования дескриптора, но его дескриптор безопасности игнорируется.
Если аргумент lpName имеет значение NULL, то создается неименованный объект события.
Если аргумент lpName совпадает с именем семафора, мютекса, таймера ожидания, задания или объекта карты файла (file-mapping object), функция прекращает свою работу, а функция GetLastError возвращает значение ERROR_INVALID_HANDLE. Это происходит потому, что эти объекты разделяют одно и то же пространство имен.
Описание
Функция CreateEvent позволяет создавать именованные и неименованные объекты событий. Дескриптор, возвращаемый данной функцией, имеет право доступа EVENT_ALL_ACCESS к новому объекту события и может быть использован любой функцией, имеющей в качестве аргумента дескриптор данного объекта. Любой поток вызывающего процесса может использовать дескриптор объекта события при вызове любой из функций ожидания. Функции ожидания, использующие один объект, возвращают свое значение после того, как будет отмечен соответствующий объект события. Функции ожидания, использующие несколько объектов, могут быть настроены таким образом, чтобы возвращать свое значение после того, как будет отмечен любой из связанных с нею объектов событий, или когда будут отмечены все эти объекты событий. После возвращения значения функцией ожидания ожидающий поток может продолжить свою работу. Начальное состояние объекта события определяется аргументом bInitialState. Для установления объекта события в отмеченное состояние используется функция SetEvent, а для сброса отмеченного состояния используется функция ResetEvent. Отмеченное состояние объекта события, созданного в режиме ручного сброса, остается таковым до вызова пользователем функции ResetEvent. Пока состояние объекта события остается отмеченным может быть возобновлена работа любого количества ждущих потоков или потоков, которые последовательно запускали операцию ожидания с использованием данного объекта события. Отмеченное состояние объекта события, созданного в режиме автоматического сброса, остается таковым до возобновления работы одного ожидающего потока. После этого система автоматически сбрасывает отмеченное состояние объекта события. Если ожидающие потоки отсутствуют, состояние объекта события остается отмеченным. Несколько процессов могут использовать дескрипторы одного объекта события, позволяющего этим процессам осуществлять взаимную синхронизацию. Возможно использование следующих механизмов разделения объектов:
функция CreateProcess создает дочерний процесс, который наследует дескриптор объекта события, если значение аргумента lpEventAttributes в функции CreateEvent допускает это наследование; процесс может передать дескриптор объекта события в аргументе функции DuplicateHandle создающей дубликат дескриптора, который может использоваться другим процессом; процесс может передать имя объекта события в аргументе функций OpenEvent или CreateEvent.
Для уничтожения дескриптора используется функция CloseHandle. Система автоматически уничтожает дескриптор при завершении процесса. Объект события уничтожается при уничтожении его последнего дескриптора.
CreateFontIndirect
HFONT CreateFontIndirect(CONST LOGFONT *lplf);
Возвращаемое значение

В случае успешного завершения функции, дескриптор логического шрифта, и нулевое значение в противном случае. В Windows NT более подробную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`lplf` - указатель на объект структуры `LOGFONT`, содержащий описание логического шрифта.

Описание

Функция `CreateFontIndirect` позволяет создать логический шрифт по его параметрам, заданным в объекте структуры `LOGFONT`. После этого созданный шрифт может выбираться в любой контекст устройства в качестве текущего шрифта. При выборе созданного шрифта в контекст устройства с использованием функции `SelectObject` интерфейс графических устройств Windows старается найти среди физических шрифтов такой, который бы максимально соответствовал параметрам заданного логического шрифта. В случае если ему не удается обеспечить полное соответствие, он использует шрифт, чьи параметры максимально соответствуют требуемым. После того, как приложение закончило работу с данным логическим шрифтом, необходимо вызвать функцию `DeleteObject` для его уничтожения.

DispatchMessage

LRESULT DispatchMessage(CONST MSG * lpmsg);

Возвращаемое значение

Возвращаемое значение совпадает с возвращаемым значением процедуры окна. Его трактовка зависит от обрабатываемого сообщения. Как правило, оно игнорируется.

Аргументы

`lpmsg` - указатель на объект структуры `MSG`, содержащей обрабатываемое сообщение.

Описание

Данная функция передает сообщение процедуре окна. Как правило, это сообщение было извлечено до этого функцией `GetMessage`. Объект структуры `MSG` должен содержать корректные значения сообщения. Если переменная `lpmsg` данного объекта структуры содержит указатель на сообщение `WM_TIMER`, то ее переменная `lParam` содержит указатель на функцию, вызываемую вместо процедуры окна.

EnumFontFamilies

int EnumFontFamilies(HDC hdc, LPCTSTR lpszFamily, FONTENUMPROC lpEnumFontFamProc, LPARAM lParam);

Возвращаемое значение

Значение, возвращенное последней функцией обратного вызова. Трактовка этого значения зависит от контекста, в котором была вызвана данная функция.

Аргументы

`hdc` - дескриптор контекста устройства.

`lpszFamily` - указатель на заканчивающуюся нулем текстовую строку, определяющую имя семейства запрошенных шрифтов. Если аргумент `lpszFamily` имеет нулевое значение, то функция `EnumFontFamilies` случайным образом выбирает и нумерует по одному шрифту для каждого доступного семейства типов шрифтов.

`lpEnumFontFamProc` - определяет адрес экземпляра процедуры определенной в приложении функции обратного вызова. Функции обратного вызова описаны при рассмотрении функции `EnumFontFamProc`.

`lParam` - указатель на блок данных. Структура блока данных определяется приложением. Эти данные передаются функции обратного вызова вместе с информацией о шрифте.

Описание

Функция `EnumFontFamilies` нумерует шрифты в указанном семействе шрифтов, доступном на указанном устройстве. Функции `EnumFontFamilies` и

EnumFontFamProc оставлены для обеспечения совместимости с 16-разрядными версиями Windows. Приложения Win32 должны использовать функцию EnumFontFamiliesEx. Функция EnumFontFamilies позволяет получить информацию по каждому шрифту, семейство которого указано в аргументе lpszFamily, и передать эту информацию функции, указанной в аргументе lpEnumFontFamProc. Определенная в приложении функция обратного вызова может производить любую обработку полученной информации о шрифте. Нумерация шрифтов продолжается до тех пор, пока не останется необработанных шрифтов или пока функция обратного вызова не возвратит нулевое значение.

FreeLibrary

BOOL FreeLibrary(HMODULE hModule);

Возвращаемое значение

В случае успешного завершения работы возвращает ненулевое значение. В противном случае возвращает нулевое значение. Для получения более подробной информации об ошибке вызовите функцию GetLastError.

Аргументы

hModule - дескриптор модуля загруженной библиотеки динамической компоновки, возвращенный функцией LoadLibrary или GetModuleHandle.

Описание

Данная функция уменьшает на единицу счетчик ссылок загруженной библиотеки динамической компоновки. Как только значение этого счетчика достигнет нуля, модуль удаляется из адресного пространства процесса, а его дескриптор становится некорректным. Каждый процесс поддерживает счетчик ссылок для каждого загруженного им модуля библиотеки динамической компоновки. Значение этого счетчика увеличивается на единицу при каждом вызове функции LoadLibrary и уменьшается на единицу при каждом вызове функции FreeLibrary. После загрузки модуля библиотеки динамической компоновки его счетчик ссылок имеет единичное значение. При удалении модуля библиотеки динамической компоновки из адресного пространства процесса система вызывает его функцию DllMain (если таковая имеется) с аргументом DLL_PROCESS_DETACH. Это позволяет библиотеке динамической компоновки освободить все ресурсы, выделенные ей процессом. После завершения работы этой функции библиотечный модуль удаляется из адресного пространства процесса. Поэтому не рекомендуется вызывать данную функцию из функции DllMain. Вызов функции FreeLibrary не оказывает никакого действия на другие процессы, использующие тот же модуль библиотеки динамической компоновки.

GetCurrentDirectory

DWORD GetCurrentDirectory(DWORD nBufferLength, LPTSTR lpBuffer);

Возвращаемое значение

Если функция завершается успешно, возвращается количество символов, записанное в буфер, исключая завершающий строку нулевой символ. В противном случае возвращается нулевое значение. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError. Если текстовый буфер, на который указывает аргумент lpBuffer, имеет недостаточный размер, возвращаемое значение определяет требуемый размер буфера, включая байты, необходимые для размещения завершающего строку нулевого символа.

Аргументы

nBufferLength - определяет размер буфера для хранения строки, содержащей текущий каталог, выраженный в символах. Размер буфера должен учитывать необходимость записи в него завершающего строку нулевого символа.

lpBuffer - указатель на буфер для хранения строки, содержащей текущий каталог. В

данный буфер будет записана тестовая строка, завершающаяся нулевым символом, и содержащая абсолютный путь к текущему каталогу.
Описание
Функция `GetCurrentDirectory` позволяет получить путь к текущему каталогу данного процесса.
GetDlgItem
HWND GetDlgItem(HWND hDlg, int nIDDlgItem);
Возвращаемое значение
В случае успешного завершения функции возвращается дескриптор окна указанного элемента управления. В противном случае возвращается нулевое значение, указывающее на то, что в качестве аргументов данной функции были использованы дескриптор несуществующего диалогового окна или идентификатор несуществующего элемента управления. Дополнительную информацию по ошибке можно получить, вызвав функцию `GetLastError`.
Аргументы
`hDlg` - дескриптор диалогового окна, содержащего данный элемент управления.
`nIDDlgItem` - идентификатор элемента управления, дескриптор которого требуется получить.
Описание
Функция `GetDlgItem` позволяет получить дескриптор окна элемента управления в указанном диалоговом окне. Данная функция может использоваться для любой пары родительского и диалогового окна, а не только в диалоговых окнах. В тех случаях, когда аргумент `hDlg` определяет родительское окно, а дочернее окно имеет уникальный идентификатор (определяемый аргументом `hMenu` в функциях `CreateWindow` или `CreateWindowEx` при создании дочернего окна), функция `GetDlgItem` возвращает корректный дескриптор соответствующего дочернего окна.
GetExitCodeThread
BOOL GetExitCodeThread(HANDLE hThread, LPDWORD lpExitCode);
Возвращаемое значение
Ненулевое, если функция успешно завершает свою работу, и нулевое в противном случае. Дополнительную информацию по ошибке можно получить, вызвав функцию `GetLastError`.
Аргументы
`hThread` - дескриптор потока. В Windows NT дескриптор должен иметь уровень доступа `THREAD_QUERY_INFORMATION`.
`lpExitCode` - указатель на 32-разрядную величину, в которую будет записан код завершения потока.
Описание
Функция `GetExitCodeThread` позволяет получить код завершения указанного потока. Если указанный поток еще не завершил свою работу, код завершения имеет значение `STILL_ACTIVE`. Если поток завершил свою работу, то его код завершения может иметь следующие значения:
код завершения, указанный в качестве аргумента функции `ExitThread` или `TerminateThread;`. возвращаемое значение исполняющей функции потока; код завершения процесса потока.
GetLastError
DWORD GetLastError(VOID)
Возвращаемое значение
Данная функция возвращает значение кода последней ошибки, возникшей в вызывающем потоке. Функция устанавливает это значение с помощью вызова функции `SetLastError`. Раздел Возвращаемое значение справки по каждой функции содержит

условия, при которых данная функция устанавливает код последней ошибки. Поскольку функция `SetLastError` является исключительно 32-разрядной функцией, функции Win32, представляющие собой скрытый вызов 16-разрядных функций, не устанавливают код последней ошибки. В этих функциях необходимо игнорировать эту величину. К подобным функциям относятся функции работы с окнами, функции GDI и функции работы с устройствами мультимедиа.

Описание

Функция `GetLastError` возвращает значение кода последней ошибки, возникшей в вызывающем потоке. Каждый поток имеет свой код последней ошибки. Вызов функции `GetLastError` должен следовать немедленно после возврата любой функцией значения, свидетельствующего о том, что при ее выполнении произошла ошибка. Это необходимо потому, что некоторые функции вызывают функцию `SetLastError(0)` в случае своего успешного завершения, что приводит к уничтожению кода ошибки, установленного предыдущей функцией. Большинство функций Win32 API устанавливают код последней ошибки в случае возникновения ошибки, но некоторые устанавливают его в случае своего нормального завершения. Ошибка при выполнении функции индицируется, обычно следующими кодами ошибки: `FALSE`, `NULL`, `0xFFFFFFFF` или `-1`. Все случаи, когда функция вызывает функцию `SetLastError` при своем нормальном завершении, отмечены в справке по этой функции. Коды ошибок представляют собой 32-разрядные величины (бит 31 является старшим битом). Бит 29 зарезервирован для кодов ошибок, определяемых пользователем, поэтому при определении собственного кода ошибки пользователь должен установить этот бит в единицу. Чтобы получить строку, описывающую ошибку, по ее коду нужно вызвать функцию `FormatMessage`.

GetMenuContextHelpId

DWORD GetMenuContextHelpId(HMENU hmenu);

Возвращаемое значение

Возвращает контекстный идентификатор справки, если такой идентификатор связан с данным меню, и ноль в противном случае.

Аргументы

`hmenu` - дескриптор меню, для которого необходимо получить контекстный идентификатор справки.

Описание

Позволяет получить контекстный идентификатор справки для указанного меню.

AfxGetApp

CWinApp* AfxGetApp();

Возвращаемое значение

Указатель на единственный объект класса `CWinApp` данного приложения.

Описание

Указатель, возвращаемый данной функцией, может быть использован для доступа к информации о данном приложении, такой, как главная процедура обработки сообщения или объект главного окна программы.

AfxGetResourceHandle

HINSTANCE AfxGetResourceHandle();

Возвращаемое значение

Дескриптор `HINSTANCE` ресурсов, принадлежащих данному приложению по умолчанию.

Описание

Возвращаемый данной функцией дескриптор может быть использован для непосредственного доступа к ресурсам приложения, например, при вызове функции Windows `FindResource`.

AfxInitExtensionModule

BOOL AFXAPI AfxInitExtensionModule(AFX_EXTENSION_MODULE& state, HMODULE hModule);

Возвращаемое значение

Если инициализация библиотеки динамической компоновки прошла успешно, возвращает значение TRUE. В противном случае возвращает значение FALSE.

Аргументы

`state` - ссылка на объект структуры AFX_EXTENSION_MODULE, в котором будет сохранено состояние библиотеки расширения MFC после ее инициализации. Помимо прочего, в ней содержится состояние объектов классов, инициализированных статическими конструкторами до вызова функции `DllMain`.

`hModule` - дескриптор модуля библиотеки расширения MFC.

Описание

Данная функция вызывается в функции `DllMain` для инициализации библиотеки расширения MFC. Функция `AfxInitExtensionModule` копирует `HMODULE` библиотеки динамической компоновки и сохраняет объекты структуры `CRuntimeClass` и фабрики объектов (объекты `COleObjectFactory`) для последующего использования при создании объекта `CDynLinkLibrary`. В функции `DllMain` библиотеки расширения MFC необходимо произвести две операции:

вызвать функцию `AfxInitExtensionModule` и проверить возвращаемое ею значение;

создать объект класса `CDynLinkLibrary`, если библиотека динамической компоновки экспортирует объекты `CRuntimeClass` или имеет собственные пользовательские ресурсы.

При завершении работы процесса или отключении библиотек динамической компоновки вызовом функции `AfxFreeLibrary` можно вызвать функцию `AfxTermExtensionModule` для очистки библиотеки расширения MFC.

AfxIsMemoryBlock

BOOL AfxIsMemoryBlock(const void* p, UINT nBytes, LONG* plRequestNumber = NULL);

Возвращаемое значение

Ненулевое, если блок памяти выделен и имеет указанный размер, и нулевое в противном случае.

Аргументы

`p` - указатель на проверяемый блок памяти.

`nBytes` - размер проверяемого блока в байтах.

`plRequestNumber` - указатель на переменную типа `long`, в которую будет записан номер блока в последовательности. Эта переменная заполняется только в том случае, если данная функция возвратила ненулевое значение.

Описание

Данная функция проверяет адрес в памяти, чтобы убедиться в том, что он соответствует активному блоку памяти, выделенному диагностической версией операции `new`. Кроме того, производится проверка соответствия указанного размера блока его истинным размерам. Если функция `AfxIsMemoryBlock` возвращает ненулевое значение, в переменную, на которую указывает аргумент `plRequestNumber`, записывается номер блока в последовательности, представляющий собой порядковый номер вызова операции `new` при создании данного блока.

AfxIsValidAddress

BOOL AfxIsValidAddress(const void* lp, UINT nBytes, BOOL bReadWrite = TRUE);

Возвращаемое значение

Ненулевое, если указанный блок памяти полностью расположен в области памяти, выделенной данному приложению, и нулевое в противном случае.

Аргументы

lp - указатель на проверяемый адрес в оперативной памяти.
nBytes - содержит размер проверяемой области памяти.
bReadWrite - определяет режим доступа к памяти: по чтению и записи (TRUE) или только для чтения (FALSE).
Описание
Данная функция проверяет указанный блок, чтобы убедиться в том, что он полностью расположен в области памяти, выделенной данному приложению. Эта проверка не ограничивается блоками памяти, выделенными операцией new.
AfxIsValidString
BOOL AfxIsValidString(LPCSTR lpsz, int nLength = -1);
Возвращаемое значение
Ненулевое, если передан указатель на строку указанного размера, и нулевое в противном случае.
Аргументы
lpsz - проверяемый указатель.
nLength - длина проверяемой строки в байтах. Значение -1 указывает на то, что строка завершается нулем.
Описание
Данная функция используется для определения того, что переданный ей указатель указывает на корректную строку.
AfxMessageBox
int AfxMessageBox(LPCTSTR lpszText, UINT nType = MB_OK, UINT nIDHelp = 0); int AFXAPI AfxMessageBox(UINT nIDPrompt, UINT nType = MB_OK, UINT nIDHelp = (UINT) -1);
Возвращаемое значение
Если для вывода диалогового окна недостаточно памяти, возвращает нулевое значение. В противном случае возвращает одно из следующих значений:
IDABORT - была нажата кнопка Abort (Прекращение);
IDCANCEL - была нажата кнопка Cancel (Отмена);
IDIGNORE - была нажата кнопка Ignore (Пропуск);
IDNO - была нажата кнопка No (Нет);
IDOK - была нажата кнопка OK;
IDRETRY - была нажата кнопка Retry (Повторение);
IDYES - была нажата кнопка Yes (Да).
Аргументы
lpszText - указатель на объект класса CString или заканчивающуюся нулем строку, содержащую выводимое в окне сообщение.
nType - стиль окна сообщения. Список этих стилей приведен в описании функции MessageBox.
nIDHelp - идентификатор контекстной справки данного сообщения. Если в данном аргументе передается нулевое значение или значение -1, выводится контекстное сообщение, определенное по умолчанию.
nIDPrompt -идентификатор ресурса строки.
Описание
Функция AfxMessageBox выводит на экран окно сообщения. Если данное диалоговое окно имеет кнопку Cancel (Отмена), то значение IDCANCEL возвращается не только при нажатии данной кнопки, но и при нажатии клавиши . В противном случае нажатие клавиши не приводит ни к каким последствиям. Для форматирования строк в окне сообщения могут использоваться функции AfxFormatString и AfxFormatString2.
AfxSetAllocHook
AFX_ALLOC_HOOK AfxSetAllocHook(AFX_ALLOC_HOOK pfnAllocHook);

Возвращаемое значение
Ненулевое, если память следует выделить, и нулевое в противном случае.
Аргументы
`pfnAllocHook` - содержит имя вызываемой функции.
Описание
Данная функция задает имя функции обратного вызова, которая будет вызываться перед каждой операцией выделения памяти. Диспетчер памяти MFC позволяет вызывать пользовательскую функцию обратного вызова, проверяющую корректность выполняемой операции выделения памяти. Пользовательская функция обратного вызова должна иметь следующий формат: **BOOL AFXAPI AllocHook(size_t nSize, BOOL bObject, LONG lRequestNumber);** где: `nSize` - размер требуемой области памяти.
`bObject` - флаг, устанавливаемый в том случае, если выделяется память под объект, производный от класса `CObject`.
`lRequestNumber` - номер блока в последовательности.
Следует помнить о том, что соглашение о вызовах `AFXAPI` предполагает, что вызываемая функция должна удалить свои аргументы из стека.
AfxTermExtensionModule
void AFXAPI AfxTermExtensionModule(AFX_EXTENSION_MODULE& state, BOOL bAll = FALSE);
Аргументы
`state` - ссылка на объект структуры `AFX_EXTENSION_MODULE`, хранящий состояние модуля библиотеки расширения MFC.
`bAll` - если данный аргумент имеет значение `TRUE`, очищаются все модули библиотек расширения MFC. Если он имеет значение `FALSE`, очищается только текущий модуль.
Описание
Вызов данной функции позволяет очистить библиотеку расширения MFC при завершении работы процесса или отключении библиотек динамической компоновки вызовом функции `AfxFreeLibrary`. Функция `AfxTermExtensionModule` освобождает всю локальную память, выделенную модулю и удаляет все записи из кэша карты сообщений. Эту функцию необходимо вызывать при динамической загрузке и освобождении библиотеки расширения MFC. Поскольку большинство библиотек динамической компоновки не используют динамической загрузки, а подключаются через импортные библиотеки, использовать для них функцию `AfxTermExtensionModule` необязательно.
ChooseFont
BOOL ChooseFont(LPCHOOSEFONT lpcf);
Возвращаемое значение
Если пользователь нажимает кнопку OK в диалоговом окне Шрифт, возвращаемое значение не равно нулю. Выбор пользователя фиксируется в объекте структуры `CHOOSEFONT`. Если пользователь нажимает кнопку Cancel (Отмена) или закрывает диалоговое окно Шрифт (Font), данная функция возвращает нулевое значение. Для получения дополнительной информации используется вызов функции `CommDlgExtendedError`, возвращающей следующие значения: `CDERR_FINDRESFAILURE`, `CDERR_INITIALIZATION`, `CDERR_LOCKRESFAILURE`, `CDERR_LOADRESFAILURE`, `CDERR_LOADSTRFAILURE`, `CDERR_MEMALLOCFAILURE`, `CDERR_MEMLOCKFAILURE`, `CDERR_NOHINSTANCE`, `CDERR_NOHOOK`, `CDERR_NOTEMPLATE`, `CDERR_STRUCTSIZE`, `CFERR_MAXLESSTHANMIN` и `CFERR_NOFONTS`.
Аргументы
`lpcf` - указатель на объект структуры `CHOOSEFONT`, содержащий информацию об установках элементов управления в диалоговом окне.
Описание

Функция ChooseFont открывает стандартное диалоговое окно Шрифт (Font), позволяющее пользователю установить атрибуты логического шрифта. Эти атрибуты включают в себя начертание, стиль (жирный, курсив или обычный), размер шрифта, эффекты (подчеркивание, зачеркивание или цвет) и набор символов шрифта. В диалоговом окне Шрифт для обработки сообщений, посылаемых данному окну может быть использован объект класса CFHookProc. Для использования данного объекта установите флаг CF_ENABLEHOOK в переменной Flags, являющейся членом структуры CHOOSEFONT, поместите указатель на соответствующую функцию обратного вызова в переменную lpfnHook объект структуры CHOOSEFONT. Функция обратного вызова может посылать диалоговому окну сообщения WM_CHOOSEFONT_GETLOGFONT, WM_CHOOSEFONT_SETFLAGS и WM_CHOOSEFONT_SETLOGFONT для получения от него информации о текущих установках его элементов управления.

CloseHandle

BOOL CloseHandle(HANDLE hObject);

Возвращаемое значение

Ненулевое, если функция успешно завершает свою работу, и нулевое в противном случае. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

hObject - дескриптор открытого объекта.

Описание

Функция CloseHandle уничтожает дескриптор открытого объекта. Данная функция уничтожает дескриптор объекта, уменьшает счетчик дескрипторов объекта и производит проверку этого счетчика. Как только значение счетчика становится равным нулю, объект удаляется из системы. Уничтожение дескриптора потока не приводит к завершению связанного с ним потока. Для уничтожения объекта потока необходимо сначала завершить выполнение потока, а затем уничтожить все его дескрипторы. Функция CloseHandle используется для уничтожения дескрипторов, полученных при вызове функции CreateFile. Для уничтожения дескрипторов, возвращаемых функцией FindFirstFile, используйте функцию FindClose. Уничтожение недопустимого дескриптора вызывает исключение. Это происходит, например, при повторном уничтожении дескриптора.

CreateEvent

HANDLE CreateEvent(LPSECURITY_ATTRIBUTES lpEventAttributes, BOOL bManualReset, BOOL bInitialState, LPCTSTR lpName);

Возвращаемое значение

В случае успешного завершения функции возвращается дескриптор объекта события. Если именованный объект события существовал до вызова данной функции, то функция возвращает дескриптор существующего объекта, а функция GetLastError возвращает значение ERROR_ALREADY_EXISTS. Если в процессе выполнения данной функции возникает ошибка, то возвращается нулевое значение. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

lpEventAttributes - указатель на объект структуры SECURITY_ATTRIBUTES, определяющую, может ли возвращаемый дескриптор наследоваться дочерним процессом. Если данный аргумент имеет нулевое значение, то дескриптор не может наследоваться. В Windows NT переменная lpSecurityDescriptor, являющаяся членом структуры, определяет дескриптор безопасности нового объекта события. Если эта переменная равна нулю, то объект события использует дескриптор безопасности, заданный по умолчанию.

bManualReset - определяет ручной или автоматический сброс объекта события при его создании. Если этот аргумент имеет значение TRUE, то для сброса данного объекта

события в неотмеченное состояние необходимо использовать функцию ResetEvent. В противном случае система автоматически сбрасывает состояние данного объекта события в неотмеченное, после освобождения единственного ждущего потока.
bInitialState - определяет начальное состояние объекта события. Если данный аргумент имеет значение TRUE, то данный объект события изначально отмечен.
lpName - указатель на заканчивающуюся нулем строку, содержащую имя объекта события. Длина данного имени ограничивается величиной MAX_PATH и может содержать любые символы кроме символа черты, наклоненной влево, используемой для разделения полей при задании пути к файлу. При сравнении имен учитывается регистр использованных символов.
Если аргумент lpName совпадает с именем существующего именованного объекта, данная функция запрашивает доступ к данному объекту в режиме EVENT_ALL_ACCESS. В этом случае значения аргументов bManualReset и bInitialState игнорируются.
Если аргумент lpEventAttributes имеет ненулевое значение, он определяет возможность наследования дескриптора, но его дескриптор безопасности игнорируется.
Если аргумент lpName имеет значение NULL, то создается неименованный объект события.
Если аргумент lpName совпадает с именем семафора, мютекса, таймера ожидания, задания или объекта карты файла (file-mapping object), функция прекращает свою работу, а функция GetLastError возвращает значение ERROR_INVALID_HANDLE. Это происходит потому, что эти объекты разделяют одно и то же пространство имен.
Описание
Функция CreateEvent позволяет создавать именованные и неименованные объекты событий. Дескриптор, возвращаемый данной функцией, имеет право доступа EVENT_ALL_ACCESS к новому объекту события и может быть использован любой функцией, имеющей в качестве аргумента дескриптор данного объекта. Любой поток вызывающего процесса может использовать дескриптор объекта события при вызове любой из функций ожидания. Функции ожидания, использующие один объект, возвращают свое значение после того, как будет отмечен соответствующий объект события. Функции ожидания, использующие несколько объектов, могут быть настроены таким образом, чтобы возвращать свое значение после того, как будет отмечен любой из связанных с нею объектов событий, или когда будут отмечены все эти объекты событий. После возвращения значения функцией ожидания ожидающий поток может продолжить свою работу. Начальное состояние объекта события определяется аргументом bInitialState. Для установления объекта события в отмеченное состояние используется функция SetEvent, а для сброса отмеченного состояния используется функция ResetEvent. Отмеченное состояние объекта события, созданного в режиме ручного сброса, остается таковым до вызова пользователем функции ResetEvent. Пока состояние объекта события остается отмеченным может быть возобновлена работа любого количества ждущих потоков или потоков, которые последовательно запускали операцию ожидания с использованием данного объекта события. Отмеченное состояние объекта события, созданного в режиме автоматического сброса, остается таковым до возобновления работы одного ожидающего потока. После этого система автоматически сбрасывает отмеченное состояние объекта события. Если ожидающие потоки отсутствуют, состояние объекта события остается отмеченным. Несколько процессов могут использовать дескрипторы одного объекта события, позволяющего этим процессам осуществлять взаимную синхронизацию. Возможно использование следующих механизмов разделения объектов:
функция CreateProcess создает дочерний процесс, который наследует дескриптор объекта события, если значение аргумента lpEventAttributes в функции CreateEvent допускает это наследование; процесс может передать дескриптор объекта

события в аргументе функции DuplicateHandle создающей дубликат дескриптора, который может использоваться другим процессом; процесс может передать имя объекта события в аргументе функций OpenEvent или CreateEvent.
Для уничтожения дескриптора используется функция CloseHandle. Система автоматически уничтожает дескриптор при завершении процесса. Объект события уничтожается при уничтожении его последнего дескриптора.
CreateFontIndirect
HFONT CreateFontIndirect(CONST LOGFONT *lplf);
Возвращаемое значение
В случае успешного завершения функции, дескриптор логического шрифта, и нулевое значение в противном случае. В Windows NT более подробную информацию об ошибке можно получить, вызвав функцию GetLastError.
Аргументы
lplf - указатель на объект структуры LOGFONT, содержащий описание логического шрифта.
Описание
Функция CreateFontIndirect позволяет создать логический шрифт по его параметрам, заданным в объекте структуры LOGFONT. После этого созданный шрифт может выбираться в любой контекст устройства в качестве текущего шрифта. При выборе созданного шрифта в контекст устройства с использованием функции SelectObject интерфейс графических устройств Windows старается найти среди физических шрифтов такой, который бы максимально соответствовал параметрам заданного логического шрифта. В случае если ему не удается обеспечить полное соответствие, он использует шрифт, чьи параметры максимально соответствуют требуемым. После того, как приложение закончило работу с данным логическим шрифтом, необходимо вызвать функцию DeleteObject для его уничтожения.
DispatchMessage
LRESULT DispatchMessage(CONST MSG * lpmsg);
Возвращаемое значение
Возвращаемое значение совпадает с возвращаемым значением процедуры окна. Его трактовка зависит от обрабатываемого сообщения. Как правило, оно игнорируется.
Аргументы
lpmsg - указатель на объект структуры MSG, содержащей обрабатываемое сообщение.
Описание
Данная функция передает сообщение процедуре окна. Как правило, это сообщение было извлечено до этого функцией GetMessage. Объект структуры MSG должен содержать корректные значения сообщения. Если переменная lpmsg данного объекта структуры содержит указатель на сообщение WM_TIMER, то ее переменная lParam содержит указатель на функцию, вызываемую вместо процедуры окна.

EnumFontFamilies
int EnumFontFamilies(HDC hdc, LPCTSTR lpszFamily, FONTENUMPROC lpEnumFontFamProc, LPARAM lParam);
Возвращаемое значение
Значение, возвращенное последней функцией обратного вызова. Трактовка этого значения зависит от контекста, в котором была вызвана данная функция.
Аргументы
hdc - дескриптор контекста устройства.
lpszFamily - указатель на заканчивающуюся нулем текстовую строку, определяющую имя семейства запрошенных шрифтов. Если аргумент lpszFamily имеет нулевое значение, то функция EnumFontFamilies случайным образом выбирает и нумерует по

одному шрифту для каждого доступного семейства типов шрифтов.
lpEnumFontFamProc - определяет адрес экземпляра процедуры определенной в приложении функции обратного вызова. Функции обратного вызова описаны при рассмотрении функции EnumFontFamProc.
lParam - указатель на блок данных. Структура блока данных определяется приложением. Эти данные передаются функции обратного вызова вместе с информацией о шрифте.
Описание
Функция EnumFontFamilies нумерует шрифты в указанном семействе шрифтов, доступном на указанном устройстве. Функции EnumFontFamilies и EnumFontFamProc оставлены для обеспечения совместимости с 16-разрядными версиями Windows. Приложения Win32 должны использовать функцию EnumFontFamiliesEx. Функция EnumFontFamilies позволяет получить информацию по каждому шрифту, семейство которого указано в аргументе lpszFamily, и передать эту информацию функции, указанной в аргументе lpEnumFontFamProc. Определенная в приложении функция обратного вызова может производить любую обработку полученной информации о шрифте. Нумерация шрифтов продолжается до тех пор, пока не останется необработанных шрифтов или пока функция обратного вызова не возвратит нулевое значение.
FreeLibrary
BOOL FreeLibrary(HMODULE hModule);
Возвращаемое значение
В случае успешного завершения работы возвращает ненулевое значение. В противном случае возвращает нулевое значение. Для получения более подробной информации об ошибке вызовите функцию GetLastError.
Аргументы
hModule - дескриптор модуля загруженной библиотеки динамической компоновки, возвращенный функцией LoadLibrary или GetModuleHandle.
Описание
Данная функция уменьшает на единицу счетчик ссылок загруженной библиотеки динамической компоновки. Как только значение этого счетчика достигнет нуля, модуль удаляется из адресного пространства процесса, а его дескриптор становится некорректным. Каждый процесс поддерживает счетчик ссылок для каждого загруженного им модуля библиотеки динамической компоновки. Значение этого счетчика увеличивается на единицу при каждом вызове функции LoadLibrary и уменьшается на единицу при каждом вызове функции FreeLibrary. После загрузки модуля библиотеки динамической компоновки его счетчик ссылок имеет единичное значение. При удалении модуля библиотеки динамической компоновки из адресного пространства процесса система вызывает его функцию DllMain (если таковая имеется) с аргументом DLL_PROCESS_DETACH. Это позволяет библиотеке динамической компоновки освободить все ресурсы, выделенные ей процессом. После завершения работы этой функции библиотечный модуль удаляется из адресного пространства процесса. Поэтому не рекомендуется вызывать данную функцию из функции DllMain. Вызов функции FreeLibrary не оказывает никакого действия на другие процессы, использующие тот же модуль библиотеки динамической компоновки.
GetCurrentDirectory
DWORD GetCurrentDirectory(DWORD nBufferLength, LPTSTR lpBuffer);
Возвращаемое значение
Если функция завершается успешно, возвращается количество символов, записанное в буфер, исключая завершающий строку нулевой символ. В противном случае возвращается нулевое значение. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError. Если текстовый буфер, на который указывает аргумент

lpBuffer, имеет недостаточный размер, возвращаемое значение определяет требуемый размер буфера, включая байты, необходимые для размещения завершающего строку нулевого символа.

Аргументы

nBufferLength - определяет размер буфера для хранения строки, содержащей текущий каталог, выраженный в символах. Размер буфера должен учитывать необходимость записи в него завершающего строку нулевого символа.

lpBuffer - указатель на буфер для хранения строки, содержащей текущий каталог. В данный буфер будет записана тестовая строка, завершающаяся нулевым символом, и содержащая абсолютный путь к текущему каталогу.

Описание

Функция GetCurrentDirectory позволяет получить путь к текущему каталогу данного процесса.

GetDlgItem

HWND GetDlgItem(HWND hDlg, int nIDDlgItem);

Возвращаемое значение

В случае успешного завершения функции возвращается дескриптор окна указанного элемента управления. В противном случае возвращается нулевое значение, указывающее на то, что в качестве аргументов данной функции были использованы дескриптор несуществующего диалогового окна или идентификатор несуществующего элемента управления. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

hDlg - дескриптор диалогового окна, содержащего данный элемент управления.

nIDDlgItem - идентификатор элемента управления, дескриптор которого требуется получить.

Описание

Функция GetDlgItem позволяет получить дескриптор окна элемента управления в указанном диалоговом окне. Данная функция может использоваться для любой пары родительского и диалогового окна, а не только в диалоговых окнах. В тех случаях, когда аргумент hDlg определяет родительское окно, а дочернее окно имеет уникальный идентификатор (определяемый аргументом hMenu в функциях CreateWindow или CreateWindowEx при создании дочернего окна), функция GetDlgItem возвращает корректный дескриптор соответствующего дочернего окна.

GetExitCodeThread

BOOL GetExitCodeThread(HANDLE hThread, LPDWORD lpExitCode);

Возвращаемое значение

Ненулевое, если функция успешно завершает свою работу, и нулевое в противном случае. Дополнительную информацию по ошибке можно получить, вызвав функцию GetLastError.

Аргументы

hThread - дескриптор потока. В Windows NT дескриптор должен иметь уровень доступа THREAD_QUERY_INFORMATION.

lpExitCode - указатель на 32-разрядную величину, в которую будет записан код завершения потока.

Описание

Функция GetExitCodeThread позволяет получить код завершения указанного потока. Если указанный поток еще не завершил свою работу, код завершения имеет значение STILL_ACTIVE. Если поток завершил свою работу, то его код завершения может иметь следующие значения:

код завершения, указанный в качестве аргумента функции `ExitThread` или `TerminateThread;`. возвращаемое значение исполняющей функции потока; код завершения процесса потока.

GetLastError

DWORD GetLastError(VOID)

Возвращаемое значение

Данная функция возвращает значение кода последней ошибки, возникшей в вызывающем потоке. Функция устанавливает это значение с помощью вызова функции `SetLastError`. Раздел Возвращаемое значение справки по каждой функции содержит условия, при которых данная функция устанавливает код последней ошибки. Поскольку функция `SetLastError` является исключительно 32-разрядной функцией, функции Win32, представляющие собой скрытый вызов 16-разрядных функций, не устанавливают код последней ошибки. В этих функциях необходимо игнорировать эту величину. К подобным функциям относятся функции работы с окнами, функции GDI и функции работы с устройствами мультимедиа.

Описание

Функция `GetLastError` возвращает значение кода последней ошибки, возникшей в вызывающем потоке. Каждый поток имеет свой код последней ошибки. Вызов функции `GetLastError` должен следовать немедленно после возврата любой функцией значения, свидетельствующего о том, что при ее выполнении произошла ошибка. Это необходимо потому, что некоторые функции вызывают функцию `SetLastError(0)` в случае своего успешного завершения, что приводит к уничтожению кода ошибки, установленного предыдущей функцией. Большинство функций Win32 API устанавливают код последней ошибки в случае возникновения ошибки, но некоторые устанавливают его в случае своего нормального завершения. Ошибка при выполнении функции индицируется, обычно следующими кодами ошибки: `FALSE`, `NULL`, `0xFFFFFFFF` или `-1`. Все случаи, когда функция вызывает функцию `SetLastError` при своем нормальном завершении, отмечены в справке по этой функции. Коды ошибок представляют собой 32-разрядные величины (бит 31 является старшим битом). Бит 29 зарезервирован для кодов ошибок, определяемых пользователем, поэтому при определении собственного кода ошибки пользователь должен установить этот бит в единицу. Чтобы получить строку, описывающую ошибку, по ее коду нужно вызвать функцию `FormatMessage`.

GetMenuContextHelpId

DWORD GetMenuContextHelpId(HMENU hmenu);

Возвращаемое значение

Возвращает контекстный идентификатор справки, если такой идентификатор связан с данным меню, и ноль в противном случае.

Аргументы

`hmenu` - дескриптор меню, для которого необходимо получить контекстный идентификатор справки.

Описание

Позволяет получить контекстный идентификатор справки для указанного меню.

LoadLibrary

HMODULE LoadLibrary(LPCTSTR lpFileName);

Возвращаемое значение

В случае успешного завершения работы возвращает дескриптор модуля. В противном случае возвращает нулевое значение. Для получения дополнительной информации по ошибке вызовите функцию `GetLastError`. В Windows 95 ошибки при вызове данной функции возникают в следующих случаях:

при попытке использовать данную функцию для загрузки модуля, содержащего ресурсы с идентификаторами, значения которых превышают 0x7FFF; при попытке непосредственной загрузки 16-разрядных библиотек динамической компоновки в 32-разрядные приложения; при загрузке библиотек динамической компоновки с версией подсистемы большей, чем 4.0. при попытке вызывать в функции DllMain версии Unicode для функции Win32.

Аргументы

lpFileName - указатель на заканчивающуюся нулем строку, содержащую имя исполняемого модуля (файла .dll или .exe). Указанное имя является исключительно именем файла и не имеет никакого отношения к имени, указанном в файле определения модуля (.def) в выражении LIBRARY.

Описание

Данная функция помещает указанный исполняемый модуль в адресное пространство вызывающего процесса. Если указанный модуль еще не располагается в адресном пространстве вызывающего процесса, система вызывает функцию DllMain данной библиотеки динамической компоновки с аргументом DLL_PROCESS_ATTACH. Если эта функция не возвращает значения TRUE, функция LoadLibrary аварийно завершает свою работу и возвращает нулевое значение. В этом случае система немедленно вызывает ту же самую функцию с аргументом DLL_PROCESS_DETACH для удаления модуля. Поэтому функцию LoadLibrary не рекомендуется вызывать в функции DllMain. Возвращаемый данной функцией дескриптор модуля не является глобальным и не наследуется. Поэтому этот дескриптор не может использоваться другим процессом. Если файл отсутствует в указанном каталоге, функция аварийно завершает свою работу. При задании каталога следует использовать символ \, а не символ /. Если в файле не указано расширение, по умолчанию к его имени добавляется расширение .dll. Для указания того, что данный файл не имеет расширения необходимо закончить его имя точкой. Если файл указан без пути, то при его поиске используется стандартная стратегия поиска файлов: текущий каталог; системный каталог Windows, путь в который может быть получен с использованием функции GetSystemDirectory; каталог Windows, путь в который может быть получен с использованием функции GetWindowsDirectory; каталоги, перечисленные в переменной PATH.

В Windows NT/ 2000 после поиска в системном каталоге 32-разрядной Windows, путь в который может быть получен с использованием функции GetSystemDirectory и имеющий имя SYSTEM32, производится поиск в системном директории 16-разрядной Windows, путь в который не может быть получен с использованием какой-либо функции, и имеющий имя SYSTEM.

MessageBox

int MessageBox(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType);

Возвращаемое значение

Нулевое, если в системе недостаточно оперативной памяти для создания окна сообщения. В случае успешного завершения функции возвращаемая величина может принимать одно из перечисленных ниже значений:

IDABORT - была нажата кнопка Abort (Прекращение);

IDCANCEL - была нажата кнопка Cancel (Отмена);

IDIGNORE - была нажата кнопка Ignore (Пропуск);

IDNO - была нажата кнопка No (Нет);

IDOK - была нажата кнопка OK;

IDRETRY - была нажата кнопка Retry (Повторение);

IDYES - была нажата кнопка Yes (Да).

Если окно сообщения имеет кнопку Cancel (Отмена), функция возвращает значение IDCANCEL независимо от того, была ли нажата кнопка Cancel (Отмена) или клавиша .

Если окно сообщения не имеет кнопки Cancel (Отмена), то нажатие клавиши не приводит ни к каким результатам.

Аргументы

hWnd - определяет окно, являющееся собственником создаваемого окна сообщения. Если этот аргумент имеет значение NULL, то создаваемое окно сообщения не имеет собственника.

lpText - указатель на текстовую строку, завершающуюся нулевым символом, содержащую выводимое сообщение.

lpCaption - указатель на текстовую строку, завершающуюся нулевым символом, используемую в качестве заголовка диалогового окна. Если этот аргумент имеет значение NULL, то создаваемое окно содержит стандартный заголовок Error (Ошибка).

uType - определяет комбинацию битовых флагов, определяющих содержимое и поведение диалогового окна. Этот аргумент может представлять собой комбинацию флагов из приведенных ниже групп (не более одного из каждой группы).

Флаги, определяющие состав кнопок окна сообщения:

MB_ABORTRETRYIGNORE - окно сообщения содержит три кнопки: Abort (Прекращение), Retry (Повторение) и Ignore (Пропуск);

MB_OK - окно сообщения содержит одну кнопку OK. Это установка по умолчанию;

MB_OKCANCEL - окно сообщения содержит две кнопки: OK и Cancel (Отмена).

MB_RETRYCANCEL - окно сообщения содержит две кнопки: Retry (Повторение) и Cancel (Отмена);

MB_YESNO - окно сообщения содержит две кнопки: Yes (Да) и No (Нет);

MB_YESNOCANCEL - окно сообщения содержит три кнопки: Yes (Да), No (Нет) и Cancel (Отмена).

Флаги, определяющие значок, выводимый в окне сообщения:

MB_ICONEXCLAMATION, MB_ICONWARNING - в окне сообщения появляется восклицательный знак;

MB_ICONINFORMATION, MB_ICONASTERISK - в окне сообщения появляется прописная буква i, помещенная в кружок;

MB_ICONQUESTION - в окне сообщения появляется вопросительный знак;

MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND - в окне сообщения появляется сигнал остановки.

Флаги, задающие кнопку, используемую по умолчанию.

MB_DEFBUTTON1 - по умолчанию используется первая кнопка. Если в данном аргументе не установлен флаг этой группы, то кнопкой по умолчанию является первая кнопка;

MB_DEFBUTTON2 - по умолчанию используется вторая кнопка;

MB_DEFBUTTON3 - по умолчанию используется третья кнопка;

MB_DEFBUTTON4 - по умолчанию используется четвертая кнопка.

Флаги, определяющие модальность диалогового окна.

MB_APPLMODAL - указывает на то, что пользователь должен закончить работу с окном сообщения прежде, чем он получит возможность продолжить работу с окном, определенным в аргументе hWnd. В зависимости от иерархии окон в приложении пользователь может иметь возможность продолжить работу с другими окнами данного потока. Все дочерние окна, принадлежащие родительскому окну, становятся недоступными, но может быть продолжена работа со вспомогательными окнами. Этот режим устанавливается в том случае, если в данном аргументе не установлен флаг этой группы.

MB_SYSTEMMODAL - действует аналогично флагу MB_APPLMODAL за тем исключением, что окно сообщения получает стиль WS_EX_TOPMOST. Данный флаг используется для извещения пользователя о серьезных ошибках, требующих его

немедленной реакции (например, выход за границы памяти). Этот флаг не оказывает никакого влияния на возможности пользователя по работе с окнами, не связанными с окном, определенным в аргументе hWnd.

MB_TASKMODAL - действует аналогично флагу MB_APPLMODAL за тем исключением, что в том случае, если аргумент hWnd имеет нулевое значение, то пользователь теряет возможность доступа ко всем основным и дочерним окнам, принадлежащим данному потоку. Данный флаг используется в том случае, когда приложение или библиотека не имеют дескриптора окна, но должны прекратить доступ ко всем окнам данного потока, не препятствуя работе других потоков.

Кроме перечисленных выше флагов, объединенных в группы, пользователь может использовать следующие флаги:

MB_DEFAULT_DESKTOP_ONLY - рабочий стол, получивший фокус ввода должен быть рабочим столом, выбираемым по умолчанию. В противном случае функция завершается с ошибкой. Под рабочим столом, выбираемым по умолчанию, понимается рабочий стол, появляющийся после загрузки системы;

MB_HELP - добавляет в окно сообщения кнопку Help (Справка). Нажатие кнопки Help (Справка) или клавиши отмечает соответствующее событие;

MB_RIGHT - текст в окне сообщения выравнивается по правому краю;

MB_RTLREADING - выводит текст сообщения и заголовка справа налево, как это принято в еврейском и арабском языках;

MB_SETFOREGROUND - делает активным поток, создающий окно сообщения, и устанавливает в него фокус ввода. Для этого система вызывает для окна сообщения функцию SetForegroundWindow;

MB_TOPMOST - окно сообщения создается с флагом WS_EX_TOPMOST;

MB_SERVICE_NOTIFICATION - специфический флаг для Windows NT, означающий, что данное окно вызывает служба, извещая пользователя о событии. Данная функция отображает окно сообщения на текущем активном рабочем столе даже в том случае, если на данном компьютере не зарегистрирован ни один пользователь. Если установлен данный флаг, то аргумент hWnd должен иметь нулевое значение. Это означает, что окно сообщения может появиться на другом рабочем столе, а не на том, которому принадлежит окно, определяемое аргументом hWnd.

В Windows NT версии 4.0 значение MB_SERVICE_NOTIFICATION изменено. Новые и старые значения описаны в файле WinUser.h. Операционная система Windows NT 4.0 обеспечивает совместимость снизу вверх для существующих служб, преобразуя старые значения в новые, при работе с функциями MessageBox и MessageBoxEx. Это преобразование производится только для тех исполнительных файлов, которые имеют номер версии, установленный компоновщиком, меньший, чем 4.0.

MB_SERVICE_NOTIFICATION_NT3X - специфический флаг для Windows NT, означающий значение MB_SERVICE_NOTIFICATION, используемое в Windows NT версии 3.51.

Описание

Функция MessageBox создает, отображает и осуществляет интерфейс с окном сообщения. Окно сообщения состоит из определяемого приложением текста, заголовка, предопределенного значка и набора кнопок. При использовании модального системного окна сообщения для вывода информации о том, что в системе не хватает памяти, значения аргументов lpText и lpCaption нельзя загружать из файла ресурсов (что разрушает всю концепцию использования строковых ресурсов), поскольку в данном случае попытка загрузить ресурс может закончиться безрезультатно. Если приложение вызывает функцию MessageBox и указывает в аргументе uType комбинацию флагов MB_ICONHAND и MB_SYSTEMMODAL, операционная система отображает окно сообщения вне зависимости от того, имеется ли в системе свободная память. При данной комбинации флагов

операционная система ограничивает размер сообщения тремя строками. Система не производит автоматического разбиения текста на строки. Для этого необходимо использовать управляющие символы перевода строки. Если окно сообщения создается в классе диалогового окна, то в качестве аргумента hWnd следует использовать дескриптор диалогового окна. В качестве данного аргумента нельзя использовать дескриптор дочернего окна, например дескриптор окна элемента управления. В Windows 95 максимальное число дескрипторов окон составляет 16 384.

OpenEvent

HANDLE OpenEvent(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName);

Возвращаемое значение

В случае успешного завершения функции возвращается дескриптор объекта события. Если в процессе работы функции возникла ошибка, возвращается нулевое значение. Более подробную информацию об ошибке можно получить, вызвав функцию GetLastError.

Аргументы

dwDesiredAccess - определяет режим доступа к объекту события. В системах, обеспечивающих безопасность объектов, данная функция аварийно завершает свою работу, если дескриптор указанного объекта не допускает использование указанного режима доступа для вызывающего процесса. Этот аргумент может принимать одно из следующих значений:

EVENT_ALL_ACCESS - устанавливает все возможные флаги доступа к объекту события;

EVENT_MODIFY_STATE - обеспечивает возможность использования дескриптора объекта события в функциях SetEvent и ResetEvent, изменяющих состояние объекта события;

SYNCHRONIZE - используется в Windows NT и позволяет использовать дескриптор объекта события в любой из функций ожидания для задания состояния объекта.

bInheritHandle - определяет возможность наследования возвращаемого дескриптора. Если этот аргумент имеет значение TRUE, то процесс, созданный функцией CreateProcess, может наследовать дескриптор. В противном случае дескриптор не может наследоваться.

lpName - указатель на заканчивающуюся нулем строку, содержащую имя открываемого объекта события. При сравнении имен учитывается регистр используемых символов.

Описание

Функция OpenEvent возвращает дескриптор существующего именованного объекта события. Данная функция позволяет нескольким процессам открывать дескрипторы одного и того же объекта события. Функция нормально завершает свою работу только в том случае, когда уже существует объект события с таким именем, созданный другим процессом с помощью функции CreateEvent. вызывающий процесс может использовать возвращаемый данной функцией дескриптор в качестве аргумента любой функции, использующей дескриптор объекта события, при условии соблюдения ограничений, налагаемых значением аргумента dwDesiredAccess. Дескриптор может быть дублирован с использованием функции DuplicateHandle. Для уничтожения дескриптора используется функция CloseHandle. Система автоматически уничтожает дескриптор при завершении процесса. Объект события уничтожается при уничтожении его последнего дескриптора.

RemoveFontResource

BOOL RemoveFontResource(LPCTSTR lpFileName);

Возвращаемое значение

В случае успешного завершения функции возвращается ненулевое значение. В противном случае возвращается нулевое значение. В Windows NT более подробную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`lpFileName` - указатель на заканчивающуюся нулем строку, содержащую имя файла ресурса шрифта.

Описание

Функция `RemoveFontResource` удаляет ресурс из системной таблицы шрифтов, записывая его в указанный файл. Любые приложения, добавляющие или удаляющие шрифты из системной таблицы шрифтов, извещают об этом другие приложения посылкой сообщения `WM_FONTCHANGE` всем окнам верхнего уровня в операционной системе. Для посылки этого сообщения приложение должно использовать функцию `SendMessage`, в аргументе `hWnd` которой должно стоять значение `HWND_BROADCAST`. Если на данный ресурс шрифта имеются ссылки из других приложений, данный ресурс останется загруженным до тех пор, пока не останется использующих его контекстов устройств.

ResetEvent

BOOL ResetEvent(HANDLE hEvent);

Возвращаемое значение

Ненулевое, если работа функции завершилась успешно. Если в процессе работы функции возникла ошибка, возвращается нулевое значение. Более подробную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`hEvent` - Дескриптор объекта события, возвращенный функцией `CreateEvent` или `OpenEvent`. В Windows NT этот аргумент должен иметь уровень доступа `EVENT_MODIFY_STATE`.

Описание

Функция `ResetEvent` сбрасывает отмеченное состояние объекта события. Состояние объекта события после этого остается неотмеченным до следующего вызова функции `SetEvent` или `PulseEvent`. Это неотмеченное состояние блокирует выполнение любых потоков, указавших данный объект события в аргументе функций ожидания. Данная функция используется преимущественно для ручного сброса состояния объектов. Объекты состояния, для которых определен автоматический сброс, сбрасываются после запуска первого же ожидающего потока.

SetCurrentDirectory

BOOL SetCurrentDirectory(LPCTSTR lpPathName);

Возвращаемое значение

Ненулевое, если сохранение прошло успешно. Если в процессе сохранения возникла ошибка, данная функция возвращает нулевое значение. Более подробную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`lpPathName` - указатель на заканчивающуюся нулем текстовую строку, содержащую путь в новый рабочий каталог. Этот параметр может быть как относительным, так и полным путем. В любом случае по данному аргументу определяется полный путь в каталог и запоминается как текущий каталог.

Описание

Функция `SetCurrentDirectory` изменяет рабочий каталог для текущего процесса. Каждый процесс имеет единственный рабочий каталог, имя которого состоит из двух частей:

имени диска, представляющего собой букву, соответствующую данному диску, за которой стоит символ двоеточия, или имя сервера и разделяемое имя (\\servername\sharename);

имени каталога на этом диске.

SetDIBitsToDevice

int SetDIBitsToDevice(HDC hdc, int XDest, int YDest, DWORD dwWidth, DWORD dwHeight, int XSrc, int YSrc, UINT uStartScan, UINT cScanLines, CONST VOID *lpvBits, CONST BITMAPINFO *lpbmi, UINT fuColorUse);

Возвращаемое значение

В случае успешного завершения работы функции возвращается число установленных строк. В противном случае возвращается нулевое значение. В Windows NT дополнительную информацию можно получить, вызвав функцию `GetLastError`. В Windows 98, Windows NT 5.0 и более поздних версиях: если драйвер не поддерживает работу с файлами в формате JPEG, функция возвращает код ошибки `GDI_ERROR`.

Аргументы

`hdc` - дескриптор контекста устройства.

`XDest` - содержит горизонтальную координату верхнего левого угла области вывода, измеренную в логических координатах.

`YDest` - содержит вертикальную координату верхнего левого угла области вывода, измеренную в логических координатах.

`dwWidth` - содержит ширину аппаратно-независимого битового образа в логических координатах.

`dwHeight` - содержит высоту аппаратно-независимого битового образа в логических координатах.

`XSrc` - содержит горизонтальную координату нижнего левого угла аппаратно-независимого битового образа, измеренную в логических координатах.

`YSrc` - содержит вертикальную координату нижнего левого угла аппаратно-независимого битового образа, измеренную в логических координатах.

`uStartScan` - содержит начальную строку аппаратно-независимого битового образа.

`cScanLines` - содержит число строк аппаратно-независимого битового образа, содержащихся в массиве, на который указывает аргумент `lpvBits`.

`lpvBits` - указатель на байтовый массив аппаратно-независимого битового образа.

`lpbmi` - указатель на объект структуры `BITMAPINFO`, содержащий информацию об аппаратно-независимом битовом образе.

`fuColorUse` - определяет, содержит ли переменная `bmiColors` объекта структуры `BITMAPINFO` непосредственную информацию об уровнях красного, зеленого и синего цветов (RGB) или индексы текущей реализованной логической палитры. Определены следующие значения:

`DIB_PAL_COLORS` - таблица цветов содержит 16-разрядные индексы текущей реализованной логической палитры;

`DIB_RGB_COLORS` - таблица цветов содержит непосредственную информацию о цветах палитры.

Описание

Функция `SetDIBitsToDevice` выводит аппаратно-независимый битовый образ на устройстве, связанном с указанным контекстом устройства. В Windows 98 и Windows NT 5.0 возможности данной функции расширены для обеспечения возможности вывода битовых образов в формате JPEG. Оптимальная скорость вывода битового образа достигается при использовании индексов системной палитры. Для получения информации о цветах системной палитры приложение может использовать функцию `GetSystemPaletteEntries`. Для снижения объема оперативной памяти, необходимой для вывода аппаратно-независимых битовых образов большого размера, приложение может выводить изображение частями, используя несколько последовательных вызовов функции `SetDIBitsToDevice`, помещая в массив, на который указывает аргумент `lpvBits`, различные фрагменты выводимого образа. Функция `SetDIBitsToDevice` возвращает код ошибки, если она вызывается в фоновом процессе, а активным является

приложение MS-DOS, работающее в полноэкранном режиме. В Windows 98, Windows NT 5.0 и более поздних версиях: если переменная `biCompression` объекта структуры `BITMAPINFOHEADER` имеет значение `BI_JPEG`, то аргумент `lpvBits` данной функции указывает на буфер, содержащий изображение в формате JPEG. В этом случае переменная `biSizeImage` объекта структуры `BITMAPINFOHEADER` содержит размер буфера изображения. Аргумент `fuColorUse` должен иметь при этом значение `DIB_RGB_COLORS`; если переменная `bV4Compression` объекта структуры `BITMAPV4HEADER` имеет значение `BI_JPEG`, то аргумент `lpvBits` данной функции указывает на буфер, содержащий изображение в формате JPEG. В этом случае переменная `bV4SizeImage` объекта структуры `BITMAPV4HEADER` содержит размер буфера изображения. Аргумент `fuColorUse` должен иметь при этом значение `DIB_RGB_COLORS`; если переменная `bV5Compression` объекта структуры `BITMAPV5HEADER` имеет значение `BI_JPEG`, то аргумент `lpvBits` данной функции указывает на буфер, содержащий изображение в формате JPEG. В этом случае переменная `bV5SizeImage` объекта структуры `BITMAPV5HEADER` содержит размер буфера изображения. Аргумент `fuColorUse` должен иметь при этом значение `DIB_RGB_COLORS`. Описание данной функции содержится в файле заголовка wingdi.h. При работе с данной функцией следует включить в проект библиотеку gdi32.lib.

SetEvent

BOOL SetEvent(HANDLE hEvent);

Возвращаемое значение

Ненулевое, если работа функции завершилась успешно. Если в процессе работы функции возникла ошибка, возвращается нулевое значение. Более подробную информацию об ошибке можно получить, вызвав функцию `GetLastError`.

Аргументы

`hEvent` - дескриптор объекта события, возвращенный функцией `CreateEvent` или `OpenEvent`. В Windows NT этот аргумент должен иметь уровень доступа `EVENT_MODIFY_STATE`.

Описание

Функция `SetEvent` устанавливает отмеченное состояние объекта события. Состояние объекта события, открытого в режиме ручного сброса, после этого остается отмеченным до следующего вызова функции `ResetEvent`. Пока состояние объекта события остается отмеченным может быть возобновлена работа любого количества ждущих потоков или потоков, которые последовательно запускали операцию ожидания с использованием данного объекта события. Состояние объекта события, открытого в режиме автоматического сброса, остается отмеченным до запуска первого же ожидающего потока. После этого система автоматически устанавливает неотмеченное состояние данного объекта. Если ожидающие потоки отсутствуют, состояние объекта остается отмеченным.

SetMenuContextHelpId

BOOL SetMenuContextHelpId(HMENU hmenu, DWORD dwContextHelpId);

Возвращаемое значение

Ненулевое, если функция успешно завершила свою работу, и нулевое в противном случае. Более подробную информацию об ошибке можно получить вызвав функцию `GetLastError`.

Аргументы

`hmenu` - дескриптор меню, связанного с данным контекстным идентификатором.

`dwContextHelpId` - контекстный идентификатор справки.

Описание

Связывает контекстный идентификатор справки с указанным меню. Все команды меню разделяют этот идентификатор. Контекстный идентификатор справки не может быть назначен отдельной команде меню.

SetThreadPriority

BOOL SetThreadPriority(HANDLE hThread, int nPriority);

Возвращаемое значение

Ненулевое, если функция успешно завершила свою работу, и нулевое в противном случае. Более подробную информацию об ошибке можно получить вызвав функцию `GetLastError`.

Аргументы

`hThread` - дескриптор потока, приоритет которого требуется установить. В Windows NT этот дескриптор должен иметь уровень доступа `THREAD_SET_INFORMATION`.

`nPriority` - определяет значение приоритета указанного потока. Этот аргумент может принимать одно из следующих значений:

`THREAD_PRIORITY_ABOVE_NORMAL` - устанавливает значение приоритета на 1 выше нормального приоритета для приоритетного класса;

`THREAD_PRIORITY_BELOW_NORMAL` - устанавливает значение приоритета на 1 ниже нормального приоритета для приоритетного класса;

`THREAD_PRIORITY_HIGHEST` - устанавливает значение приоритете на 2 выше нормального приоритета для приоритетного класса;

`THREAD_PRIORITY_IDLE` - устанавливает базовый приоритет, равный 1, для процессов `IDLE_PRIORITY_CLASS`, `NORMAL_PRIORITY_CLASS` или `HIGH_PRIORITY_CLASS` и базовый приоритет, равный 16, для процессов `REALTIME_PRIORITY_CLASS;`

`THREAD_PRIORITY_LOWEST` - устанавливает значение приоритета на 2 ниже нормального приоритета для приоритетного класса;

`THREAD_PRIORITY_NORMAL` - устанавливает нормальное значение приоритета для приоритетного класса;

`THREAD_PRIORITY_TIME_CRITICAL` - устанавливает базовый приоритет, равный 15, для процессов `IDLE_PRIORITY_CLASS`, `NORMAL_PRIORITY_CLASS` или `HIGH_PRIORITY_CLASS` и базовый приоритет, равный 31, для процессов `REALTIME_PRIORITY_CLASS`.

Описание

Функция `SetThreadPriority` устанавливает значение приоритета для указанного процесса. Это значение, наряду с приоритетом класса процесса потока определяет базовый уровень приоритета потока. Каждый поток имеет базовый уровень приоритета, определяемый значением приоритета потока и классом приоритета его процесса. Система использует базовый уровень приоритета для всех исполняемых потоков для определения того, какому из них необходимо выделить следующий квант времени центрального процессора. Квант времени выделяется по очереди всем потокам, имеющим одинаковый уровень приоритета, и только в том случае, если нет ни одного запроса на исполнение от потоков высшего уровня приоритета рассматриваются запросы на обслуживание от потоков более низкого уровня приоритета. Функция `SetThreadPriority` позволяет установить базовый уровень приоритета относительно класса приоритета его процесса. Например, задание уровня приоритета `THREAD_PRIORITY_HIGHEST` в функции `SetThreadPriority` для процесса, имеющего класс приоритета `IDLE_PRIORITY_CLASS`, устанавливает базовый уровень приоритета равный 6. Для процессов, имеющих классы приоритетов `IDLE_PRIORITY_CLASS`, `NORMAL_PRIORITY_CLASS` и `HIGH_PRIORITY_CLASS` система динамически повышает базовый уровень приоритета, если происходит событие, важное для данного потока. Для процессов, имеющих класс приоритета `REALTIME_PRIORITY_CLASS`

динамическое изменение базового уровня невозможно. Все потоки запускаются со значением приоритета THREAD_PRIORITY_NORMAL. Для получения и установки класса приоритета используются функции GetPriorityClass e SetPriorityClass. Для получения значения приоритета потока используется функция GetThreadPriority. Использование классов приоритета процесса позволяет разделять приложения, работающие в реальном времени, обычные и фоновые приложения. Использование значений приоритетов потоков позволяет определить важность выполнения отдельных потоков в приложении. Например, поток, выводящий окно на экран, должен иметь более высокий приоритет, чем поток, осуществляющий интенсивные вычисления. При установке приоритетов нужно следить за тем, чтобы потоки, имеющие высокий приоритет не занимали бы все имеющееся процессорное время. Потоки, имеющие базовый уровень приоритета выше 11 включаются в нормальную работу операционной системы. Использование класса приоритета процесса REALTIME_PRIORITY_CLASS может нарушить процесс кеширования диска, прекратить работу с мышью и вызвать другие подобные проблемы.

SetWindowContextHelpId

BOOL SetWindowContextHelpId(HWND hwnd, DWORD dwContextHelpId);

Возвращаемое значение

Ненулевое, если функция успешно завершила свою работу, и нулевое в противном случае. Более подробную информацию об ошибке можно получить вызвав функцию GetLastError.

Аргументы

hwnd - дескриптор окна, связанного с данным контекстным идентификатором.

dwContextHelpId - контекстный идентификатор справки.

Описание

Связывает контекстный идентификатор справки с указанным окном. Если дочернее окно не имеет контекстного идентификатора справки, то оно наследует его от родительского окна. Аналогично, если окно принадлежит другому окну и не имеет контекстного идентификатора справки, то оно наследует его от того окна, которому оно принадлежит. Это наследование контекстного идентификатора справки позволяет приложению использовать один контекстный идентификатор справки для диалогового окна и всех его элементов управления.

SetWindowLong

LONG SetWindowLong(HWND hWnd, int nIndex, LONG dwNewLong);

Возвращаемое значение

В случае нормального завершения функции - предыдущее значение заменяемой 32-разрядной величины. В противном случае - нулевое значение. Дополнительную информацию об ошибке можно получить вызвав функцию GetLastError. Если предыдущее значение заменяемой 32-разрядной величины было нулевым, то в случае нормального завершения функции возвращается нулевая величина, но функция не сбрасывает информацию о предыдущей ошибке, что существенно затрудняет вопрос о том, как завершилась работа данной функции. Чтобы избежать неопределенности необходимо вызвать функцию SetLastError(0) перед вызовом функции SetWindowLong. В этом случае об ошибке при выполнении данной функции будет говорить нулевое значение, возвращаемое данной функцией, и ненулевое значение, возвращаемое функцией GetLastError.

Аргументы

hWnd - дескриптор окна и, косвенно, класса, связанного с данным окном.

nIndex - определяет смещение величины, которую следует получить. Величина смещения может принимать значение в диапазоне от нуля до размера дополнительной памяти окна, выраженной в байтах, минус четыре. Например, при задании 12 или более

байт в качестве дополнительной памяти, величина 8 будет являться индексом третьего 32-разрядного целого числа. Для доступа к величинам, характеризующим некоторые параметры окна, существуют предопределенные величины. Ниже приведен их список:

GWL_EXSTYLE - устанавливает величину дополнительного стиля окна;

GWL_STYLE - устанавливает величину дополнительного стиля окна;

GWL_WNDPROC - устанавливает указатель на процедуру обработки окна или дескриптор, позволяющий получить доступ к этому указателю;

GWL_HINSTANCE - устанавливает дескриптор экземпляра (instance) данного приложения;

GWL_HWNDPARENT - устанавливает дескриптор родительского окна, если таковое имеется;

GWL_ID - устанавливает идентификатор окна;

GWL_USERDATA - устанавливает 32-разрядную величину, связанную с окном. Эта величина предназначена для использования приложением, связанным с окном.

В случае, если аргумент hWnd является дескриптором диалогового окна, второй параметр может дополнительно принимать следующие значения:

DWL_DLGPROC - устанавливает указатель на процедуру обработки диалогового окна или дескриптор, позволяющий получить доступ к указателю на эту процедуру;

DWL_MSGRESULT - устанавливает возвращаемое значение процедуры обработки диалогового окна;

DWL_USER - устанавливает дополнительную информацию, содержание которой специфично для каждого приложения. В качестве такой информации могут выступать указатели и дескрипторы.

dwNewLong - новое значение устанавливаемой величины.

Описание

Функция SetWindowLong изменяет атрибуты указанного окна. Функция записывает 32-разрядную величину в дополнительную память окна по заданному смещению. Функция SetWindowLong возвращает ошибку, если окно, заданное аргументом hWnd не принадлежит тому же самому процессу, что и вызывающий поток. Некоторые окна кэшируют свою память, поэтому изменения, произведенные функцией SetWindowLong вступят в силу только после последующего вызова функции SetWindowPos. Если в функции SetWindowPos используется индекс GWL_WNDPROC для установки новой процедуры обработки окна, то новая процедура обработки окна должна соответствовать формату функции WindowProc. Если в функции SetWindowPos используется индекс DWL_MSGRESULT для установки возвращаемого значения процедуры обработки окна, то после этого необходимо непосредственно возвратить значение TRUE. В противном случае при вызове любой функции посылающей сообщение вашему диалоговому окну его собственное сообщение может переписать возвращаемое значение, установленное с использованием индекса DWL_MSGRESULT. Вызов функции SetWindowLong с индексом GWL_WNDPROC приводит к созданию подкласса класса окна, использованного для создания данного окна. Приложение может создавать подклассы системных классов, но не может создавать подклассы для классов окон, созданных другими процессами. Функция SetWindowLong создает подкласс окна путем изменения процедуры обработки окна, связанной с конкретным классом окна, заставляя приложение вызывать новую процедуру обработки окна вместо старой. Приложение должно передавать все сообщения, не обработанные новой процедурой окна старой процедуре посредством вызова функции CallWindowProc. Это позволяет приложению создавать цепочки процедур обработки окна. Для резервирования дополнительной памяти окна необходимо задать ненулевое значение величины cbWndExtra, являющейся членом структуры WNDCLASSEX, используемой функцией RegisterClassEx. Нельзя вызывать функцию

SetWindowLong с индексом GWL_HWNDPARENT для замены родительского окна у дочернего окна. Вместо этого следует использовать функцию SetParent.

Sleep

VOID Sleep(DWORD dwMilliseconds);

Аргументы

dwMilliseconds - определяет промежуток времени, измеряемый в миллисекундах, на который следует приостановить исполнение данного потока. Нулевое значение данного аргумента заставляет поток передать остаток своего кванта времени любому другому потоку, имеющему одинаковый с ним приоритет и готовому приступить к работе. Если такой поток отсутствует, то функция немедленно завершает свою работу и поток возобновляет свою работу. Значение INFINITE означает бесконечную задержку.

Описание

Функция Sleep приостанавливает исполнение данного потока на указанный интервал времени. Поток может уступить остаток своего кванта времени при вызове данной функции с нулевым временем ожидания. Необходимо соблюдать известную осторожность при использовании функции Sleep в программах прямо или косвенно создающих окна. Если поток создает любое окно оно должно обрабатывать сообщения. Сообщения передаются всем окнам в системе. При использовании функции Sleep с бесконечным временем ожидания система может зависнуть. Поэтому, в тех случаях, когда поток создает окна, вместо функции Sleep следует использовать функции MsgWaitForMultipleObjects или MsgWaitForMultipleObjectsEx.

WaitForSingleObject

DWORD WaitForSingleObject(HANDLE hHandle, DWORD dwMilliseconds);

Возвращаемое значение

Если функция успешно завершает свою работу, то возвращаемое значение указывает на причину завершения работы функции. Оно может принимать одно из следующих значений:

WAIT_ABANDONED - указанный объект является мютексом, который не был освобожден потоком, которому он принадлежит, перед завершением данного потока. Принадлежность мютекса вызывающему потоку гарантируется. Поэтому этот объект остался неотмеченным;

WAIT_OBJECT_0 -указанный объект находится в отмеченном состоянии;

WAIT_TIMEOUT - истек период ожидания, а объект остался неотмеченным.

Если функция аварийно завершает свою работу, она возвращает значение WAIT_FAILED. Более подробную информацию об ошибке можно получить, вызвав функцию GetLastError.

Аргументы

hHandle - дескриптор объекта. Список типов объектов, дескрипторы которых могут использоваться в качестве данного аргумента, содержится в примечании. В Windows NT дескриптор должен иметь уровень доступа SYNCHRONIZE.

dwMilliseconds - определяет интервал времени, измеряемый в миллисекундах. По истечении этого интервала времени, если указанный объект остается неотмеченным, функция завершает свою работу. Если данный аргумент имеет нулевое значение, функция проверяет состояние объекта и немедленно прекращает свою работу. Если аргумент dwMilliseconds имеет значение INFINITE, то функция ждет отметки объекта неограниченное время.

Описание

Функция WaitForSingleObject возвращает свое значение в двух случаях: когда указанный объект устанавливается в отмеченное состояние; когда истекает время ожидания. Данная функция проверяет текущее состояние указанного объекта. Если данный объект находится в неотмеченном состоянии, выполнение потока

приостанавливается. В процессе ожидания поток практически не использует процессорное время. Перед завершением своей работы функция изменяет состояние некоторых объектов синхронизации. Изменения происходят только в том случае, если изменение состояния объекта привело к выходу из функции. Например, счетчик семафора уменьшается на единицу. Функция `WaitForSingleObject` может использоваться со следующими объектами:
извещениями об изменениях;
вводом с системной консоли;
объектами событий;
заданиями;
мютексами;
процессами;
семафорами;
потоками;
таймерами ожидания.
Необходимо соблюдать известную осторожность при вызове функций ожидания и программ, прямо или косвенно создающих объекты окон. Если поток создает любое окно, оно должно обрабатывать сообщения. Сообщения посылаются всем окнам системы. Поток, использующий функцию ожидания без интервала ожидания, может "подвесить" систему.

WinHelp
BOOL WinHelp(HWND hWndMain, LPCTSTR lpszHelp, UINT uCommand, DWORD dwData);
Возвращаемое значение
Ненулевое, в случае успешного завершения функции, и нулевое в противном случае. Дополнительную информацию по ошибке можно получить, вызвав функцию `GetLastError`.
Аргументы
`hWndMain` - дескриптор окна, из которого вызывается справка. Функция `WinHelp` использует данный дескриптор для определения того, какое из приложений запросило справочную информацию. Если аргумент `uCommand` имеет значение `HELP_CONTEXTMENU` или `HELP_WM_HELP`, то данный аргумент определяет элемент управления, по которому нужно получить справку.
`lpszHelp` - указатель на заканчивающуюся нулем текстовую строку, содержащую имя и, если это необходимо, путь к файлу справки, текст которой данная функция должна выводить на экран. После имени файла может стоять угловая скобка (>) за которой указывается имя вторичного окна, если информация выводится во вторичное, а не в первичное окно справки. Имя вторичного окна справки должно быть определено в разделе `[WINDOWS]` файла проекта справки (.hpj).
`uCommand` - определяет тип запрашиваемой справочной информации.
`dwData` - дополнительная информация. Структура данного аргумента определяется значением аргумента `uCommand`.
Описание
Вызывает справочную систему Windows (Winhelp.exe) и передает ей дополнительную информацию, определяющую характер запрашиваемой справочной информации. Прежде, чем закрыть окно, запросившее справочную информацию необходимо вызвать функцию `WinHelp` и передать в аргументе `uCommand` команду `HELP_QUIT`. До того, как все приложения не произведут эту операцию справочная система Windows не может быть закрыта. В этой операции нет необходимости, если для вызова справки использовалась команда `HELP_CONTEXTPOPUP`. В таблице П2.2 указаны возможные значения аргумента

uCommand, предпринимаемые при этом действия и соответствующий ему формат аргумента dwData.

Таблица П2.2. Соответствие значений агументов uCommand и dwData

Значения аргумента uCommand	Предпринимаемые дейст-вия	Формат аргумента dwData
HELP_COMMAND	Выполняет макрос справки или строку макроса.	Указатель на строку, содержащую имя выполняемого макроса справки. Если в строке указано несколько имен макросов, имена долж-ны разделяться точкой с запятой. Для некоторых макросов необходимо ис-пользовать короткую форму имени, поскольку справочная система Windows не позволя-ет работать с длинными именами.
HELP_CONTENTS	Выводит содержимое раздела справки, определенное ключевым словом Contents в разделе [OPTIONS] файла .hpj. Эта команда использует-ся для обеспечения совмес-тимости с предыдущими версиями. Новые приложения должны использовать файл .cnt и команду HELP_FINDER.	Игнорируется и устанавлива-ется в 0.
HELP_CONTEXT	Выводит тему справки, заданную контекстным идентификатором, определенным в разделе [MAP] файла .hpj.	Целое число без знака, содержащее контекстный идентификатор раздела.
HELP_CONTEXTMENU	Выводит меню Help для указанного окна. Затем выводит справочную информацию по выделенному элементу управления во всплывающем окне.	Указатель на массив пар идентификаторов. Первое двойное слово в каждой паре представляет собой идентификатор элемента управле-ния, а второе слово - кон-текстный идентификатор раздела.
HELP_CONTEXTPOPUP	Выводит во всплывающем окне тему справки, заданную контекстным идентификатором, определенным в разделе [MAP] файла .hpj.	Целое число без знака, содержащее контекстный идентификатор раздела.
HELP_FINDER	Выводит диалоговое окно Справочная система.	Игнорируется и устанавлива-ется в 0.
HELP_FORCEFILE	Обеспечивает вывод требуемого файла справки справочной системой. Если выводит-ся другой файл справки, справочная система выводит	Игнорируется и устанавлива-ется в 0.

	нужный. В противном случае не выполняет никаких действий.	
HELP_HELPONHELP	Если доступен файл WINHLP32.HLP, выводит справочную информацию по пользованию справочной системой.	Игнорируется и устанавливается в 0.
HELP_INDEX	Выводит содержимое раздела справки, определенное ключевым словом Contents в разделе [OPTIONS] файла .hpj. Эта команда использует-ся для обеспечения совмес-тимости с предыдущими версиями. Новые приложения должны использовать файл .cnt и команду HELP_FINDER.	Игнорируется и устанавливается в 0.
HELP_KEY	Выводит тему справки по ключевому слову, содержащемуся в таблице ключевых слов в том случае, если обеспечено полное совпадение. Если найдено более одного раздела, соответствующего данному ключевому слову, выводит диалоговое окно Найденные разделы, содержащее список найденных разделов.	Указатель на строку, содержащую ключевое слово. Несколько ключевых слов должны разделяться точкой с запятой.
HELP_MULTIKEY	Выводит тему справки по ключевому слову, содержащемуся в альтернативной таблице ключевых слов.	Указатель на объект структуры MULTIKEYHELP, определяющей символ нижнего индекса таблицы и ключевое слово.
HELP_PARTIALKEY	Выводит тему справки по ключевому слову, содержащемуся в таблице ключевых слов в том случае, если обеспечено полное совпадение. Если найдено более одного раздела, соответствующего данному ключевому слову, выводит диалоговое окно Найденные разделы.	Указатель на строку, содержащую ключевое слово. Несколько ключевых слов должны разделяться точкой с запятой. Для вывода содержания без указания ключево-го слова необходимо пере-дать указатель на пустую строку.
HELP_QUIT	Сообщает справочной систе-ме Windows о прекращении работы с ней. Если со справочной системой не работают другие приложения, закрыва-ет справочную систему Win-dows.	Игнорируется и устанавливается в 0.

HELP_SETCONTENTS	Определяет содержимое раздела Contents. Справочная система Windows выводит эту тему, если файл справки не имеет связанного с ним файла .cnt.	Целое число без знака, содержащее контекстный идентификатор раздела Contents.
HELP_SETPOPUP_POS	Устанавливает позицию всплывающего окна. Позиция всплывающего окна устанавливается таким образом, как будто бы указатель мыши располагался в указанной точке при вызове данного окна.	Указатель на объект структуры POINT.
HELP_SETWINPOS	Выводит окно справки, если оно было минимизировано или располагалось в памяти.	Указатель на объект структуры HELPWININFO, определяющей размер и положение первичного или вторичного окна справки.
HELP_TCARD	Указывает на то, что данная команда относится к последовательности вторичных окон. Данная команда комбинируется с другими командами с использованием операции логического ИЛИ.	Зависит от команды, с кото-рой объединена данная ко-манда.
HELP_WM_HELP	Выводит во всплывающее окно справку об элементе управления, указанном в аргументе hWndMain.	Указатель на массив пар идентификаторов. Первое двойное слово в каждой паре представляет собой идентификатор элемента управле-ния, а второе слово - кон-текстный идентификатор раздела.